AnalyzeIT™

Automated Analysis of Legacy Applications

White Paper

October 1999

Intercomp Ltd.
6 Maskit Street
Herzliya 46733
Israel

Tel: +972-9-9526777
Fax: +972-9-9526170
E-mail: info@cobol2java.com
Web Site: www.cobol2java.com

Contents

About Intercomp Ltd.

Intercomp Ltd. is a subsidiary of Crystal Systems Solutions (NASDAQ: "CRYS") and a Formula Group company (NASDAQ: "FORTY").

Intercomp's line of products is based on a set of compiler-centric software tools that provide a comprehensive automated solution for the migration of legacy mainframe and non-mainframe COBOL applications to Java N-tier client/server e-commerce applications.

The set of products, certified by Sun Microsystems as "100% Pure Java," offers its users the ability to analyze their legacy systems, reengineer and convert them into a new N-tier paradigm.

About Crystal Systems Solutions Ltd.

Crystal Systems Solutions Ltd., and its subsidiaries, provide Fortune 500 companies, and others, with assessments, conversion methodology, software and professional services for all their mainframe and non-mainframe based activities with minimum investment in manpower, resources, and time.

Partial Customer List: FORD Motor Co., Kraft Foods, Farmer Insurance Group, KeyCorp, MCI, BMW, MetLife, United Technologies Pratt & Whitney, Blue Cross / Blue Shield, Phillips Petroleum, Ralston Purina, Alex Laurie Factors (UK), Reynolds Metal, Medical Mutual of Ohio, El Al Israel Airlines, Bank HaMizrahi, Bezeq (Israeli PTT), Bank Leumi–Israel

Crystal Systems Solutions works both independently, and in conjunction with local business partners worldwide who provide project management and system integration activities in conjunction with its products. The list of business partners includes Ernst & Young LLP, Logica Ltd., EDS, SAIC, CGI Informatik GmbH and others.

About the Formula Group

Formula Systems Ltd. (1985) is an information technologies company principally engaged, through its subsidiaries and affiliates, in the development and marketing of proprietary software products and in providing information systems solutions. Formula Systems has a number of subsidiaries which are traded on the Tel-Aviv Stock Exchange (TASE), as well as subsidiaries which are traded on NASDAQ. Additionally, Formula Systems itself is traded on the TASE and has ADR's listed on NASDAQ. Between 1994 and 1998 Formula Systems' revenues grew to over $250 million making Formula Systems the largest publicly-traded software group in Israel. Formula Systems currently employs a staff of over 3,500 employees of which more than 3,000 are computer experts.

Strategic Partnerships of the Formula Group: Formula Systems has developed partnerships with leading players in a broad range of fields. Formula Systems' strategic partners include, among others, BAAN Company, Cincinnati Bell information Systems Inc., Cooper Industries Inc., Ernst & Young LLP and Informix Software Inc.

Key Customers of the Formula Group: ABB, Amdocs, AT&T, Bell Atlantic, Bezeq, Coca-Cola, Comverse, Deutsche Telekom, Dow Chemical, ECI Telecom, Elect. de FranceEskom, FAA, Ford GEC Alsthom, Informix, Israel Electric Corp., Israel Air- Force, Keycorp, Krupp Kraft, Scitex, Telecom Asia, Telstra, Tivoli, Wharf Cable.

1. Legacy System Challenge

The Problem

Effective management of the complex legacy system environment is a challenge, given the norms of employee turnover rates. Decisions about modernizing these systems, controlling their performance, and handling changes and enhancements are tough to make without thorough acquaintance with the underpinnings of the legacy code.

IT managers are concerned today with giving their organization the capacity for faster reaction to changing market conditions and competition, for easier access to information and data, and for lower cost of business processes and IT development. These concerns touch two major aspects of the IT world: its maintenance on the one hand, and it renovation on the other hand.

Understanding the applications and the systems that exist in the organization, making decisions based on information and knowledge, and simplifying the tasks of maintenance and development would enhance an organization's competitive ability.

Analysis Tools vs. Y2K Tools

Most organizations are well acquainted by now with the inventory, search and replace capabilities of the Y2K tools they have been using for solving this problem. An important question is whether or not these tools are able to perform tasks that are needed for the daily maintenance of such applications and for their reengineering for the purpose of renovation and modernization.

When developing a product, engineering is usually done in the best and shortest method to get the product "out the door." Once the basic needs are met by the product, incremental engineering is a luxury most organizations can not afford. The capabilities that are needed for handling the Y2K problem were the ability to find all the sources of the application, then search the application for date specific fields, and then change the dates to be Y2K compliant. Using tools with greater capabilities would have been "overkill" concerning these objectives.

Nevertheless, in the post-Y2K world reengineering of the mainframe environment is one of the issues that must be considered. In some cases the organization will need to renovate the data model, in some cases it is the user interface, in some cases it is the platform technology base, and in some cases new application functionality needs to be added. In assessing a code base, IT departments often do not fully understand the functionality of legacy systems. A tool is required to analyze and document this functionality, and to describe to users the exact structure and complexity of mission-critical systems. By having a picture of the application, it is possible to "see" the code in a new and clearer way. To be able to trace the code graphically and be able to look directly at the sources of that code and trace its flow can be an important capability for design, development and QA departments.

2. AnalyzeIT Solution

Intercomp's **AnalyzeIT** assesses legacy applications, creates a repository and inventory of the applications' components, and establishes the relationships between them. The tool collects the legacy information into the inventory, analyzes it and stores the results in a repository, and presents it in a user-friendly way with various graphs and reports.

The **AnalyzeIT** process can assist everyday legacy maintenance, precede a process of legacy migration, component mining, or even rewriting the legacy application, since it supports the decision making process by providing understanding of the legacy system.

Compiler-Centric Approach

The capabilities of **AnalyzeIT** to collect and extract information about an application, to examine it carefully, and to present it in a coherent manner, are based on an engine which is essentially a "smart compiler." This compiler-centric approach gives **AnalyzeIT** an automated capability to analyze and understand an entire system along with all of its components.

The kernel of the **AnalyzeIT** product is a large multi-phase compiler package that supports a wide range of COBOL dialects including embedded software packages such as CICS (Customer Information Control System), SQL (Structured Query Language), DDL (Data Definition Language), BMS (Bitmap Screens), etc.

The basic translation scheme is syntax directed and uses mapping techniques in order to achieve maximum clarity and efficiency in the resulting Java source code.

The compiler builds large data structures that semantically describe the application. This description can be saved as an IDE (Integrated Development Environment) file, which enables the user to refine or change the description and perform the translation at a later time.

There are multiple information-collecting phases that take place during compilation: lexical, syntactic and semantic. The final phase performs the actual translation of the application into Java using the results of the previous phases. The original source code is significantly improved by means of the translation schemes used by the translator in conjunction with the information collected.

Lexical Analysis

The input source code file is lexically analyzed. The lexical analyzer partitions the input stream into character strings by matching them with a set of predefined regular expressions. The constructed linked token sequence provides a list of terms that is based on the source file. In addition, each token object contains the exact location of its string; the complete token sequence is used as a base leaf sequence for the parsing tree.

The set of predefined regular expressions is large enough to match a wide range of COBOL terms from different versions of the language including special forms of embedded statements, such as CICS, SQL, etc.

The lexical analyzer does not impose strike rules on the source program format, and relates to it as a virtual space with strings to be matched. However, it does recognize and handle the various areas (sequential number, area A and B, etc.).

Together with the symbol table, the lexical analyzer splits the terms from the input program into syntactic categories. Special internal terms such as white space and comments are also linked into the token sequence but are not passed to the syntactic analysis phase.

At the end of the process this sequence is actually a map of the source code file in the main memory. This sequence is used as the basis for most of the operations that will process this file.

Syntactic Analysis

The purpose of this unit is to build a parse tree of the source code file based on the selected token provided by the lexical analyzer.

The syntactic analysis is conducted by a large set of context free grammars. This collection of grammars is designed to catch a wide range of COBOL versions, COBOL dialects and embedded statements from other supplementary packages such as CICS, SQL, etc.

The parse tree nodes are designed using a special strategy that expresses grammatical equivalence and reduces and simplifies the semantic analysis and therefore the translation process and results.

The parse tree is fully object oriented and implements smart inheritance relations between its nodes. The main principle of the strategy is to design the inheritance relation in such a way that properties with major semantics will be at the bases of the inheritance structures, while nodes that add minor semantics will be derived from these bases. This strategy also helps to split the different parts of the code between different nodes thus helping to build good structured software, where each node is an object that knows its own semantics.

As in the previous phase, this phase also creates a data structure, i.e., a parse tree for later use. Splitting the code into different objects simplifies the translation scheme and improves code readability and reusability.

Semantic Analysis

Semantics analysis captures the semantics of the different object structures, the meaning of their position in the source code file and the way they relate to each other. An additional purpose is to understand the ideas that hide behind the code. These ideas are expressed by the parse tree and the token sequence.

Understanding the programming techniques that were used by the original COBOL programmers and implemented in the code can help later in the translation phase by producing much more readable and efficient Java code.

Benefits of the Compiler-Centric Approach

There are three main reasons why it is useful for the compiler to understand the programming techniques that were used in the source program:

- *To facilitate the change in the software architecture*. Legacy COBOL source programs were programmed using concepts of structured procedural programming style while Java source code is based upon object-oriented programming and event-driven programming style.
- *To retain the ideas behind the code*. This information helps in generating new code that keeps these ideas and by doing so makes it easier for the programmers to understand the differences between the old and the new parts of code. In addition it helps them to find their way through the many newly generated lines and modules.
- *To improve the methods used by the original programmers*. Not all programs were written clearly and efficiently. Furthermore, during years of maintenance, COBOL code frequently becomes a patchwork of fixes and original written in different styles. However, grouping the knowledge of programming techniques in sophisticated software can help to transform inconsistent techniques into consistent and efficient ones.

3. AnalyzeIT Process

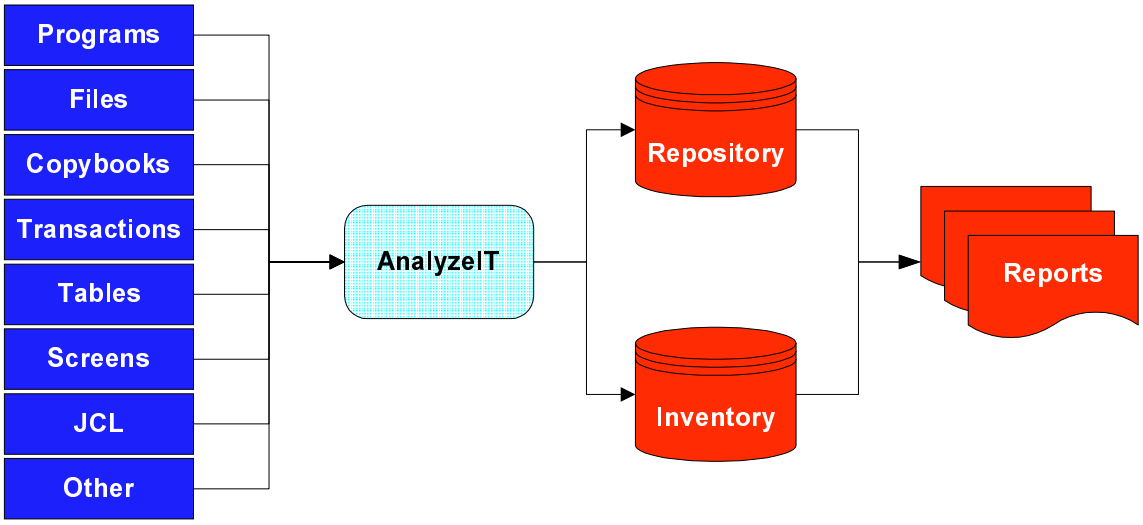

Figure 1. AnalyzeIT Process Flow

Collecting and Analyzing the Information

The collection of information for an application or an entire environment comprises three major steps that are performed by **AnalyzeIT**:

Inventory

The first step of the **AnalyzeIT** process is to collect an inventory of all components in an application, and to perform an inspection of the application's integrity.

For example, a typical COBOL application is composed of:

- COBOL programs,
- Screen definitions (such as BMS or MFS),
- Copybooks,
- File or table definitions,
- Batch scripts (such as JCL), and
- Other items depending on the environment of the application (for example FCT and PCT tables for CICS applications).

AnalyzeIT assesses the application components and lists them in the inventory tables. Moreover, **AnalyzeIT** determines if application entities are needed but are missing, such as a called program, a displayed screen, or an accessed file or table definition. By using the built-in reports provided by **AnalyzeIT**, the user is able to determine what is missing and from where it is referenced.

Analysis

The second step in the **AnalyzeIT** process is to parse the application's components and to verify that they are syntactically and semantically correct. Each of the entities is put through a compilation process; code fragments that are rejected from this process are displayed on the screen or in a report. These code fragments may originate, for example, from proprietary software components developed in-house by the customer. To manage these rejections, **AnalyzeIT** uses the "Decisions and Rules Mechanism" to control the **AnalyzeIT** behavior. Once the parsing and analysis are over, the tool prepares a final repository of the application.

Repository

After all of the entities are examined and verified, a repository is generated using a standard database (such as Oracle, Access, etc.), containing all of the information extracted from the application. The repository contains information about the application's components, the relationships between these components, abstracted semantic information about each program's behavior, the usage of data items inside programs, control flow and more. The information in the repository can be accessed using **AnalyzeIT**, or simply using any reporting tool that can access a standard relational database. The following screen is an example of a repository stored in an Access database, including the interrelationships between the tables:

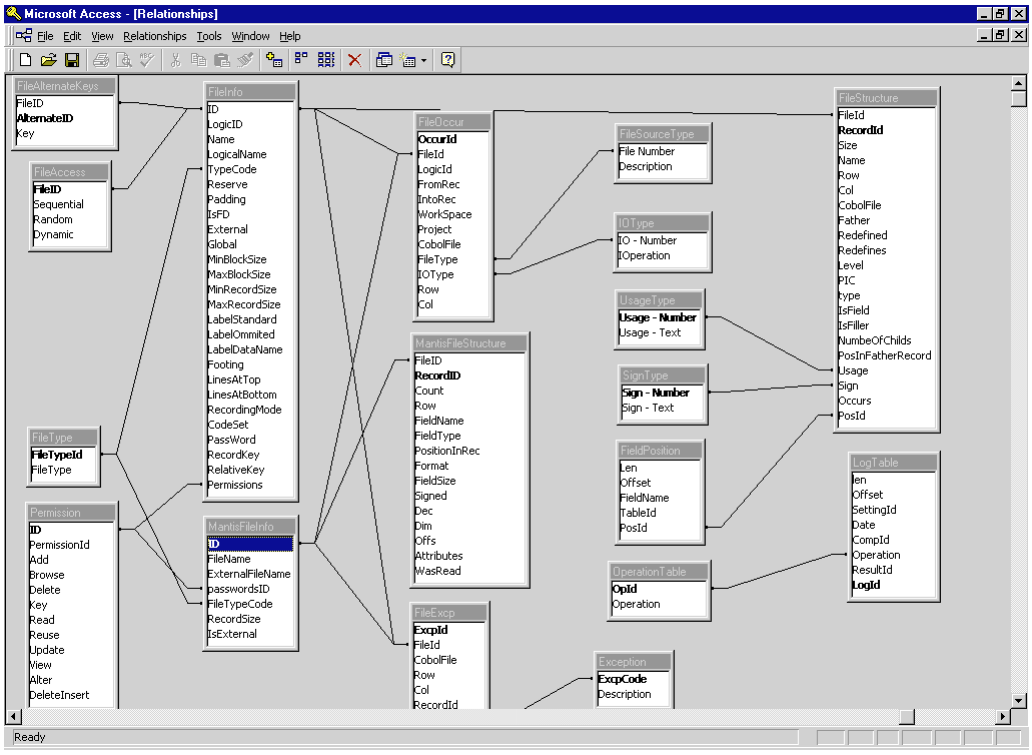

Figure 2. AnalyzeIT Repository (Access)

Presenting the Information

In order to perform an analysis of the application, the **AnalyzeIT** tool offers several options:

Graphs

There are several graphs generated from the repository of the application:

- *Cross application flow* – This graph shows the flow between different programs in the application. Programs and screens are nodes in the graph, while links between programs and screens are the edges. Edges can be the following statements: CICS LINK, CICS XCTL, CICS RETURN, COBOL CALL, CICS RECEIVE MAP, CICS SEND MAP, etc.

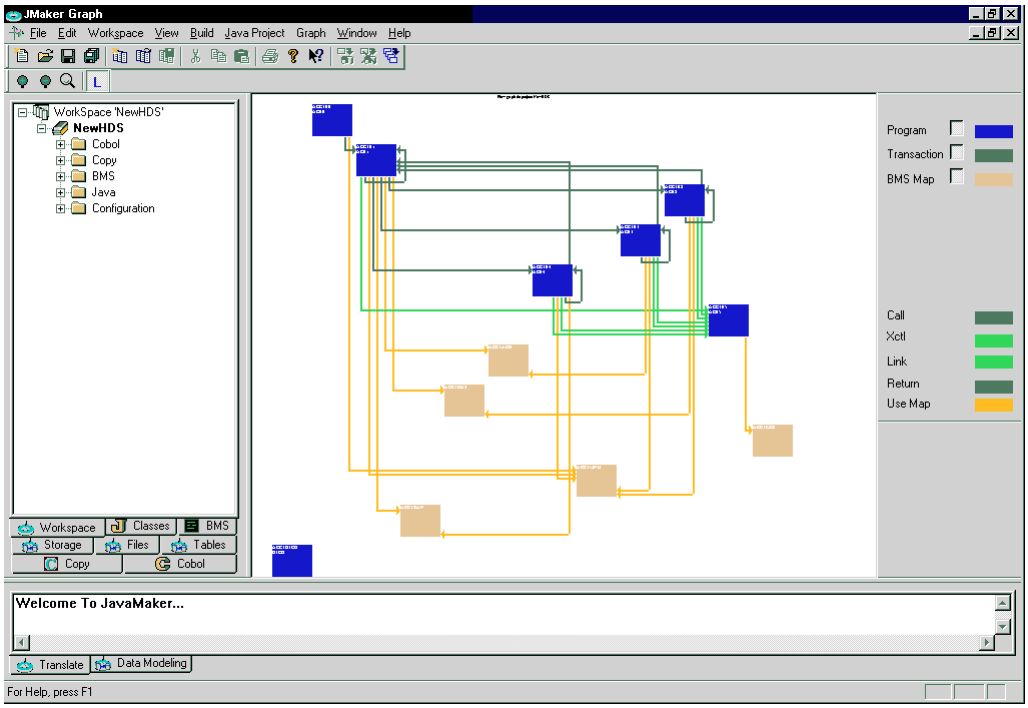

Figure 3. Cross-Application Flow Graph

- *Inner program flow* – This graph shows the flow inside a program. The nodes of the graphs are paragraphs, as well as programs, screens, tables and files. The edges can be any statement that changes the flow between the paragraphs, such as GOTO and PERFORM, as well as the statements that transfers the flow to external programs such as LINK, RETURN, CALL and so on. The graph also shows where screens, tables, and files are used in the program.

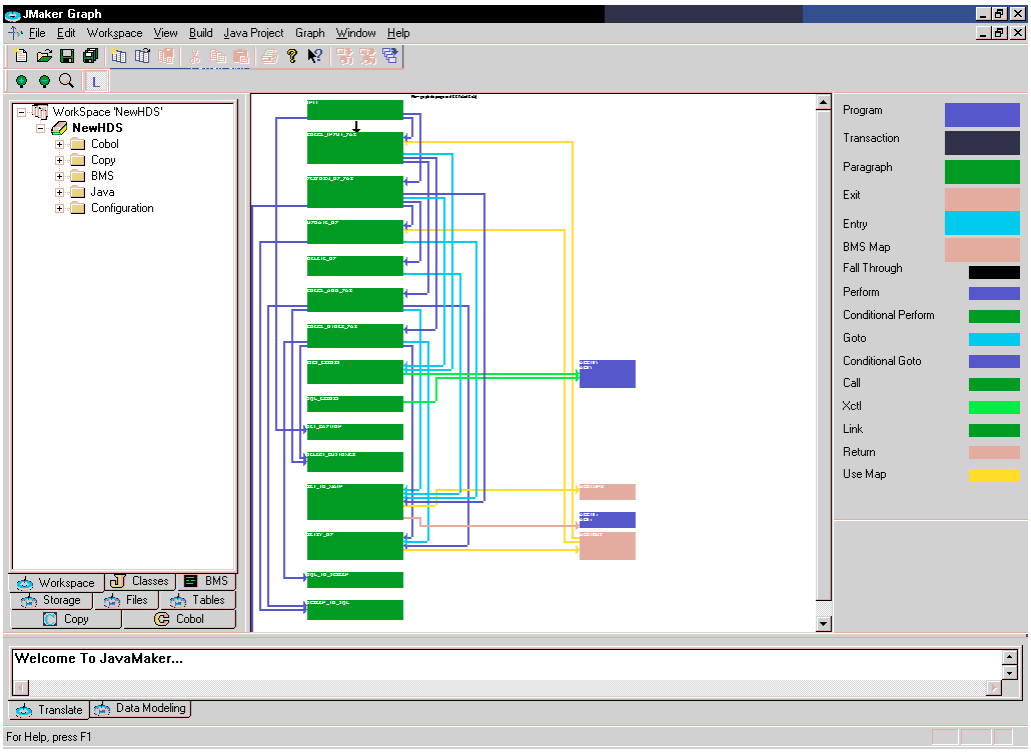

Figure 4. Inner-Program Flow Graph

- *Data flow* – This graph shows the way specific data is passed inside a program or between programs.
- *Resource graph* – This graph shows the usage of resources for each program. Typical resources are files, tables, screens, copybooks, temporary storage of CICS and so on. Information about the resource is extracted from the programs and from JCL, CICS tables and so on.

Reports

The analysis reports are separated into four categories:

- *Usage reports* – These reports specify usage details for each item. An example is a report listing all statements in which a specific variable is used, or all programs (including specific lines inside the program) in which a screen or a file is used, and so on.
- *Resource reports* – Similar to the resource graph, these reports list all of the resources of a program, a group of programs or even specific paragraphs.
- *Statistics reports* – These reports provide statistical information such as how many lines exist in the whole application, how many times a specific screen is used in the application, which file is the most used, which program has the most I/O statements, what is the ratio between I/O and logic in a specific program and so on.
- *Dynamic reports* – These reports are user-defined reports that can include any of the information found in the above three categories of reports. They provide the flexibility to focus on specific information according to the user's needs.

Tree Views

Tree views are used to display the following information:

- *Relationships* – Shows which program uses a specific copybook, which copybooks are used by a specific program, which variables are used in a program and so on.
- *Tree to source* – Enables the user to move from the tree view to the source and back, depending on the information. When browsing variable information, you can move to the declaration part of the variable, or to sentences that use it and so on.

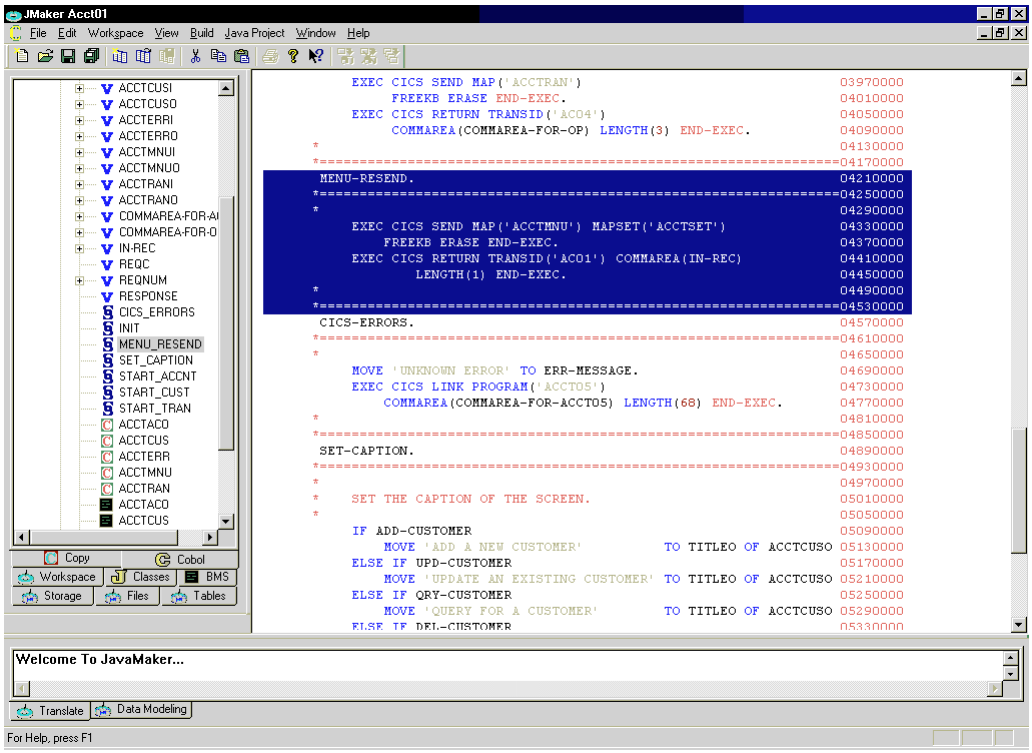

Figure 5. Tree Views

- *Paragraphs* – Shows the tree view of a paragraph and includes information about the variables used in the paragraphs; the same for screens, files, and tables, etc.
- *Resources* – For each resource the tree view displays information about relationships between the source and the application (declaration and usage).
- *Data elements* – For each data element, the tree view displays declaration and usage information. In addition, its hierarchical position relative to other data elements is shown.

Code Analysis

AnalyzeIT offers several optimizations that can be performed on the application:

- *Pattern matching* – Enables you to try and match a specific programming sequence to similar patterns found in the application. This process is useful when trying to find duplicate code parts that can be joined.

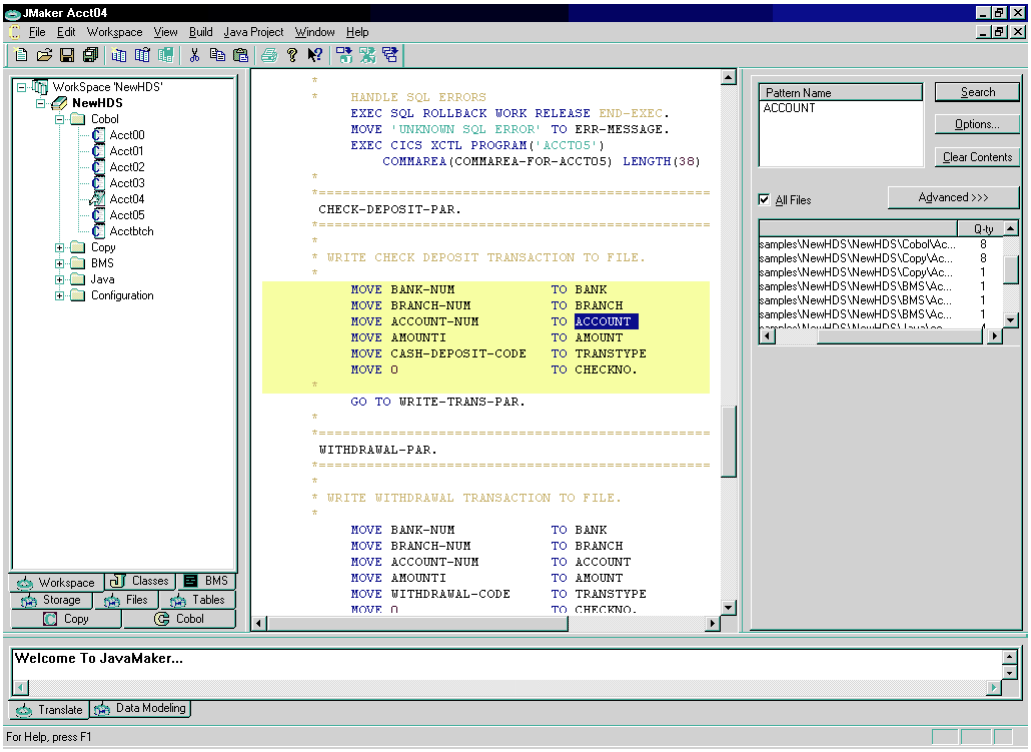

Figure 6. Pattern Matching Code Analysis

- *Redundant analysis* – Enables you to find redundant parts in the application. Redundant code can include variables declared but not used, paragraphs that are not reached, files or tables that are declared but not used and so on.

4. Advantages of Using Intercomp's AnalyzeIT

The first and most direct result of the **AnalyzeIT** process is the ability to "view" an entire system graphically and systematically. As previously described, this overall view is given by means of graphical presentation and reporting methods. This combination provides the users of **AnalyzeIT** increased control over their systems: they are now able to point out missing components of the system, to locate unused or unreferenced code segments, to produce quantity reports of various system components, etc.

Change management has always been a difficult issue for IT maintenance. The control **AnalyzeIT** provides over the system's internals, makes changes and enhancements to a system an easier and simpler process to manage. Working on the basis of accurate information about the system's components is now an integral part of decision making.

Another major benefit of the **AnalyzeIT** process is the asset of the inventory and the repository that are created as a result of the process. Such documentation of a system, which is centralized in one standard database, contains the organizational body of knowledge. It can be used as the core and the "gate-keeper" for maintenance of the applications and for their future development.

The benefits of **AnalyzeIT** also apply to the redesign and reengineering of existing systems. These aspects of development work are easier and faster if they rely upon an accurate view of the current system structure.

Intercomp Ltd.

6 Maskit Street
Herzliya 46733
Israel

Tel: +972-9-9526777
Fax: +972-9-9526170
E-mail: info@cobol2java.com
Web Site: www.cobol2java.com