

**The Metaphor Project Summary Report:
Technology for Analyzing Change and Composing
Reusable, Real-Time Components and Applications**

Carol L. Hoover
Project and Technical Lead
clh@cs.cmu.edu

Pradeep K. Khosla
Principal Investigator
pkk@ices.cmu.edu

CMU-RI-TR-98-29

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

December 1998

©1998 Carol L. Hoover and Pradeep K. Khosla

The Metaphor Project effort was sponsored by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Material Command, USAF, under agreement number F30602-96-2-0240. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, Rome Laboratory, or the U.S. Government.

Abstract: There is an increasing need for software systems to adaptively support changes in application-level objectives [Fayad96]. Real-time software evolution involves changes in software structure and meaning over time to satisfy changes in application requirements. The problem is that the process of changing real-time software often involves extensive impact of change (non-localized change) and substantial manual effort. The process can be costly and error-prone. The Metaphor Project hypothesis was that a systematic approach to thinking about change used with an analytical method for “localizing” software solution features that would be impacted by the same changes can result in a significant reduction in the effort needed to modify these solutions over time. The proposed work was the research and development of a model for organizing information about change, of algorithms to systematically and automatically localize change to basic solution features such as data/operations and control flow, and of a process for analyzing the basic features with respect to change and reuse. This report summarizes the Metaphor Project objectives, accomplishments, and technical as well as educational contributions.

Table of Contents

1.	Introduction	7
2.	Project Goals and Objectives	7
3.	Accomplishments and Demonstrations	7
3.1	Year 1 (July 1996 Summary Report)	8
3.2	Year 2 (July 1997 Summary Report)	8
3.3	Year 3 (July 1998 Summary Report)	8
4.	Technical Contributions	9
4.1	State of the Practice	9
4.2	Conceptual Results	9
4.3	Description of Tools	12
5.	Lessons Learned	13
6.	Educational Contributions	14
7.	Summary and Future Work	14
	Acknowledgments	15
	Appendix A: Accomplishments With Respect to the Statement of Work	15
	References	17

List of Figures

4.1	Model for Change Information Base (CARAT)	10
4.2	Componentization Based on Change and Data-operational Dependencies	11
4.3	Componentization Based on Invariance in Control Flow	11
4.4	Partitioning of Basic Software Solution Elements	12
4.5	Interface to the Change Information Manager (CIM)	13

1. Introduction

The Metaphor Project hypothesis was that a systematic approach to thinking about change used with an analytical method for “localizing” software solution features that would be impacted by the same changes can result in a significant reduction in the effort needed to modify these solutions over time. We hypothesized (1) that historical information about change can help facilitate predictions of feasible or expected future changes and (2) that it is feasible to develop algorithms for partitioning software solutions into components that would localize the impact of these changes. From our experience in developing factory automation and robotics systems, we rationalized that there are generic as well as domain-specific types of required changes. We also thought that the commercial emphasis on “software component-based” development and the rising importance of evolvable product-line architectures motivated the need for automated techniques to help the software engineer to determine components that would not only maximize the potential for reuse but also simplify change by localizing its impact.

The purpose of the Metaphor Project was therefore threefold: (1) to develop a rationale for thinking and capturing knowledge about change in real-time software solutions (including change to support adaptable performance) and a process for creating taxonomies of anticipated changes, (2) to research and develop systematic and analytical techniques for integrating knowledge about change into the design of reusable software solutions that minimize the impact of change, and (3) to demonstrate the use of tools to support the first two objectives.

This report reviews the Metaphor Project goals and objectives, reports objectives that were accomplished during the funding period (1 July 1996 to 30 June 1998), discusses the project’s technical contributions to the field of software design as well as lessons learned, and briefly outlines future research.

2. Project Goals and Objectives

The Metaphor Project researched and developed analytical techniques to understand, to design for, and to better manage change in architecturally similar real-time software solutions. Our objectives were to research and develop qualitative and quantitative techniques that would help the software engineer to systematically:

1. Identify and analyze the types of changes likely to be made to software solutions for real-time applications.
2. Use knowledge about change, reuse, and adaptation requirements to partition software solutions into adaptable, evolvable, and reusable software components for architecturally similar real-time applications.
3. Map application requirements to reusable software solutions.

The scope of our research was the transformation between requirements analysis and design. Our approach was to mathematically model and automate parts of this transformation. Our research focused on basic attributes of a software design that software architects use to partition a solution into components. Our goal was to develop analytical methods for quantified comparisons of alternative partitions with respect to these attributes and with respect to design criteria such as evolvability, reusability, and adaptable performance. The complementary project goal was to formulate an organizational structure to categorize knowledge and information about change and to develop a tool that demonstrates the automatable features of collecting and organizing this data.

The detailed project objectives are listed in Appendix A. Via quarterly reports, we charted our progress with respect to these objectives.

3. Accomplishments and Demonstrations

The Metaphor Project developed analytical methods for partitioning data and operations as well as control flow with respect to reusability and evolution. Our methods guide the designer to decompose a software solution with respect to reuse and to group the decomposed solution elements with respect to anticipated or feasible changes to the required behavior or architectural design. We developed a model for collecting and organizing information about change. Our techniques and tools enable the semi-automatic use of this information to generate a software architecture that satisfies design constraints such as reusability and evolution.

The sections which follow describe the accomplishments that we reported in the DARPA summary reports for each year that the Metaphor Project was under contract. The reports are shown in chronological order. Detailed accomplishments are listed in Appendix A: Accomplishments With Respect to the Statement of Work.

3.1. Year 1 (July 1996 Summary Report)

The Metaphor Project was a new start in July 1996.

3.2. Year 2 (July 1997 Summary Report)

We developed an analytical method for partitioning software solutions to minimize the impact of change and to maximize reuse across architecturally similar solutions. Our method can be applied to the following different software design approaches: object-oriented design, design of abstract data types, and algorithm decomposition.

We developed an analytical method for designing control flow components that minimize potential changes to the required control flow of high-level tasks. This method applies to a master control type of architecture as well as to the flow of control in an object-oriented architecture.

We developed a model for creating a change knowledge base and a physical design for implementing a relational database version of the knowledge base.

This model and implementation supports the collection, organization, and reuse of information about software change.

We completed the first prototypes of our ACT and CARAT tools. We demonstrated these tools at the DARPA/EDCS Workshop in July 1997. These tools support the automatable features of our partitioning and change information management technologies.

3.3. Year 3 (July 1998 Summary Report)

We enhanced our change information model for the change analysis rationale and technology (CARAT) to include project tracking data and implemented these features in the CARAT tool.

We developed the change information manager (CIM) tool that enables a change manager (human) to:

1. Define the terms used to describe information about software change (create a change information configuration).
2. Create workspaces which are the logical work areas of projects and organizations and which are implemented as physical groupings of information records in a change knowledge base. The workspace also provides a logical and physical grouping of information used to partition software solutions into components.
3. Assign a predefined configuration to project or organizational workspaces.
4. View, modify, and delete configurations.
5. Create reports regarding workspaces assigned to configurations, keywords assigned to configurations, and existing configurations.

We modified the CARAT tool to allow the user to select change information keywords from pre-defined lists (configured by the change manager) or to dynamically define new keywords.

We modified the analytical composition tool (ACT) and the CARAT tool to require the user to select from a list of existing workspaces. The human change manager creates the workspace(s) for his/her project or organization.

We investigated existing commercial tools for change management and software design to update our knowledge of the state-of-the-art in software development tools.

We combined our partitioning methods into one method and submitted for publication a paper that overviews current software design methods and that describes our combined method [Hoover & Khosla, 1998].

We applied our partitioning and change analysis methods to the design of evolvable and reliable software tools for designing microelectromechanical (MEMS) devices. MEMS software design tools are applications which should enable the hardware engineer to create designs for MEMS device that will function properly with as few necessary

prototypes as possible and that will operate reliably. These CAD applications are expected to evolve significantly over the next decade to support automated design of critical MEMS devices such as accelerometers. Our study resulted in a way to partition these CAD tools into components to support the anticipated evolution.

4. Technical Contributions

The primary contribution of this project is the development of a systematic and precise way to analyze basic elements of a software solution with respect to the design criteria of changeability and reuse. The basic principle is that knowledge about change, reuse, and adaptation (either historical or feasible and anticipated) can help to guide the software engineer in the partition of a software solution into components. Previous and current research supports this principle. As early as 1969, McIlroy discussed the construction of software components to create adaptable software systems [McIlroy, 1969]. Parnas presented guidelines for modularizing software systems in two notable papers [Parnas, 1972] and [Parnas et al., 1984] as well as in the A-7E case study [Britton & Parnas, 1981]. More recently, Westerberg as well as Weide and his colleagues studied ways to modularize algorithms to enhance their reuse [Westerberg, 1989] and [Weide et al., 1994]. Stewart under the direction of his advisor Professor Khosla developed a software design and operating environment for the construction and execution of reconfigurable and reusable sensor-based modules [Stewart, 1994]. Our work extends previous work through the development of systematic and mathematically-based ways to determine a partition of functionality into modules to satisfy design goals such as reuse, evolution, and adaptability.

In the following subsections, we briefly describe the state of commercial software design and management tools, the conceptual and theoretical results of our research, and the tools that we developed to demonstrate the research concepts.

4.1. State of the Practice

In our review of some widely-used commercial computer-assisted software engineering (CASE) tools, we found that most tools help the software analyst or designer to diagrammatically and textually document software usage scenarios and elements of software design such as objects and their interactions, state behavior and concurrency, data/control flow, and class/procedure/module structures. Other tools help the software engineer to manage versions of software systems and to track changes made to code (usually at the granularity of files). Some research tools help engineers to restructure code written in a specific language. Many of these tools are based on design concepts developed by researchers and designers such as [Booch, 1994], [Selik et al., 1994], and [Ward & Mellor, 1985]. More recently, research systems that pictorially represent codified knowledge about architectural styles and design patterns are emerging. These tools are based on work by researchers such as [Buschmann et al., 1996], [Gamma et al. 1995], and [Shaw & Garlan, 1996]. Another approach involves techniques to evaluate proposed software architectures with respect to desired properties. One such technique is the Software Architecture Analysis Method (SAAM) as discussed in [Bass et al., 1998]. From the project management point of view, there are commercial tools that help managers to delineate and track project activities, milestones, work units, completion dates, etc. Some research tools help the members of a project team to annotate and review issues related to a particular decision that must be made collectively.

The interesting fact about the current software design tools is that though they help the software engineer to document a design and possibly to select from codified designs, they do not really help the designer to systematically generate new or alternative design that satisfies design objectives. One might argue that component (code) generators synthesize software solutions. Code generators are a form of codified design in which the solution for a finite set of problems is determined ahead of time. Given the description of a predetermined problem, the computer is programmed to generate the correct solution. This process can be helpful for a well-known application in which alternative uses or configurations are fixed and encoded in the design. But for new problems and application domains, the software designer would like a tool that helps him/her to generate alternative designs that satisfy varying design objectives. The synthesis of a design (especially at the architectural level) requires the designer to decompose and group (partition) elements of the software solution such as data/operations and control. The Metaphor Project work involved a study of ways to systematically group these elements to achieve partitions that localize for change, reuse, or adaptability.

4.2. Conceptual Results

The information model that we developed to organize information about change is shown below in Figure 4.1. Our model is based on the concept that for every historical or anticipated change there is a stimulus for the change (such as a particular functionality that must change), an object of change (such as a particular component, module, or file that must be modified or replaced), and a more detailed reason for the change (such as a change in requirements

or a need to change to a different execution platform). The change information record contains data about the type of change, the “behavioral” characteristics of the change, a complementary description of the change, and pointers to other related change records. We have annotated the information records with the author and date of the change. This model is generic and can easily be tailored for hardware or software development. Each attribute is associated with one or more keywords to describe the stimulus, object, or reason of change. A change information manager could specify the appropriate types and keywords to be used by the organization. This would help to standardize the communications between the development engineers.

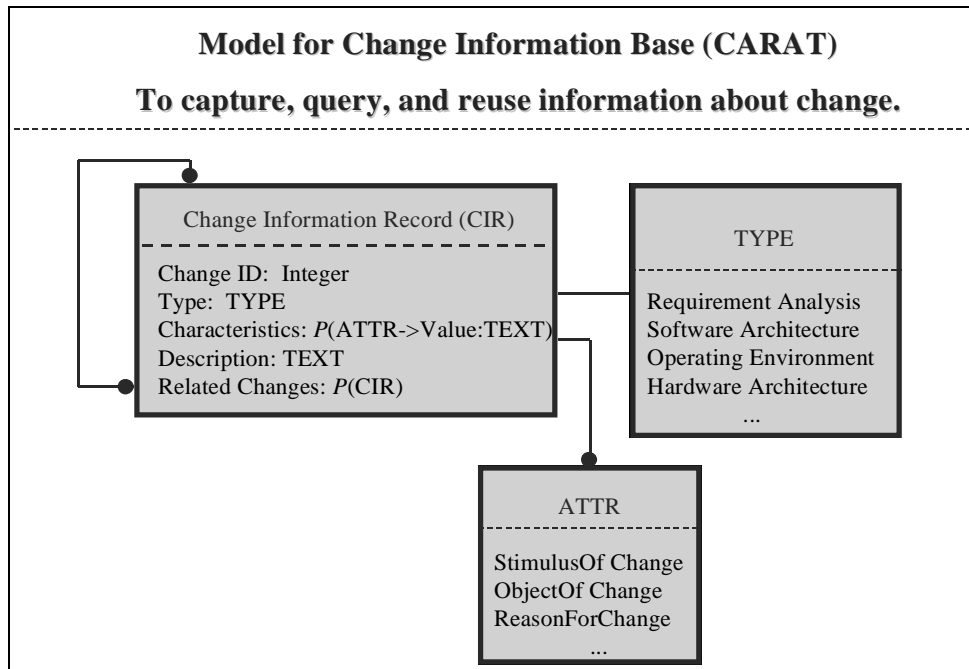


Figure 4-1: Model for Change Information Base (CARAT)

The partitioning techniques are based on decomposition for reuse and recomposition to localize change. We developed algorithms to localize change in basic solution elements such as data/operations and control flow. The decomposition process is done manually by the designer while the grouping process is automated. We applied the techniques to the decomposition of algorithms for a class of search techniques, to the organization of the Metaphor Project tools within directories, and to the generation of an architecture for CAD tools to design MEMS devices. Figures 4-2 and 4-3 shown below describe the theoretical concepts embodied in our tools.

In Figure 4-2 we use formal logic to explain the mathematical process by which we group data and operations into components to localize change. A change set CS is a set of data/operations that would be impacted by a particular change. A change could be an expected or feasible alteration of required behavior, design, or execution environment. A data or operation is included in the change set related to a change if implementing the change would require the human (not a compiler) to alter the code representing the data or operation. The union of change sets which overlap directly via intersection (Overlap relation) or indirectly via transitivity (Affinity relation) form a component (Same-Component relation). The figure shows the results of partitioning genetic algorithms for optimizing the search of a solution space into components. This diagram is based on work reported in [Hoover & Khosla, 1996].

Figure 4-3 describes the key idea for proving that partitioning a sequence of high-level task activations so that invariant subsequences are each located in a single component minimizes the complexity of changing the sequence. Such sequences represent the control flow of a master-controller style of architecture. This figure is based on work reported in [Hoover & Khosla, 1997].

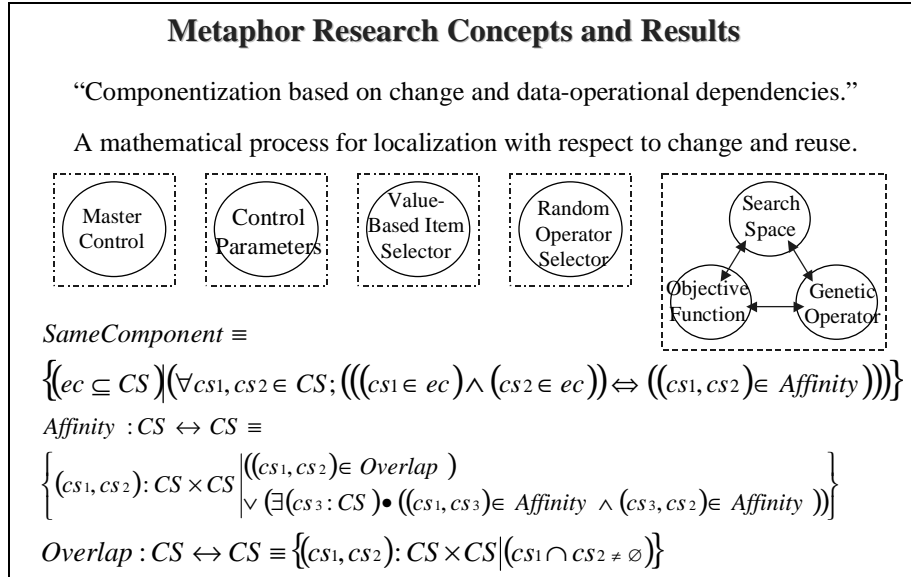


Figure 4-2: Componentization Based on Change and Data-operational Dependencies

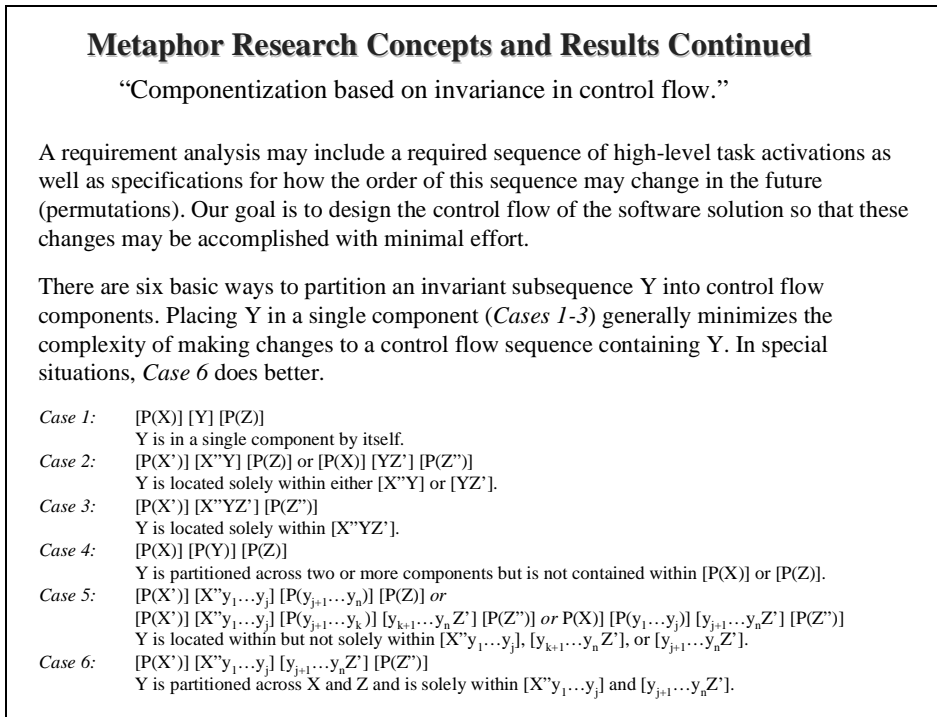


Figure 4-3: Componentization Based on Invariance in Control Flow

4.3. Description of Tools

We developed three basic tools in the Metaphor Project: CARAT (Change Analysis Rational and Technology), ACT (Analytical Composition Technology), and CIM (Change Information Manager). The prototypes of these tools execute in a web environment. All of the tools are implemented in java and support standard interfaces between a java web-server and a relational database. Hence with minimal change to the original tools, we were able to port them from a Unix to a WindowsNT execution environment.

CARAT (Change Analysis Rational and Technology)

The problem addressed by the CARAT tool is how to capture and organize information about change in a way that enables (1) analysis of change information from differing viewpoints, (2) use of change information during the design process, and (3) analysis of the likely impact of changes to requirements. The solution involved a meta-model and process for creating domain-specific change information bases as well as a user interface for interacting with the change knowledge base. The CARAT tool enables the hardware or software developer to store, search, and retrieve historical information about changes to existing systems. The designer can also record information about anticipated or feasible changes that result from the domain analysis. The back-end relational database can support SQL queries and has a standard interface to the java web-server.

ACT (Analytical Composition Technology)

The problem addressed by the ACT tool is how to analytically partition a software solution into components that are easily changed, reusable, and tunable to satisfy adaptable quality-of-service (QoS) requirements. The solution involved semi-automated and analytical transformation from requirements analysis to high-level design, analytical software component partitioning to promote localized change, reuse, and design trade-offs based on quantified analysis of alternative solutions. We did not have time to study partitioning for QoS requirements such as performance and reliability. The transformation from requirements into the basic solution elements is still a manual process. The user must enter the basic solution elements and their dependencies into the tool. As shown in Figure 4-4, ACT partitions these elements and produces a recommended set of components. The data-operational dependencies, alternative control flow sequences, and component partitions are stored in a database for future review and reuse.

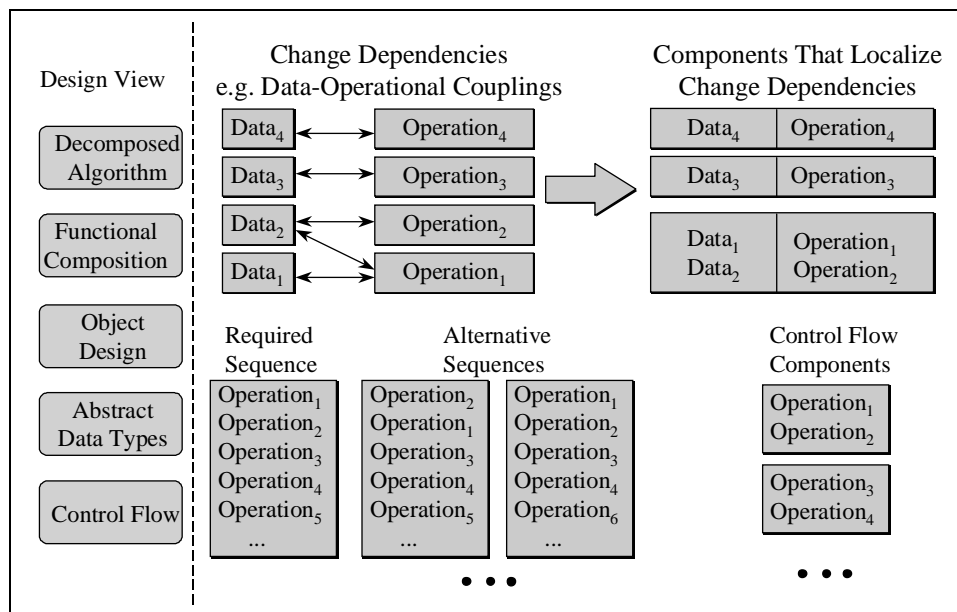


Figure 4-4: Partitioning of Basic Software Solution Elements

CIM (Change Information Manager)

The change information manager enables the software change manager to specify a set of allowable change types and attribute-keyword pairs to describe change information about a particular development project or organization. A set of specified types and attribute-keyword pairs is stored in a database of configurations. The change information manager or project leader can then select a configuration to be associated with a particular workspace. Workspaces are individual change information stores for a project, group, or organization. All of the keywords are stored in a master database that change information managers can peruse and modify. Likewise, configurations as well as their associations with workspaces can be created, modified, and deleted. The reports feature allows the manager to review configurations with particular values for the types or attribute-keyword pairs, to compare configurations, and to list workspaces associated with a particular configuration.

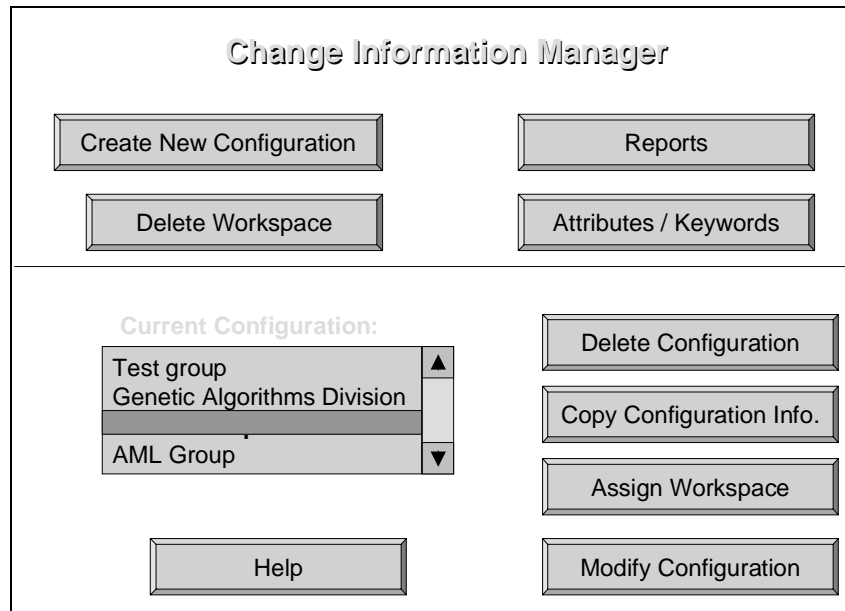


Figure 4-5: Interface to the Change Information Manager (CIM)

5. Lessons Learned

Despite the fact that the contract period was shortened, the Metaphor Project was an excellent learning experience for our team. In this section, we would particularly like to share our observations with “young” researchers. Working with DARPA is an exciting, rewarding, and challenging experience. The positive aspects are, of course, the funding to support the research but also the opportunity to interact with potential government and industrial users of the research and with other researchers from government and academe. Unfortunately, this interaction can be a double-edged sword. While interaction with other researchers is not only encouraged but sometimes mandatory, the competitive funding environment motivates selective cooperation in the interest of securing DARPA approval and not necessarily in the interest of advancing knowledge and technology.

DARPA is a challenging customer because, on the one hand, it has a mission to promote the research and development of new technologies that improve the state of the practice (the design and development of evolutionary and complex software systems in the case of the DARPA-EDCS program) and, on the other hand, it must continually demonstrate the effectiveness of these technologies. Obviously, there is a timing conflict between these two goals: research and development, at least of prototype technology, must precede demonstration of effectiveness. So how can DARPA fund meaningful research if it needs demonstration from the beginning to the end of a project contract?

Some DARPA programs fund research that is shielded by already existing technology. This technology is ready to market or has been already been marketed in a slightly different version and can be demonstrated while the new research is being done. The lesson learned is that acquiring and maintaining DARPA-funding for new research requires

a shield. The shield may be existing technology for demonstration purposes, collaboration with one or more government contractors who plan to use and in some sense validate the work, or program managers who think that the new research could potentially be useful in the future.

From a technical viewpoint, one of the most relevant questions that we received about our work concerned the value of designing for evolution, reuse, and adaptability. The question is whether or not one can know in advance how the software system is likely to be modified, reused, or used differently in the future. We think this is a valid concern but that it does not invalidate the need to plan for change and adaptability. When the analyst specifies the requirements for a software system, the person should elicit the advice of domain experts who best understand how the system would be used. These domain experts can certainly suggest alternative or ranges of use. Given this analysis, the software designer tries to generate an architecture to support variation and evolution. The current industrial approach is to design families of software systems which can be derived from a single, evolvable architecture with reusable components. Our goal is to help the designer more systematically generate an architecture that can be easily modified to support alternative uses of the software system. These uses may include properties such as performance and differing levels of need for reliability.

6. Educational Contributions

In addition to its technical contributions, the Metaphor Project helped to fund or enhance the education of several Carnegie Mellon students. David Patron was able to complete about 80% of his Masters of Software Engineering degree as a part-time student with employee benefits funded by this project. As undergraduate summer interns during 1997 and part-time student assistants from Fall 1997 to Spring 1998, Reynald Ong and Shannon Qui participated in the design and implementation of innovative research tools and worked on a research team. In the summer of 1998, both of these students secured internships with companies specializing in financial applications that depend upon database and web technologies, both of which they explored in the Metaphor Project.

A group of students (Diego Basch, Chris Canon, Jun Goo, and Pradip Hari) taking the School of Computer Science graduate course Introduction to Real-Time Software and Systems researched and developed a software component calibrator and a distributed software components library server based on CORBA connections. The target platform was WindowsNT. The Metaphor Project leader served as their mentor.

The Metaphor Project leader was also a part-time doctoral candidate in the Department of Electrical and Computer Engineering at Carnegie Mellon University. The research ideas and theoretical concepts for the Metaphor Project are part of her thesis work. She specified and directed the development of the tools which were built to demonstrate the research concepts. This DARPA-funded research project enabled her to research and demonstrate some of her ideas.

7. Summary and Future Work

This report reviewed and summarized the Metaphor Project objectives and accomplishments for the funding period 1 July 1996 to 30 June 1998. Due to the shortened contract, we did not have the opportunity to develop and test the Metaphor research ideas as extensively as needed to demonstrate their significance and potential for enhancing commercial software design tools. We completed prototype versions of the ACT, CARAT, and CIM tools that demonstrate the change information capture, change taxonomy configuration, and component partitioning technologies.

The first author is redirecting her thesis work to the development of a framework for characterizing basic elements of software solutions with respect to properties such as candidacy for evolution, reuse, and adaptability. The candidacy properties are variables that can be weighted and correlated depending on the designer's goals. The goal is the development of functions (or metrics) for determining "good" partitions of a software solution into components. The work would integrate concepts and mathematical techniques from hardware-software codesign [Adams & Thomas, 1997], information theory [Diday, 1994] and [Reinke, 1991], and software engineering [Belady, 1981]. The Metaphor tools would be integrated and expanded into a software architect's toolbox for generating a software architecture from basic solution elements such as data/operations and control flow with respect to change, reuse, and adaptability. This type of synthesis would be based on mathematical analysis of the relationships between the solution elements and would result in recommendations for how to partition these elements into software components to achieve the target design objectives.

Acknowledgments

We would like to acknowledge the contributions of David Patron, Reynald Ong, and Shannon Qui, all of whom were members of the Metaphor Project team. David designed and implemented the ACT, CARAT, and CIM databases and their web server interfaces. He contributed to the behavioral specification, design and implementation of the CIM tool. Reynald developed most of the client-side interfaces for accessing the ACT, CARAT, and CIM tools via the web. He contributed to the design of the client-server communication protocols for all three tools. Shannon helped to design and implement the ACT tool. She investigated the feasibility of migrating from custom-built applets to applets generated by a web-based GUI builder and contributed to the design of the client-server communication protocol for the CIM tool. We thank David, Reynald, and Shannon for their individual technical contributions and for their ability and willingness to work together as a team.

Last and most significantly, we would like to sincerely thank DARPA for the financial support that enabled us to achieve the accomplishments discussed in this report.

Appendix A: Accomplishments With Respect to the Statement of Work

Task 1 Change Analysis Rationale and Technology (CARAT)

Subtask 1.1 (Qtr. Report Subtask 2.1.1: Capturing & Analyzing Changes to Real-Time Applications)

Deliverable 1.1.1 Application of the rationale to the analysis of requirements and design of a speech system for various military uses.

Deliverable 1.1.2 Technical report.

We investigated changes that are commonly made to real-time software systems. We developed an information model for organizing knowledge about change and a tool (CARAT) for capturing this information. The user interface to the tool is a web-based environment. Due to personnel changes, we applied our model to an analysis of changes applicable to generic hardware and software components and systems, to a class of algorithms, to the organization of the Metaphor software system, and to the architect of an evolvable architecture for CAD tools used to design microelectromechanical devices (MEMS). A description of the change information model appears earlier in this report. A discussion of the MEMS application can be found in the attachments [Hoover & Khosla, 1998].

Subtask 1.2 (Qtr. Report Subtask 2.1.2: Taxonomies of Historical Changes)

Deliverable 1.2.1 Instantiation of a meta-process for creating taxonomies of likely changes.

Deliverable 1.2.2 Technical report.

We designed an information model for creating taxonomies of historical changes. The user can instantiate the model for a particular application domain or scope (for example by project, organization, or product-line). We designed and implemented a tool (CIM) that enables a change manager to configure the CARAT interface (via selectable keywords) for the intended application. The user interacts with the tool from a web browser.

Subtask 1.3 (Qtr. Report Subtask 2.1.3: Planning for Change During Software Requirements Analysis and Design)

Deliverable 1.3.1 Demonstration of incorporating knowledge about change into the specification of application requirements and into the design of reusable software.

Deliverable 1.3.2 Technical report.

We investigated several commercial information capture tools to determine the extent to which they help the software engineer design for change. Our work goes beyond the current state of the art by guiding the engineer on how to think about and specify the expected types of changes to be made to the system. The CARAT tool demonstrates the information capture mechanism that we developed. The ACT tool demonstrates the inclusion of information about change into the partition of software components that localize change. Currently the user can analyze information retrieved from the CARAT database and manually submit the relevant change information to the ACT partitioning tool.

We had planned to more thoroughly study techniques for specifying requirements and to investigate a way to map requirements to software components that localize change. Our synthesis process would have included a feedback loop for integrating historical information about change into the design of new components. We had planned to conduct this investigation during the final two years of the project. In this report, we briefly outlined some current trends in commercial tools.

Subtask 1.4 (Qtr. Report Subtask 2.1.4: Database Implementation of a Taxonomy of Historical Changes)

Deliverable 1.4.1 Instantiation of a meta-level design for creating databases that are software realizations of the related taxonomies of change.

Deliverable 1.4.2 Technical report.

Deliverable 1.4.3 User's manual.

We designed and implemented relational databases for the change information records (CARAT) and for the change information management (CIM). We instantiated the CARAT and CIM databases to test the CARAT and CIM tools. A description of the CARAT and CIM tools appears previously in this report. We had planned to complete the user's manual during the final two years of the project.

Task 2 Analytical Composition Technique (ACT)

Subtask 2.1 (Qtr. Report Subtask 2.2.1: Formulation of ACT Algorithm)

Deliverable 2.1.1 Application of an algorithmic method for designing reusable software solutions that minimize the impact of change.

Deliverable 2.1.2 Technical report.

We investigated and developed algorithms for localizing change with respect to data/operations and control flow. We developed first-order metrics for measuring the ability of the components synthesized by our algorithms to simplify anticipated changes. Because of the shortened funding period, we were not able to complete the investigation of partitioning components to promote adaptable QoS. We applied our techniques to the partition of algorithms, high-level architectural components for a real-time application, the directory structure for the tools that we developed, and the architecture of a CAD tool for designing MEMS devices. Due to the reduced time span of our project, we were not able to apply our partitioning techniques to a real-world system as planned.

Subtask 2.2 (Qtr. Report Subtask 2.2.2: Automation of ACT Algorithm)

Deliverable 2.2 Tool which mechanizes the composition algorithm.

We demonstrated a prototype of a software tool that automates the ACT partitioning algorithms. The ACT tool performs the automatable parts of our algorithms. We integrated the ACT tool into a web-accessible environment.

Subtask 2.3 (Qtr. Report Subtask 2.2.3: Evaluation of ACT Algorithm)

Deliverable 2.3.1 Experiment which applies the measurable process for determining the human effort required to build architecturally similar software solutions.

Deliverable 2.3.2 Technical report.

Deliverable 2.3.3 User's manual.

The development and application of an experimental method to measure the effectiveness of our technologies as well as the completion of the technical report and user's manual was to be done during the last two years of the contract. We developed a first-order metric for measuring the difficulty of changing control flow components.

Task 3 Virtual Software Design Environment (VSDE)

Subtask 3.1 (Qtr. Report Subtask 2.3.1: Interoperability Between Heterogeneous Systems)

Deliverable 3.1 Demonstration of interoperability between heterogeneous real-time and general purpose operating environments.

We deployed the CARAT, ACT, and CIM tools in a web-based environment. Our java-implemented tools are portable and interoperable with respect to the standards set by the original developers of java. We tested them successfully on Windows NT and Unix platforms.

Subtask 3.2 (Qtr. Report Subtask 2.3.2: Meta-language for Defining Speech Interface)

Deliverable 3.2.1 Multi-modal (e.g. speech) interface to design, compose, and test real-time applications.

Deliverable 3.2.2 Technical report.

Due to personnel changes, we redirected efforts from deliverables 3.2.1 and 3.2.2 to the web deployment of the CARAT and ACT tools.

Subtask 3.3 (Qtr. Report 2.3.3: Integration of CARAT and ACT into VSDE)

Deliverable 3.3 Integration of soft real-time computing (e.g. multimedia) with hard real-time computing (e.g. aircraft control).

We completed the integration of the CARAT, ACT, and CIM technologies into a web-based software design environment.

Subtask 3.4 (Qtr. Report 2.3.4: QoS Monitoring and Management Tool)

Deliverable 3.4.1 End-to-end analysis of real-time performance.

Deliverable 3.4.2 Technical report.

Deliverable 3.4.3 User's manual.

We developed a prototype component profiler and manager. The profiler determines and archives the CPU and memory usage for CORBA-enabled components running on a Window's NT platform. We tested the effectiveness of pre-profiling applications to forecast their resource needs. We did not have time to complete the technical report and user's manual for this segment of our investigations.

References

- Adams, J. and D. Thomas (1997), "Design Automation for Mixed Hardware-Software Systems", In *Electronic Design*, Vol. 45, No. 5, pp. 64-66 & 71-72n
- Bass, L., P. Clements, and R. Kazman, "Chapter 9: Analyzing Development Qualities at the Architectural Level: The Software Architecture Analysis Method," In *Software Architecture in Practice*, Addison Wesley Longman, Reading, MA, 1998, pp. 189-220.
- Belady, L. (1981), "Complexity of Large Systems," Chapter 13 in *Software Metrics: An Analysis and Evaluation*, A. Perlis, F. Sayward, and M. Shaw, Eds., MIT Press, Cambridge, MA, pp. 225-233.
- Booch, G. (1994), *Object-Oriented Analysis and Design: With Applications*, Second Edition, The Benjamin/Cummings Publishing Company, Redwood City, CA.
- Britton, K. and D. Parnas (1981), *A-7E Software Module Guide*, NRL Memorandum Report 4702, December.
- Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal (1996), *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Chichester, England.
- Diday, E. (1994), *New Approaches in Classification and Data Analysis*, Springer-Verlag, Berlin, Germany.
- Fayad, M. and M. Cline (1996), "Aspects of Software Adaptability," *Communications of the ACM*, Vol. 39, No. 10, Oct., pp. 58-59.
- Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Company, Reading, MA.

- Hoover, C. and P. Khosla (1996), "An Analytical Approach to Change for the Design of Reusable Real-Time Software," In *Proceedings of the Second Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'96)*, Feb. 1-2, IEEE Computer Society Press, Los Alamitos, CA, pp.144-151.
- Hoover, C. and P. Khosla (1997), "Analytical Design of Evolutionary Control Flow Components," In *Proceedings of the Second Workshop on High Assurance Systems Engineering (HASE'97)*, Aug. 11-12, IEEE Computer Society Press, Los Alamitos, CA, pp. 48-55.
- Hoover, C. and P. Khosla (1998), "Analytical Design of Evolutionary and Reliable Software Components for MEMS Design Tools," In *Proceedings of the Third IEEE High Assurance Systems Engineering Symposium (HASE'98)*, Nov. 13-14, IEEE Computer Society Press, Los Alamitos, CA, pp. 188-199.
- McIlroy, D. (1969), "Mass Produced Software Components," *Software Engineering Concepts and Techniques*, 1968 NATO Conference on Software Engineering, Eds. J. Buxton, P. Naur, and B. Randell, Petrocelli/Charter, NY, pp. 88-98.
- Parnas, D. (1972), "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, Vol. 15, No. 12, pp. 1053-1058.
- Parnas, D., P. Clements, and D. Weiss (1984), "The Modular Structure of Complex Systems," In *Proceedings of the Seventh International Conference on Software Engineering*, Mar., pp. 408-417, Reprinted in *IEEE Transactions on Software Engineering*, SE-11, 1985, pp. 259-266.
- Reinke, R. (1991), *Symbolic Clustering*, Ph.D. Dissertation, Report No. UIUCDCS-R-91-1704, Department of Computer Science, University of Illinois, Urbana-Champaign, IL.
- Selic, B., G. Gullekson, and P. Ward (1994), *Real-Time Object-Oriented Modeling*, John Wiley & Sons, New York, NY.
- Shaw, M. and D. Garlan (1996), *Software Architecture: Perspectives on an Emerging Discipline*, Chapter 2: "Architectural Styles," Prentice Hall, Upper Saddle River, NJ, pp. 19-32.
- Stewart, D. (1994), *Real-Time Software Design and Analysis of Reconfigurable Multi-Sensor Based Systems*, Ph.D. Dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA.
- Ward, P. and S. Mellor (1985), *Structured Development for Real-Time Systems*, Vol. 2: Essential Modeling Techniques, Prentice Hall, Englewood Cliffs, NJ.
- Westerberg, K. (1989), *Development of Software for Solving Systems of Linear Equations*, Technical Report: EDRC-05-35-89, Engineering and Design Research Center, Carnegie Mellon University.
- Weide, B., W. Ogden, and M. Sitaraman (1994), "Recasting Algorithms to Encourage Reuse," *IEEE Software*, Sept., pp. 80-88.