Proposal: BAA # 00-20
Dynamic Assembly for Systems Adaptability, Dependability, and Assurance
DARPA Program Manager: John Salasin

DYNAMO
DYNamic Assembly from MOdels

Volume I - Technical Proposal

Prime offeror: Georgia Institute of Technology
Subcontractor: Michigan State University

Technical Point of Contact:
Dr. Spencer Rugaber
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

phone: (404)-894-8450
fax: (404)-894-9442
e-mail: spencer@cc.gatech.edu

Administrative Point of Contact:
David R. Hendrix
Office of Contract Administration
Georgia Institute of Technology
400 Tenth Street N.W.
Atlanta, Georgia 30332-0240

phone: (404)-894-6939
fax: (404) 894-6956

Type of business: Other educational
Duration of effort: 24 months

1 Executive Summary

Goal Automated composition of software systems in such a way as to guarantee various properties such as correctness, reliability, and resource utilization. The software system components may be Commercial Off-the-Shelf (COTS) or they may be custom-built for military applications. The composition may take place statically, when the system is first configured, or dynamically, as new components are added or old components are replaced.

Approach The DYNAMO (Dynamic Assembly from Models) project addresses this challenge by making extensive use of declarative models of the software components. Models may be built from scratch for new components or they may be derived by analyzing existing components. Models are abstract and therefore easier to maintain than software built by hand. Models enable automatic generation of software with guaranteed properties. Additionally, the same model may be used to construct the system and to gauge its performance when it runs.

Background The two Principle Investigators have extensive experience using formal models to solve practical software development problems. Specifically, they participated in two related DARPA-funded efforts. The MASTERMIND project (from the HCI/HSI program) demonstrated how to automatically generate provably correct user interfaces from models describing the interfaces' presentations, dialogs, and underlying software functional capabilities. The MORALE project (from the EDCS program) made use of architectural models to guide the evolution of existing systems. In addition, it also developed tools to perform both static and dynamic program analysis to determine properties of those systems.

Deliverables DYNAMO will deliver the following technology for the DASADA program:

- Modeling notations for specifying system components
- Automatic code generators for building components from models
- A system composer supporting the static and dynamic composition of components
- An evaluation framework in the form of *critics* (gauges) that measure system properties
- A program analysis tool (EVA) for determining properties of existing software systems.

Evaluation We intend to apply our technology directly to military requirements. We will provide a specific demonstration of the application of DYNAMO to a realistic dynamic assembly problem, as would occur in an operations planning (war game) exercise. Operations planning is a well-defined process for which documentation and training materials exist. We intend to use DYNAMO to build an operations planning environment that includes heterogeneous information sources, multiscale visualizations, and severe robustness requirements.

2 Innovative Claims

Models are declarative rather than being written in a procedural programming language such as Ada or C++. This leads to the advantages of abstractness/conciseness and maintainability/flexibility. A potential disadvantage, however, is inefficiency.

Claim 1: DYNAMO will provide a solution to the inefficiency problem. We have already developed code-generator technology and a run-time environment that overcomes the inefficiencies evident in other approaches to model-based system development [52, 53].

Given that we have overcome the major disadvantage of using models, we can leverage the advantages of model-based software development to address the DASADA challenge.

Claim 2: High-level (abstract) models enable static and dynamic determination of system properties. Static determination for existing software components is obtained from program analysis tools [43]. Dynamic determination is obtained from both analysis of system events [25] and from run-time, model-aware critics (gauges) [35].

The other major advantage of models is their flexibility. In particular, different components of a complex software environment may be represented by models described using different notations. This may lead to what is called in the literature either the *multi-paradigm programming problem* [62] or the *model composition problem* [53]. For example, imagine the difficulty of integrating a real-time executive monitoring live sensors with an expert system.

Claim 3: DYNAMO will support the automatic static or dynamic composition of components that differ greatly in their programming paradigms or their execution engines.

Our previous research in the use of models has focussed on models of user interfaces [35, 56, 9, 49, 52, 50, 53]. Consequently, we intend to demonstrate and evaluate DYNAMO on a problem in which significant user interface issues arise: How to configure and sustain a heterogeneous suite of information gathering, manipulation, and visualization tools (such as might be used during a battle-planning exercise) where problems of scale lead to resource utilization difficulties. For example, how can we assure that certain data items are current and accurately portrayed across a wide-variety of devices, such as personal digital assistants (PDAs) and wall boards.

Impact on the Research State of the Art

- From the point of view of software research, the proposed work fits into the area of model-based software development and directly addresses the *model composition problem* [53] (how to efficiently compose systems using models to guarantee properties such as correctness).
- It also makes contributions to the problem of how to construct multi-paradigm software systems. Paradigms illustrated in the operations-planning example include real-time systems, information systems, distributed systems, interactive systems, and computationally intensive systems.
- Other areas of contribution include event-based program analysis, automatic code generation from models, and scalable and collaborative user interfaces.

Impact on Military Systems

- Increased assurance due to design time configuration checking and run-time monitoring
- Increased flexibility due to integration of heterogeneous systems
- Increased robustness particularly related to scalable resource utilization

3 Summary of Deliverables and Technical Approach

In DYNAMO, we will develop, demonstrate, evaluate, and transition new technologies for the dynamic assembly and adaptation of software systems. We will validate the technology on a complex assembly of components, such as might be used in a military battle-planning exercise. Key ingredients of this assembly include sub-systems with a wide variety of highly-graphical user interfaces. The offerors are willing to work with other contractors and to supply our technology results to others.

The high-level goal of DYNAMO is to measure (gauge) and guarantee properties of systems built from components that will be assembled dynamically at run-time. The fundamental problem that our work addresses is the practical intractability of static analysis methods in reasoning about composition properties of dynamic assemblies of components. In such systems, *a priori* analysis of many interesting properties is infeasible. These properties can only be guaranteed dynamically by monitoring the state and behavior of software components in composition.

Technical approach DYNAMO will transform *component models* into a representation that can be consulted and analyzed at run time. The run-time representation and coordinated execution of models will allow dynamic analysis of component composition. Moreover, by using models during component design, it will be possible to perform static analysis of some composition properties, and to defer those properties that cannot be checked statically to run-time validation. Figure 1 illustrates the major functional components of DYNAMO. A typical use begins with a collection of models, each of which describes a different behavioral aspect of a software component. For each model, a specialized Code Generator creates a module that implements the behavior specified in the model. These modules are then *composed* into a code component. To support dynamic assembly and adaptability, the Composer augments the generated component with code to monitor and adapt the status of the component at run time (Dynamic Component). Using these mechanisms, the behavior of a component can be monitored and affected by a Critic, which runs concurrently and interacts with the component.

Enabling technologies DYNAMO will employ the following enabling technologies:

- Static analysis of components and their models to (when feasible) formally verify composition properties.
- Dynamic event analysis to monitor and analyze an assembly for composition properties that cannot be verified statically.
- The MASTERMIND model-composition toolkit and run-time infrastructure [53] to generate a component from a model of its presentation and a model of the sub-system that it controls.

Static and dynamic analysis can be used in combination to make difficult problems tractable and to improve the precision of analysis. It is possible, for example, to apply static analysis after an initial dynamic analysis (using EVA) is applied. The dynamic analysis narrows the state space over which the static analysis (which is exhaustive and therefore expensive) must apply.

Deliverables The deliverables of this effort include the following:

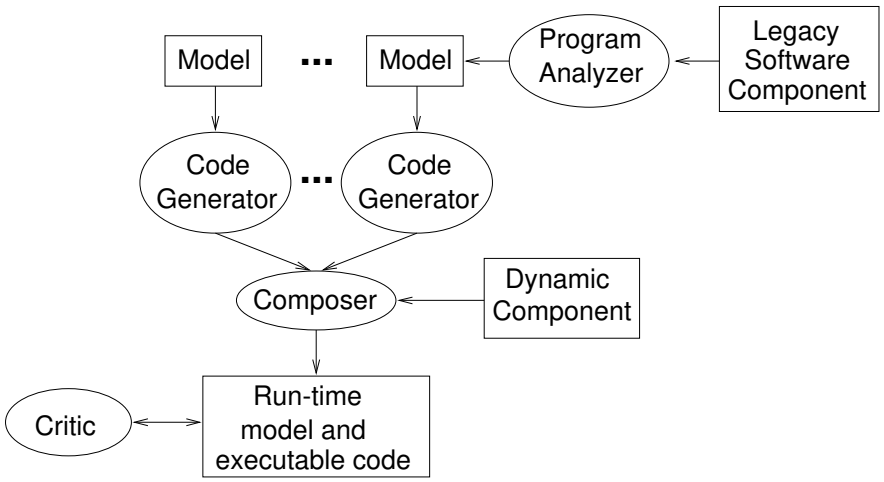


Figure 1: Conceptual architecture of DYNAMO

- Robust presentation modeling language and tool. The existing MASTERMIND presentation modeling language [11] is a powerful notation for expressing graphical designs and for making these presentations interactive. We will update and extend this language for DASADA.

- Code generators and run-time environment for components with dynamic, model-based interfaces. To support dynamic assembly and adaptation, components must provide modular interfaces that expose their status in a form that can be used to initialize other components that might replace them dynamically. These component interfaces are difficult to design, but they can be expressed intuitively using declarative models [51]. To convert these models into component interfaces requires tools and run-time support.

- Run-time critics (gauges) and a model manager. To support the use of gauges to monitor the behavior of these systems, we need run-time model managers and critics, which will monitor the status and behavior of the dynamic components and present this information to users in an intuitive way.

- The Event Visualizer and Abstractor (EVA) tool. To incorporate an existing software component into a DYNAMO environment, the properties of this component must be determined. EVA provides both static (compile time) and dynamic (run-time) analyses. Both types of analyses provide a structural (architectural) view of the components of a software system and how they interact. The units of interaction are events, and EVA uses visualization and abstraction to deal with the wealth of data produced when analyzing large systems capable of generating millions of events.

We plan to demonstrate these deliverables in the combined static/dynamic analysis of a composition property that is critical to the correct operation of mobile military planning and control systems. The property is correct composition with respect to resource utilization in the presence of *multi-scale* user interfaces. A monitoring and control system with a multi-scale user interface is assembled from interactive components that are parameterized by a scale of presentation resolution and control precision. At run-time, the resolution of an entity's presentation and the precision of its remote control will vary by region of the display being used to manage a battle-planning application. Ideally, these applications would always run with presentations at maximum resolution and with maximum precision of remote control. In reality,

however, both resolution and precision will be limited by the availabilty and bandwidth of one or more physical resources, such as real-time data from satellites. At run-time, based on resource availability and demand, one or more components in this assembly may be replaced by another set of components within the same family. We intend to validate our approach on these applications.

Gauges The dynamic guarantee of properties will be accomplished by model aware, run-time critics (gauges). A critic is a run-time object that monitors the status and behavior of a separate run-time component. A critic maintains a run-time model of the behavior and composition needs of the component that is being monitored. This model is updated by the critic at run time by actively observing some subset of the monitored component's internal status and inter-component events. Critics will be specified using a modeling notation that is appropriate for the type of property that we want to gauge.

A distinguishing feature of DYNAMO is its support for multi-paradigm modeling. This support enables the generation of critics from one of many different modeling notations. A correctness property, such as that two components never simultaneously access a shared resource, can be declaratively specified using a notation with features for specifying the temporal behavior of systems. Using model-based code generation, we can generate a critic that checks for adherence to this property dynamically. Another interesting property is resource utilization, which can be specified declaratively as a visualization of the internal attributes of one or more components in an assembly. Such a notation will likely appeal to a different paradigm than that of the temporal notation. Multi-paradigm modeling and generation is therefore necessary to support the convenient specification of different kinds of gauges.

Differentiators

- Code generation and efficiency

  Generating stand-alone code from a specialized modeling notation is easy. Integrating code generated from multiple models is difficult. Integration is much more complicated than mere linking of compiled object modules. In DASADA systems, the efficient composition of components specified using multiple models is essential to managing complexity. Unlike middle-ware solutions, which implement composition architecturally (using event registration and notification) DYNAMO will be able to synthesize a unified component from multiple models. Consequently, our code will be more efficient.

- Run-time monitoring of components

  The information needs required to implement critics conflict with the desire to encapsulate the implementation of a component behind a rigid, modular interface. To make a component "monitorable" without compromising the benefits of separation of concerns is difficult, and this design problem is currently poorly supported by tools and methodologies. A poor extension leaves an undesirable footprint on the interfaces of application components. However, by generating these components (and their interfaces) from declarative models, programmers never have to deal with this problem [51].

- More thorough program analysis

  Because models will be used throughout design, DYNAMO will be able to statically analyze some composition properties and to defer those properties that cannot be checked statically to run-time through model-generated critics. Moreover, with the ability to visualize patterns of events using

EVA, designers and technicians will be able to detect composition problems that they may not have been able to specify (and construct a gauge for).

- Heterogeneity:

  Because we generate component code from declarative models, these models can be abstract with respect to platform dependencies. By providing alternative code generators, we can generate code for multiple platforms from the same model(s). This will enable the uniform management of heterogeneous dynamic systems.

Evaluation We will apply our technology directly to military requirements. We will provide a specific demonstration of the application of DYNAMO to a realistic dynamic assembly problem, as would occur in a battle planning (war game) exercise. Battle planning is a well-defined process for which documentation and training materials exist. We intend to use DYNAMO to build a battle-planning environment that includes heterogeneous information sources, multiscale visualizations, and severe robustness requirements.

A scenario for this exercise is:

Situation: Battle/operations planning, mission analysis, course-of-action determination

Participants: executive officer, S2, S3, engineering officer, artillery officer, etc.

Information needs: maps, terrain data, weather data, combat-strength charts, vehicle availability, enemy position and activity, etc.

Activities: status reporting, brainstorming, contingency planning, scheduling

Current methods: physical maps with transparent overlays, magnetic icons, posted charts, markers, rulers for judging artillery ranges

Future methods: lightweight, foldable, wide screen, electronic information displays, live, real-time data feeds, cooperation with personal digital assistants, expert system planning tools, etc.

The challenge in this evaluation is how to maintain the flexibility and robustness of physical drawings, charts, and markers, while taking advantage of the timeliness and power of electronic media. In particular, imagine a comprehensive suite of battle management tools that support real-time update (e.g. satellite weather or surveillance data), access to stored information (e.g. geographic information systems, personnel data), computationally intensive application programs (e.g. expert systems, weather prediction), personal digital devices (e.g. Palm Pilots, body monitoring devices) and powerful visualization tools (e.g. dynamic overlays, multiple views, user-controllable zooming).

The correctness and robustness requirements that we will guarantee concern the composition of DASADA components in the presence of changes in the *scale* of interaction. While there are many approaches to dealing with the variance of resolution of presentation output (e.g., PAD++ [6, 5]), our approach handles scale for both output and input, which facilitates the design and assembly of multi-scale, interactive components. To develop these applications, we must overcome the inherent difficulty of managing dynamic composition at many different levels of granularity simultaneously.

Interactive components at different scales will have different communication interfaces and may therefore not be dynamically replaceable in certain contexts of use. Using models and formal verification, we will be able to quantify the contexts of allowable use in a form that can be consulted at run-time to decide when it is appropriate and legal to dynamically rescale a component.

To design, manage, and maintain dynamically replaceable components, we will use modeling notations and model-based generation tools. In order to help the users of these systems cope with the non-uniformity of presentation scale, we will construct critics (gauges) that present a global picture of resource utilization and presentation fidelity. We will also construct gauges that act as *warning indicators* to communicate the violation of an important composition property that could not be statically verified. An example of such a property is violation of mutual exclusion in the presence of dynamically adapting components.

When evaluating DYNAMO, we will develop scalable classes of interactors where each interactor in a given class implements the same logical entity at a different granularity of remote control and presentation resolution. Because resource requirements correlate with precision and resolution, we can order these classes so that global resource availabilty can be used to dynamically adapt the user interface of a military application to maximally exploit (but not exceed) the available resources.

DYNAMO features used in the evaluation

- Software components specified with formal models. The models are available to the system both at design/configuration time and at run-time. Models can describe either COTS software or custom systems.
- System configuration parameters can be accepted both at design time and altered dynamically. System configuration includes not only component information, but also descriptions of available display and interaction devices, remote sensor availability, application software, and information needs.
- Where necessary, software components automatically generated from models. This includes code for displaying results on a variety of devices and at a variety of resolutions, for supporting the seamless interoperation of the tools, and for measuring various desirable properties of the software, such as availability, resource requirements, and performance.
- The software will be automatically composed in such a way that desirable properties for the system as a whole can be measured and assured.
- At run-time, determination can be automatically made (using event-based program analysis) as to whether the addition or removal of a component can be done in such a way that desirable system properties can be guaranteed.

4 Proprietary Claims

None

5 Statement of Work

5.1 Objective and Scope

1. Declarative (model-based) system description

   A *model* is a declarative specification of some single coherent aspect of a user interface, such as its appearance or how it interfaces to and interacts with the underlying application. By focusing attention on a single aspect of a user interface, a model can be expressed in a highly-specialized notation. This property makes systems developed using the model-based approach easier to build and maintain than systems produced using other approaches [59]. Developers using this approach build systems by specifying models that describe the desired system, rather than writing a program that exhibits the behavior [56].

2. Automatic code generation from models

   A model, by design, represents a single coherent aspect of a system and is neutral with respect to others [3]. Model-based code generators construct a run-time module from a design-time model. The code generation strategy is model-specific, reflecting the specialization of models to a particular aspect of the system. This feature of the model-based approach allows code generators to exploit deep domain knowledge during generation and in turn enables the modeling notation(s) to be highly declarative.

3. Model composition problem

   Once specified, automated tools must integrate separate models and generate an efficient system from them. The *model-composition problem* is the need to efficiently implement and automatically integrate interactive software specified in separate, declarative models. Our research into this problem suggests that the solution requires a unifying communication mechanism to be mutually understood by all of the model-based code generators [52, 53]. Moreover, the power of model-based code generation—the degree of domain specificity and declarativeness—is a function of the expressive power of this underlying communication mechanism. The more powerful the underlying (shared) communication mechanism, the more a designer can defer the complexity of composition to automated tools and run-time libraries. We use a mechanism based on a formal notation that provides concurrency, multi-way synchronization, and a constraint-oriented style of programming.

4. Multi-paradigm programming

   The model-composition problem can be restated as the need to unify behavior in multiple models without violating the rules of intra-model composition and while generating efficient code. The model-composition problem is a declarative instance of the problem of constructing a software system where the major components are expressed with programming languages from different families or paradigms. Zave has called this the *multi-paradigm programming problem* [62]. The different modeling notations may assume different programming paradigms, and research shows that multi-paradigm composition is difficult [62].

5. Event-based program analysis

   DYNAMO is concerned with establishing and maintaining measurable system properties. For new components, automatic generation of code from models can provide certain guarantees. For existing

components, however, it will be necessary to analyze the software to understand its capabilities. As part of the MORALE project, we developed a tool called ISVis which was capable of analyzing software both statically and dynamically [25]. Dynamic analysis was performed by collecting system events, by providing sophisticated graphical visualizations, by detecting repetitive patterns, and by abstracting high-level interactions from low-level events. For the DYNAMO project, ISVis will serve as the basis for a new tool called EVA (Event Visualizer and Abstractor). EVA differs from ISVis primarily in its ability to infer properties from event data. EVA will use more sophisticated grammatical inference algorithms [32] allowing it to determine more sophisticated properties. EVA will also include a variety of new visualizations and will be made more portable and robust.

6. Property evaluation and verification

   Often the behavior of a single model is easy to understand in isolation but is difficult to understand in composition with other models. A model contains expectations about the behavior of models with which it must cooperate, and these expectations are rarely defined explicitly. Often *composition properties*—declarative specifications that assert the consistency of expectations among models—can be written very concisely in a notation such as temporal logic or finite state automata. We applied composition-property analysis to detect flaws in a toolkit of reusable widgets. This toolkit was developed under the MASTERMIND project [35]. Components in this toolkit allow a programmer to declaratively create user interfaces with complex temporal behavior. The interaction of these components is complex, and certain properties—such as receptiveness and liveness—must be verified when a particular configuration is assembled. Finite state verification tools, such as temporal-logic model checkers [14, 10, 30, 15, 22], can be used in this capacity. Properties that can be verified might be specific to a particular component assembly or they might be generic properties of model composition, which must hold regardless of the system being constructed. Our use of model checking focused on this latter form of verification.

5.2 Background

The work proposed on this project is a natural extension of efforts undertaken on two previous DARPA-funded projects: MASTERMIND (under the HCI/HSI program) and MORALE (Mission Oriented Architectural Legacy Evolution) of the EDCS program.

5.2.1 MASTERMIND

Project Background The MASTERMIND research project is concerned with the automatic generation of user-interface software from declarative models. The project is a collaboration of researchers in the Graphics, Visualization, and Usability Center at the Georgia Institute of Technology and in the Information Science Institute (ISI) at the University of Southern California. The project was funded by the DARPA Human Systems Interface program, SUN Microsystems, and Digital Equipment Corporation.

Graphical user interfaces (GUIs) are difficult to construct and, consequently, suffer from high costs. The research question that MASTERMIND sought to address is the extent to which automatic generation of GUIs from declarative descriptions can reduce costs and enforce design principles. Furthermore, if the declarative descriptions are broken into separate components, called models, modularity of the specification is improved, and additional, beneficial features such as design critics, context-sensitive help, and

dynamic reconfiguration are enabled.

MASTERMIND supports user-interface designers, who specify systems using separate models. A designer's job is to design a user interface for an underlying software system. The designer may or may not participate in the design of the software system itself. There may be more than one designer, and the designers may be broken into teams. Furthermore, the teams may be divided along functional lines, with, for example, one team responsible for screen layout, one for dialogue, and one for application software. Multiple designers require modularization and integration. Given a declarative specification mechanism, this implies strong support for consistency checking among the parts of a design.

Project Capabilities Several MASTERMIND capabilities are directly applicable to DYNAMO:

- Multi-paradigm modeling

  Ability to describe a variety of system components with declarative models. In particular, components with different underlying paradigms could be modeled. MASTERMIND was concerned with the design of graphical user interfaces (GUIs) described by models that use three different paradigms:

Component	Paradigm
Presentation	Demand-driven widget toolkit (Amulet)
Dialog	Reactive (state machine-based)
Application	Object-oriented

- Code generation from models

  MASTERMIND models are expressed in high-level notations. Models have the advantage of being easier to analyze and maintain than code. However, to be useful to practitioners, an optimizing compiler must be available to generate efficient code from the models. MASTERMIND included compilers that converted the models from the high-level notation into C++ code.

- A solution to the *model-composition problem*

  Models, particularly those expressing components belonging to different paradigms pose a particular challenge: How to integrate the models in such a way that properties held by the individual models are maintained in the composite system. Even more challenging is how to accomplish this while generating as efficient a system as would have been obtained if the system were built directly in code as opposed to being generated from models. We call this the *model-composition problem*, and MASTERMIND showed that it can be solved for a powerful class of GUI systems.

- Automatic verification of system properties

  One advantage of models is that their high-level, declarative descriptions support automatic verification of system properties. For MASTERMIND, we used a model-checking tool (SMV [30, 31]) to verify the correctness of our transformation from dialog models into code. In principle, model checking can be applied to verify a wide variety of system properties if the models are available and are maintained so that they are always correct with respect to the code. Because model-based compilers generate code from models, this issue of model drift is not a problem in MASTERMIND or DYNAMO.

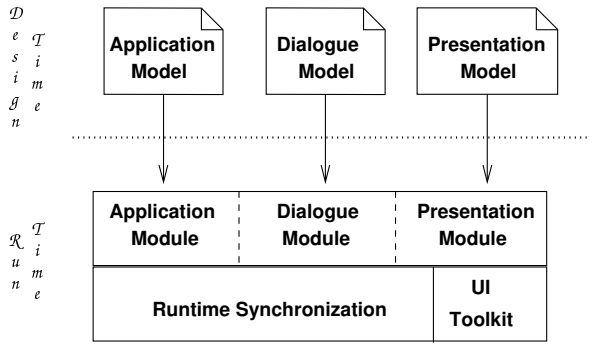


Figure 2: Model-based code generation

- Run-time management and consultation of models

  The MASTERMIND environment is a suite of user-interface generation tools that operate on models from a common (distributed) model repository. The model repository holds all forms of models from early design (user-task) models to presentation and dialogue models. The repository is managed by a ubiquitous model server, which supports dynamic object requests. Tools, like model editors and code generators, connect to this server and can then access and update models. When models change, the server notifies all dependent models and registered tools of the change. The environment was designed around an object request broker architecture (CORBA [36, 57]), and Stirewalt was directly involved with its implementation. The experience acquired while working on this project, will be directly applicable to DYNAMO, which will also use a run-time model repository and which will need to dynamically connect a distributed set of tools.

Project Accomplishments

- Conceptual Architecture for Automatic Generation of User Interfaces from Declarative Models

  The MASTERMIND tool suite generates user interfaces from three kinds of models: *Presentation models* represent the appearance of user interfaces in terms of their widgets and how the widgets behave; *Application models* represent which parts (functions and data) of applications are accessible from the user interface; and *Dialogue models* represent end-user interactions, how they are ordered, and how they affect the presentation and the application. The dialogue model acts as the glue between the presentation and application models by expressing constraints on the sequencing of behavior in these models. Model-specific compilers generate modules of code from each model, and these resulting modules are composed (Figure 2). A distinguishing characteristic of MASTERMIND is that the model-specific code generators work independently of one another. Each compiler constructs a run-time module from a design-time model. Each code generator outputs C++ source files that are then compiled and linked with a number of run-time libraries. The code generation strategy is model-specific, reflecting the specialization of models to a particular aspect of the system.

- Solution to the *model-composition problem*

  Composing the code generated from multiple models is difficult. A model, by design, represents some aspects of a system and is neutral with respect to others. Inevitably, however, functionality described in one model overlaps with or is dependent upon functionality described in another. A

button, for example, is specified in the presentation model, but the behavior of the button influences behavior in other models, such as when pressing the button causes other widgets to be enabled or disabled. The effect of pressing a button might also cause some application method to be invoked. When code generated from multiple models must cooperate, these redundancies and dependencies can be difficult to resolve. Resolving them automatically means that behavior in different models must be unified and the mechanism for this unification must be implemented efficiently. Model-based approaches to user-interface generation use models that are specified in diverse and often incompatible notations. This characteristic complicates model composition because the composition mechanisms in one model may not exist in another.

MASTERMIND solves the model-composition problem by mapping the *inter-model* behavior of each model into a semantic domain that is common across all of the models. This domain is described by the LOTOS notation [7], which specifies temporal constraints on actions and data values. LOTOS specifications can be trivially derived from the text of a model specification. To implement run-time module composition that is correct with respect to model composition, we created a virtual machine that implements the primitives of model composition [53]. The generated modules are then composed by layering them on top of this virtual machine, which resolves the inter-module composition needs of each module. DYNAMO will involve a similar model composition mechanism because it will need to integrate at run-time components that are specified using different paradigms.

- A Language for Modeling User-Application Dialogue

  The MASTERMIND Dialogue Notation (MDL) allows designers to specify the permissible ordering of the end-user interactions that carry out some task [50]. Although the notation is textual, a design tool (Dukas, described below) supports the graphical entry and editing of models. MDL allows designers to organize end-user interactions using a declarative mechanism similar to production rules in context-free grammars. MDL supports five kinds of interaction orderings, each denoted by an operator: alt (alternative), dis (disabling), seq (sequential), par (parallel), excl (exclusive execution), and int (interrupt/resume). The notation also provides operators for expressing preemption. In particular, four operators are available: opt denoting optional, user-specified preemption and cond indicating optional, application-dependent preemption. The same two kinds of preemption are provided for modeling recurring situations: loop denoting user choice, and while indication application choice. An MDL code generator can then convert a dialogue model into C++ classes, which, when linked with the MASTERMIND run-time libraries, produces an executable capable of controlling the interaction of the end user with the user interface and the application.

- Application of Model Checking

  We applied automated model checking to detect flaws in the MDL library design. The architecture of the library allows a programmer to declaratively create user-interfaces with complex temporal behavior (e.g., optionality, interruption and termination). The key to making the library declarative is to provide a set of reusable objects that impose hierarchical orderings over widgets (e.g., buttons, sliders, and text-entry boxes). The base set of widgets–that is, the widgets that are combined using the orderings–come from the Amulet user-interface library. The MASTERMIND library extends the Amulet widgets by providing a set of ordering constructors that aggregate and impose complex temporal orderings upon the underlying Amulet widgets. This design decision simplified the composition of widgets, but it resulted in complex communication protocols in the constructors.

We used SMV [30, 31] (the specification language of the symbolic model checker from Carnegie-Mellon University) as the input language to describe the behavior of each constructor and then ran the model checker to discover flaws in these designs. Using the model checker, we found several design flaws, which were subsequently fixed. Such analysis was possible because the dialog modeling notation has a formal semantics.

- Tools

  The MASTERMIND project has built a series of prototype user-interface generation tools. A designer specifies the features of an interface, including its dialogue, presentation, and application interface, and a program is automatically generated that implements that interface. The MASTERMIND environment is intended to be both a user interface design platform and a delivery platform where run-time services are provided based on the models. In the design environment, we have built a graphical design tool where models can be described to the system using a visual representation, bottom-up drag-and-drop from palette to the design space. We have also built the MASTERMIND run-time environment which can automatically generate interactive help from the model. The environment was designed to support distributed applications using CORBA technology.

  One of the tools developed is the Dukas Graphical Task Modeling tool. The Dukas tool is designed to assist an application designer in creating a task model of their system and using that model to assist in the coding of an application and its interface. Dukas allows designers to develop a graphical tree describing system behavior. This tree is defined in terms of low-level tasks (leaves) and behavioral operators between nodes. These operators are based on the MASTERMIND Dialogue Language notation. Dukas stores this representation in a grammatically simple text format, which can be used by the designer. Used by itself, Dukas can provide a designer with a tool to specify, study and refine a model of the system behavior before implementation begins. The graphical representation also presents an easy tool for collaboration and discussion of system behavior in a development team environment. Using other MASTERMIND system components, Dukas representation supports the automatic generation of C++ code to execute the task model and application. This provides Dukas with the ability to not only work as a tool for assisting analysis and design process, but allows for rapid prototyping of designs and easy maintainability of a complex application system.

5.2.2 MORALE

Background The goal of the MORALE project is to facilitate the evolution of legacy software systems. Facilitation takes the form of improved quality by requirements validation, reduced risk via architectural evaluation and assessment, and increased productivity from maintenance and access to design rationale and from high level reuse of architectural components.

MORALE exploits both high level (mission-oriented) requirements and existing software assets in order to predict the extent to which new system requirements perturb an existing system version. Requirements are derived from mission-oriented goals and are explored through scenarios, and the existing system is analyzed in terms of its response to high-level events traces (scenario instantiations). These inputs are expressed in a scenario-based architectural description language. The common representation enables predictions of architectural distortions as well suggesting reuse candidates.

Project Capabilities The MORALE project has resulted in the development of a unified framework for evolving legacy software systems. The frameworks encompasses requirements elicitation, legacy code analysis, architectural impact assessment, reflective redesign, and user interface migration.

Of particular relevance to DYNAMO are two aspects of MORALE: architectural impact assessment and event-based legacy code analysis.

- Architectural Impact Assessment

  When a development team is confronted with a suggested set of changes, it is important for them to determine the extent to which the changes will affect the overall architecture of the system. Architectural changes are error-prone and expensive. MORALE uses the Software Architecture Assessment Method (SAAM) to determine which components of the system's architecture will need to be altered in order for the system to adapt to the proposed changes.

  The most troubling sorts of changes to an architecture are those that alter its underlying architectural style. DYNAMO is specifically concerned with systems in which multiple styles (paradigms) are present and it provides explicit mechanisms for integrating disparate paradigms.

- Event-based Legacy Code Analysis

  DASADA is particularly concerned with the determination and maintenance of program properties. These properties can be evaluated both statically (by viewing a system's source code) and dynamically (by monitoring its run-time behavior). While static analysis is a mature technology, it is limited in the precision of the information that it can obtain. The MORALE project used event-based dynamic analysis as a supplement to traditional static analysis in order to determine the properties of a system's software architecture.

  The specific MORALE approach to dynamic analysis was to analyze the stream of system events that take place during a program's execution and to infer from the stream abstract inter-component interaction protocols.

  DYNAMO is also concerned with how components interact. For components generated by DYNAMO, the corresponding models can be used to determine system properties. But for preexisting components or for those built elsewhere (e.g. COTS components), it will be necessary to determine those properties. The techniques developed by the MORALE project will be directly applied to this problem.

Project Accomplishments MORALE accomplishments made progress on three fronts: methods, tools, and case studies.

- MORALE Methods

  MORALE comprises a series of methods which together encompass the legacy system evolution life cycle. The following table summarizes them.

Method	Description
ScenIC	Requirements elicitation
Synchronized Refinement	Legacy system analysis
SAAM	Architecture impact assess.
MESA	Reflective redesign
MORPH	User interface migration

- MORALE Tools

  To support the MORALE methods, a suite of tools have been developed for public release. The tools are summarized by the following table.

Name	Description
SIRRINE	Adaptive design critic
MORPH	User interface migration
Dowser	Domain-based software artifact browser
REMORA	Reconcile multiple architectural models
ISVis	Event-based dynamic program analyzer
SAAMPad	Design rationale capture and access
VisEd	Architecture visualization and editing
ACME Server	Multiplatform architecture model server
ScenIC View	Management of requirements scenarios

- MORALE Case Studies

  One case study explored the evolution of the Mosaic Web Browser, version 2.4, to add user-configurable browsers. We applied each of the constituent methods to the problem. A second case study of the MORALE software evolution method was performed at the NORTEL laboratory in Atlanta. At Nortel, we used ScenIC, SAAM, and MORPH to validate our methods.

5.3 Detailed Task Technical Descriptions

We will accomplish the goals of DYNAMO using the following work breakdown structure. In the remainder of the document, we will refer to these tasks by their item number. See § 6 for a schedule of tasks and milestones.

1. Dialog modeling notation: refine, document, and evaluate the MASTERMIND dialog modeling notation. This work will be performed by MSU.

2. EVA version 1: refactor and evolve ISVis into EVA; improve portability; generalize the class of events it is capable of handling. This work will be performed by Georgia Tech.

3. Dialog model code generator: refine and test the MASTERMIND dialog model code generator. This work will be performed by MSU.

4. Presentation modeling notation: define a modeling notation for the presentation aspects of user interfaces based on the MASTERMIND presentation modeling notation, document, and evaluate. This work will be performed by both MSU and Georgia Tech.

5. Presentation code generator: construct a code generator for presentation models; includes selecting a portable GUI toolkit as a target. This work will be performed by MSU.

6. DYNAMO run-time environment, version 1: provides run-time access to models and an API for critics; provides the run-time support for component integration. This work will be performed by both MSU and Georgia Tech.

7. SIMC version 1 - Demo preparation; SIMC is scalable, interactive map component, a graphical interactor capable of displaying geographical areas (terrain maps for battle planning) at a variety of scales and accepting user inputs (redeployments) from a variety of devices. This work will be performed by Georgia Tech.

8. System composer, version 1 (static): provides the ability to integrate the code generated from model code generators described in tasks #3 and #5. Constructs a composite executable and a run-time accessible version of the constituent models. This work will be performed by both MSU and Georgia Tech.

9. Demo (milestone): demonstrate DYNAMO on SIMC and a variety of devices.

10. Display modeling notation: devise a model of display devices (live board, laptop, workstation, PDA) including both presentation and dialog; specifically robust enough to deal with a wide range of scales. This work will be performed by Georgia Tech.

11. EVA version 2: add grammatical inferencing capabilities to EVA so that it is capable of detecting a wider variety of event conditions. This work will be performed by Georgia Tech.

12. Display model code generator: build a code generator for the display modeling notation. This work will be performed by MSU.

13. Run-time critic: build one or more run-time critics capable of analyzing dynamic events and component models and alerting the run-time environment when problems are detected. This work will be performed by both MSU and Georgia Tech.

14. System composer version 2 (dynamic): add to the system composer the ability to deal with new components. This work will be performed by both MSU and Georgia Tech.

15. SIMC version 2 - add an interface to SIMC so that it can talk to a geographic information system in order to more realistically stress the evaluation environment. This work will be performed by Georgia Tech.

16. Demo (milestone): demonstrate DYNAMO on the full evaluation (battle planning) environment

5.4 Transition of Technology

There are three categories of DYNAMO deliverables: new technology, in the form of research results; software tools; and a demonstration/evaluation of DYNAMO on a military application (battle planning). Research results will be delivered using traditional means: journal publications, conference reports, and reports to the sponsor. Tools will be delivered to the sponsor as well as being made publically available using the Internet. Demonstrations will be made to the sponsors and to other military, industrial, and academic personnel.

More specifically, in additional to the traditional technology transfer avenues described in the preceding paragraph, we intended to work with DoD and related industrial customers in the following ways. We have a long history of working with the Army Research Laboratory (ARL). (One branch of ARL is located on the GIT campus.) For this proposal, ARL supplied background materials to the PIs on battle

planning and general Army operations. We intend to continue leveraging ARL's Army contacts to search for potential DYNAMO customers. We also have worked in the past with TACOM (the Army Tank and Automotive Command in Warren, Michigan) and with Spectra Research (a defense contractor in Dayton, Ohio). We also intend to continue to work with them to locate customers.

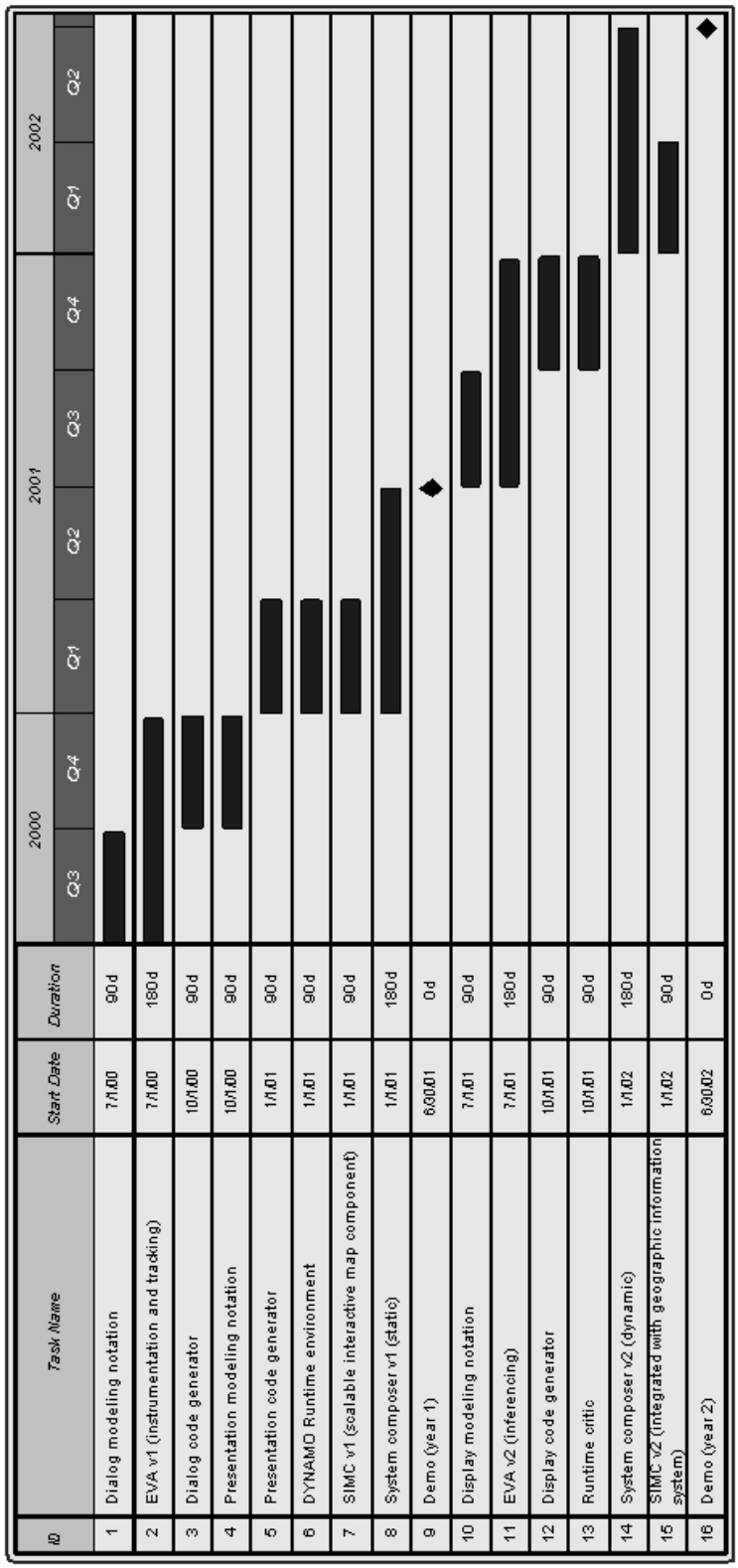
The means of evaluating and demonstrating DYNAMO is to build a battle planning (war game) environment. We intend to use this demonstration as an entry vehicle to potential customers. That is, while demonstrating how modern human-computer interface devices can enhance battle planning, we can introduce and offer evidence of our dynamic assembly technology.

5.5 Qualifications of key personnel

Dr. Spencer Rugaber Spencer Rugaber is a Senior Research Scientist in the College of Computing at the Georgia Institute of Technology. His research interests are in the area of Software Engineering, specifically software evolution and program comprehension. Dr. Rugaber received his Ph. D. in Computer Science from Yale University in 1978. He is currently Principal Investigator in several research projects in this area funded by the National Science Foundation, the Defense Advanced Research Projects Agency, and private industry. Dr. Rugaber is Vice-Chairman of the IEEE Committee on Revere Engineering and past Program Co-chair of the Workshop on Program Comprehension. He is the author of the article on Program Understanding included in the Encyclopedia of Computer Science and Technology and of numerous articles appearing in technical journals and conference proceedings.

Dr. Kurt Stirewalt Kurt Stirewalt received the PhD in Computer Science from the Georgia Institute of Technology in 1997. He is currently an assistant professor in the Department of Computer Science and Engineering at Michigan State University, where he teaches courses in software engineering and formal methods. His primary research concerns the investigation of design abstractions and software-composition mechanisms to support the development of interactive systems with highly graphical user interfaces. Dr. Stirewalt has also conducted research in reverse engineering and program analysis. Prior to joining MSU, he worked as a member of the technical staff at the MITRE corporation. He received a MITRE fellowship three years in a row, and he received a MITRE scholarship in residence to pursue his research in program analysis and software engineering. Other honors include a best-paper award for his work on the analysis of interleaving in legacy applications and a best-paper nomination for his work on the model-composition problem in user-interface generation. Dr. Stirewalt is a member of the IEEE Computer Society.

6 Schedule and Milestones



ID	Task Name	Start Date	Duration
1	Dialog modeling notation	7/1/00	90d
2	EVA v1 (instrumentation and tracking)	7/1/00	180d
3	Dialog code generator	10/1/00	90d
4	Presentation modeling notation	10/1/00	90d
5	Presentation code generator	1/1/01	90d
6	DYNAMO Runtime environment	1/1/01	90d
7	SIMC v1 (scalable interactive map component)	1/1/01	90d
8	System composer v1 (static)	1/1/01	180d
9	Demo (year 1)	6/30/01	0d
10	Display modeling notation	7/1/01	90d
11	EVA v2 (inferencing)	7/1/01	180d
12	Display code generator	10/1/01	90d
13	Runtime critic	10/1/01	90d
14	System composer v2 (dynamic)	1/1/02	180d
15	SIMC v2 (integrated with geographic information system)	1/1/02	90d
16	Demo (year 2)	6/30/02	0d

7 Management Plan

DYNAMO involves researchers from the Georgia Institute of Technology (GIT) and from Michigan State University (MSU). Officially, GIT is the prime contractor, and MSU is a subcontractor. Nevertheless, Rugaber and Stirewalt will work as Co-Principal Investigators (Co-PIs). Rugaber and Stirewalt have worked together for the last five years on the MASTERMIND and MORALE projects.

In addition to the PIs, GIT will hire a research programmer and one graduate student. MSU will employ two graduate students. All participants will be involved in both the research and development of the DYNAMO technology.

Project coordination will leverage the PIs experience on previous collaboration as well as the intersite travel incorporated in the budget. We expect both personnel from both sites to participate in DARPA workshops and demonstrations.

8 Comparison of Research

Analysis of Components One fundamental problem addressed by DYNAMO is the practical intractability of static analysis to reason about dynamic assemblies of components. DYNAMO can assemble two types of components: those which have been generated from models and those that were obtained externally. For the latter case, no model exists, and one must therefore be derived. If source code is available, static program analysis can be performed. Static analysis is a mature field, and a wide variety of analyses and tools exist [42]. Because our demonstration exercise concerns user interface models, we intend to build on the work of Moore and Rugaber in which such models of the user interface of an existing system are derived using a specialized data-flow analyzer [33].

Within the field of static analysis, much attention has been given to the verification of *composition properties*, that is properties that can be proved of a component in composition with other (as yet unknown) components. Work in this area is also quite mature. The earliest approaches formalized component composition using module interconnection languages, which provide a shallow guarantee of composition correctness in terms of modular plug-ability [37]. More powerful analyses use models of the behavior of components to reason about how they will behave under composition (c.f., [16, 17, 61, 12, 13]). With one notable exception [34], these more powerful methods work by exhaustive analysis and can, therefore, only be applied to *finite state models* of a component. Techniques exist to abstract the behavior of a component into a finite state model (c.f., [60]), however the mathematical maturity required to apply these techniques is beyond that of the average programmer.

When static analysis is infeasible or inconvenient, another form of analysis based on event monitoring must be applied. Event monitoring is a form of dynamic analysis, and dynamic analysis is also a mature field. However, it has been primarily applied to areas such as performance analysis and program testing rather than building general models of system behavior or evaluating components for adherence to composition properties. Work using event analysis to reverse engineer programs is limited [2, 4, 18, 23, 29, 1, 19, 21, 28, 27] and has never before been applied to dynamic assembly.

We intend to monitor program events in order to build a model of system behavior. Here we build upon work performed as part of the MORALE project in which we constructed a comprehensive dynamic analysis tool called ISVis [25]. ISVis is capable of building architectural models of system behavior by abstracting from sequences of intercomponent events. The abstraction takes the form of simple pattern matching. For DYNAMO, we will extend ISVis into the EVA tool. In EVA, pattern matching will be much more sophisticated, using a technique called grammatical inferencing [32]. The result is a finite state model capable, in the case of user interfaces, of deriving a human-computer dialog model.

By using models throughout development, DYNAMO enables powerful static analyses, but it also facilitates dynamic analysis through critics to monitor and gauge properties that are impractical to verify statically. We know of no other approach that can so seamlessly apply static and dynamic analysis in this complimentary fashion.

Model-Based Code Generation Automated approaches to interactive system development reduce the complexity of development by raising the level of abstraction in system specification. Issues at the graphics and basic interaction level are addressed by palette-based interface builders; whereas more syntactic issues, such as dialog sequencing, are addressed by User Interface Management Systems (UIMSs) [45, 46, 48, 47, 58]. Finally, semantic issues, such as domain-based UI-component selection, require full UI design environments [20, 54]. Application-generation tools perform well when they only must focus on one kind

of component (i.e., low-level graphics/device interaction, dialog sequencing, or semantics), but no single tool addresses all of these issues well. Automatic-generation must therefore deal simultaneously with information expressed at multiple levels in different notations.

Many different kinds of models have been studied, and different projects have focused on integrating models into the overall development process. While a common theme is to use models to generate (or formally derive) prototype designs [26, 24], models may potentially be used for much more than prototyping. Because a model addresses a single concern in isolation, the notation used to express the model can be specialized and highly declarative. For example, Castells, Szekely, and Salcher present a declarative presentation model that incorporates concepts from graphic design. Such specialized modeling abstractions simplify the expression of presentations and can be compiled into good presentation code [11]. Ideally, a designer could specify the coherent aspects of an interactive system in a collection of specialized models and then invoke a suite of model-specific code generators to generate code from each model. The Mecano [38], Mobi-D [39], and MASTERMIND projects all aim to use models in this capacity.

A distinguishing characteristic of the model-based approach is the use of multiple declarative models in separate notations to specify and then generate an interactive system. Modeling notations incorporate different specification styles or paradigms. Consequently, model-based UI specification is a form of multi-paradigm specification. An inherent obstacle in such an approach is the burden on specifiers of decomposing their problems into multiple specifications that use different paradigms and then to understand the "semantic machinery" through which the specifications interact [64, p. 527]. With the exception of MASTERMIND [52, 53], none of the model-based approaches explicitly specifies the nature of the semantic machinery through which the models compose. The semantic obstacles to *specifying* a system using multiple models are amplified when we then try to generate correct and efficient run-time code from these models. In their work on the composition of multi-paradigm specifications, Zave and Jackson use a unifying formalism (first-order predicate logic) to compose partial specifications written in different paradigms [63]. In MASTERMIND, we extended this approach to address generation and run-time composition using a virtual machine that implements the "semantic mechanisms".

DYNAMO extends the results of MASTERMIND by exploring a new solution to the multi-paradigm specification problem. The semantic obstacles to analyzing multi-paradigm specifications make it very difficult to verify properties of a multi-paradigm specification. However, by deferring some analyses to run-time critics, which are often easy to construct using model-based code generators, DYNAMO reaps the benefits of multi-paradigm specification (i.e., concision, declarativeness) without having to address its fundamental source of complexity.

Traditional software gauges Our emphasis on presentation modeling is also related to the topic of declarative visualization [40]. In PAVANE [41], mathematical transformations can be composed to map application-state representations into views of that state. This idea is similar in spirit to the MASTERMIND presentation model, but the presentation model differs in two fundamental ways. First, because Pavane is a visualization tool, it is not concerned with input and object manipulation as is the presentation modeling language. Second, in Pavane, visualizations are expressed in a functional notation; whereas in MASTERMIND, presentations are specified using an object-oriented notation, similar in spirit to graphical notations, such as the OMT object modeling notation [44] and the UML class/object notation [8].

References

[1] P. Ashton and J. Penny. A tool for visualizing the execution of interactions on a loosely coupled distributed system. *Software-Practice and Experience*, 25(10), October 1995.

[2] M. Auguston. A program behavior model based on event grammar and its application for debugging automation. In *Proceedings of the 2nd International Workshop on Algroithmic and Automated Debugging, (AADEBUG'95)*, 1995.

[3] L. Bass and J. Coutaz. *Developing Software for the User Interface*. SEI Series in Software Engineering. Addison-Wesley, 1991.

[4] P. Bates. Debugging heterogeneous distributed systems using event-based models of behavior. In *Proceedings of a Workshop on Parallel and Distributed Debugging*, pages 11–22, May 1988.

[5] B. Bederson et al. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computation*, 7:3–31, 1996.

[6] B. Bederson and J. Hollan. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. In *Proceedings of UIST*, 1994.

[7] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Comp. Netw. ISDN Sys.*, 14(1), 1987.

[8] G. Booch, I. Jacobson, and J. Rumbaugh. *Unified Modeling Language User Guide*. Addison Wesley, 1999.

[9] T. P. Browne et al. Using declarative descriptions to model user interfaces with MASTERMIND. In F. Paternò and P. Palanque, editors, *Formal Methods in Human Computer Interaction*. Springer-Verlag, 1997.

[10] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the* 5^{th} *International Symposium on Logic in Computer Science*, June 1990.

[11] P. Castells, P. Szekely, and E. Salcher. Declarative models of presentation. In *IUI'97: International Conference on Intelligent User Interfaces*, 1997.

[12] S. C. Cheung and J. Kramer. Context constraints for compositional reachability analysis. *ACM Trans. Softw. Eng. Meth.*, 5(4):334–377, October 1996.

[13] S. C. Cheung and J. Kramer. Checking safety properties using compositional reachability analysis. *ACM Trans. Softw. Eng. Meth.*, 8(1):49–78, 1999.

[14] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite–state concurrent systems using temporal logic. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[15] E. M. Clarke and R. Kurshan. Computer aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.

[16] E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional model checking. In *Fourth Annual Symposium on Logic in Computer Science*, pages 353–362, 1989.

[17] E. M. Clarke, D. E. Long, and K. L. McMillan. A language for compositional specification and verification of finite state hardware controllers. *Proceedings of the IEEE*, 79(9):1283–92, 1991.

[18] J. E. Cook and A. L. Wolf. Automating process discovery through event-data analysis. In *Proceedings of the 17th International Conference on Software Engineering*, Seattle, Washington, April 1995.

[19] S. G. Eick and A. Ward. An interactive visualization for message sequence charts. In *Proceedings of the International Workshop on Program Comprehension*, pages 2–8, March 1996.

[20] J. Foley et al. Uide - an intelligent user interface design environment. In *[55]*, pages 339–384. Addison-Wesley, Reading, Massachusetts, 1991.

[21] R. J. Hall. Call path refinement profiles. *IEEE Trans. Softw. Eng.*, 21(6):481–496, June 1995.

[22] G. J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, May 1997.

[23] W. E. Howden and G. M. Shi. Linear and structural event sequence analysis. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'96)*, pages 98–106, 1996.

[24] A. Hussey and D. Carrington. Model-based design of user-interfaces using Object-Z. In *Proceedings of the Third International Conference on Computer-aided Design of User Interfaces (CADUI'99)*, pages 43–53, 1999.

[25] D. Jerding and S. Rugaber. Using visualization for architectural localization and extraction. In *Proceedings of the Fourth Working Conference on Reverse Engineering*, pages 56–65, October 1997.

[26] H. Johnson and P. Johnson. Task knowledge structures: psychological basis and integration into system design. *Acta Psychologica*, 78, 1991.

[27] K. Koskimies et al. Automated support for modeling oo software. *IEEE Software*, 15(1):87–94, January-February 1998.

[28] D. B. Lange and Y. Nakamura. Interactive visualization of design patterns can help in framework understanding. In *Proceedings of OOPSLA'95*, pages 342–357, 1995.

[29] B. Lazzerini and L. Lopriore. Abstraction mechanism for event control in program debugging. *IEEE*, 15(7), July 1989.

[30] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, 1992. CMU-CS-92-131.

[31] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer, 1993.

[32] Laurent Miclet and Colin de la Higuera. Grammatical inference: Learning syntax from sentences, 1996.

[33] M. Moore. *User Interface Reengineering*. PhD thesis, College of Computing, Georgia Institute of Technology, 1998.

[34] G. Naumovich et al. Verification of concurrent software with FLAVERS. In *International Conference on Software Engineering*, 1997.

[35] R. Neches et al. Knowledgeable development environments using shared design models. In *Intelligent Interfaces Workshop*, pages 63–70, 1993.

[36] Object Management Group. *Common Object Request Broker Architecture and Specification*.

[37] R. Prieto-Diaz and J. M. Neighbors. Module interconnection languages. *Journal of Systems Software*, 6(4), November 1986.

[38] A. Puerta. The Mecano project: Comprehensive and integrated support for model-based user interface development. In *Proceedings of CADUI'96*, 1996.

[39] A. Puerta. A model-based interface development environment. *IEEE Softw.*, 14(4):40–47, 1997.

[40] G.-C. Roman. Declarative visualization. In J. Stasko et al., editors, *Software visualization: Programming as a multi-media experience*, chapter 13. MIT Press, 1998.

[41] G.-C. Roman et al. Pavane: A system for decalarative visualization of concurrent computations. *Journal of Visual Languages and Computation*, 3(1):161–193, January 1992.

[42] S. Rugaber. Program understanding. In A. Kent and J. G. Williams, editors, *Encyclopedia of Computer Science and Technology*, volume 35, pages 341–368. Marcel Dekker, 1996. Supplement 20.

[43] S. Rugaber. A tool suite for evolving legacy software. In *Proceedings of the International Conference on Software Maintenance*, 1999.

[44] J. Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

[45] J. L. Sibert, W. D. Hurley, Y. Chen, and T. W. Bleser. An object centered user interface management systems architecture. In *International Conference on Computer-Aided Design and Computer Graphics (CAD & CG '89)*, pages 125–131, 1989.

[46] G. Singh and M. Green. Generating graphical interfaces from high-level descriptions. In *Graphics Interface'89*, pages 70–77, 1989.

[47] G. Singh and M. Green. Automating the lexical and syntactic design of graphical user interfaces: The uofa$*$ uims. *ACM Transactions on Graphics*, 10(3):213–254, July 1991.

[48] G. Singh, C. H. Kok, and T. Y. Ngan. Druid: A system for demonstrational rapid user interface development. In *UIST'90: Third Annual ACM SIGGRAPH Symposium on User Interface Software Technology*, pages 167–177, 1990.

[49] R. E. K. Stirewalt. *Automatic Generation of Interactive Systems from Declarative Models*. PhD thesis, Georgia Institute of Technology, 1997.

[50] R. E. K. Stirewalt. MDL: A language for binding user-interface models. In *Proceedings of the Third International Conference on Computer-aided Design of User Interfaces (CADUI'99)*, pages 159–170, 1999.

[51] R. E. K. Stirewalt. Separating concerns in direct manipulation user interfaces. In *Proceedings of the IEEE International Conference on Automated Software Engineering*, 1999.

[52] R. E. K. Stirewalt and S. Rugaber. Automating user-interface generation by model composition. In *Proceedings of the IEEE International Conference on Automated Software Engineering*, 1998.

[53] R. E. K. Stirewalt and S. Rugaber. The model-composition problem in user-interface generation. *Automated Software Engineering*, 7(2), April 2000. To appear.

[54] P. Sukaviriya, J. D. Foley, and T. Griffith. A second generation user interface design environment: The model and runtime architecture. In *Bridges Between Worlds: Human Factors in Computing Systems: INTERCHI'93*, 1993.

[55] J. Sullivan and S. Tyler, editors. *Architectures for Intelligent User Interfaces*. Addison-Wesley, Reading, MA, 1991.

[56] P. Szekely et al. Declarative models for user-interface construction tools: the MASTERMIND approach. In Bass and Unger, editors, *Engineering for Human-Computer Interaction*. Chapman & Hall, 1996.

[57] S. Vinoski. Corba: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), 1997.

[58] P. D. Wellner. Statemaster: A uims based on statecharts for prototyping and target implementation. In *ACM CHI'89 Conference on Human Factors in Computing Systems*, pages 177–182, 1989.

[59] S. Wilson et al. Beyond hacking: A model based approach to user interface design. In J. L. Alty, D. Diaper, and S. Guest, editors, *People and Computers VIII, Proceedings of the HCI '93 Conference*, 1993.

[60] J. M. Wing and M. Vaziri-Farahani. Model checking software systems: A case study. In *Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, October 1995.

[61] W. J. Yeh and M. Young. Compositional reachability analysis using process algebra. In *Symposium on Testing, Analysis, and Verification*, pages 49–59, 1991.

[62] P. Zave. A compositional approach to multiparadigm programming. *IEEE Software*, 6(5), September 1989.

[63] P. Zave and M. Jackson. Conjunction as composition. *ACM Trans. Softw. Eng. Meth.*, 2(4):371–411, 1993.

[64] P. Zave and M. Jackson. Where do operations come from? a multiparadigm specification technique. *IEEE Trans. Softw. Eng.*, 22(7):508–528, July 1996.

Volume IA - Appendix

A Personnel

Dr. Spencer Rugaber is a Senior Research Scientist in the College of Computing at the Georgia Institute of Technology. His research interests are in the area of Software Engineering, specifically software evolution and program comprehension. Dr. Rugaber received his Ph. D. in Computer Science from Yale University in 1978. He is currently Principal Investigator in several research projects in this area funded by the National Science Foundation, the Defense Advanced Research Projects Agency, and private industry. Dr. Rugaber is Vice-Chairman of the IEEE Committee on Revere Engineering and past Program Co-chair of the Workshop on Program Comprehension. He is the author of the article on Program Understanding included in the Encyclopedia of Computer Science and Technology and of numerous articles appearing in technical journals and conference proceedings.

Dr. Kurt Stirewalt is an Assistant Professor in the Department of Computer Science and Engineering at Michigan State University, where he teaches courses in software engineering and formal methods. His primary research concerns the investigation of design abstractions and software-composition mechanisms to support the development of interactive systems with highly graphical user interfaces. Dr. Stirewalt has also conducted research in reverse engineering and program analysis. Prior to joining MSU, he worked as a member of the technical staff at the MITRE corporation. He received a MITRE fellowship three years in a row, and he received a MITRE scholarship in residence to pursue his research in program analysis and software engineering. Other honors include a best-paper award for his work on the analysis of interleaving in legacy applications and a best-paper nomination for his work on the model-composition problem in user-interface generation. Dr. Stirewalt is a member of the IEEE Computer Society.

Rugaber and Stirewalt have worked together continuously on various projects since 1993. Their collaborative results have impacted the areas of program analysis, reverse engineering, automated software composition, and user interface design. Both researchers have extensive industrial experience in real software design. The remainder of this section comprises the vitae for these researchers. Rugaber will devote thirty three (33) percent of his calendar year effort to DYNAMO research. Stirewalt will devote thirty five (35) percent of his calendar year effort to DYNAMO research.

SPENCER RUGABER
Senior Research Scientist
College of Computing
Georgia Institute of Technology

Educational Background:

Ph.D. 1978 Yale University Computer Science M.S. 1971 Harvard University Applied Mathematics B.S. 1970 Yale University Engineering and Applied Science

Employment History:

Academic Experience:

Senior Research Scientist, College of Computing, Georgia Institute of Technology, 1993-Present.

Research Scientist II, College of Computing (formerly School of Information and Computer Science), Georgia Institute of Technology, 1988-1993.

Industrial Experience:

National Advanced Systems Corporation, Senior Software Engineer, Atlanta, Georgia, 1987-1988.

Unisys Incorporated (formerly Sperry Corporation), Professional Consultant, Atlanta, Georgia, 1985-1986.

Interactive Systems Corporation, Senior Technical Staff, Estes Park, Colorado, 1981-1985.

Bell Telephone Laboratories, Member of the Technical Staff, Murray Hill and Whippany, New Jersey, 1970-1974, 1977- 1981.

Current Fields of Interest:

Software Engineering, Software Evolution and Maintenance, Reverse Engineering and Program Comprehension, Programming Languages, Human-Computer Interaction.

Research and Creative Scholarship

Thesis

A Model of the Understandability of Computer Programs, 1978, Alan J. Perlis advisor, Yale University, Ph.D. Thesis.

Published Journal Papers (refereed)

"Validation and Verification of Reusable Ada Software," with C. Bullard, D. Guindi, W.B. Ligon, and W.M. McCracken. Empirical Foundations of Information and Software Sciences V, Pranas Zunde and Dan Hocking, editors, Atlanta, GA, 1988, pp. 437-454.

"Recognizing Design Decisions in Programs," with Stephen B. Ornburn and Richard J. LeBlanc, Jr., IEEE Software, Vol. 9, January 1990, pp. 46-54.

"Requirements Validation via Automated Natural Language Parsing," with Sastry Nanduri, Journal of Management Information Systems, 12(3): 9-19, Winter 1995-96.

"Understanding Interleaved Code," with Kurt Stirewalt and Linda Wills, Automated Software Engineering, 3(1/2):47-76, June 1996.

"Restoring a Legacy: Lessons Learned," with Jim White, IEEE Software, 15(4):28-33, July-August 1998.

"Using Visualization for Architectural Localization and Extraction, with Dean Jerding, Science of Computer Programming, to appear.

"The Use of Domain Knowledge in Program Understanding," Annals of Software Engineering, to appear.

"The Model Composition Problem in User Interface Generation," with Kurt Stirewalt, Journal of Automated Software Engineering, to appear.

Published Books and Parts of Books

"Validation and Verification of Reusable Ada Software," with C. Bullard, D. Guindi, W.B. Ligon, and W.M. McCracken, in P. A. Lesslie, R. O. Chester and M. F. Theofanos, editors, Guidelines Document for Ada Reuse and Metrics, Martin Marietta Energy, Systems for U.S. Army Institute for Research in Management Information, Communications and Computer Science, 1988.

"Recognizing Design Decisions in Programs," with Stephen B. Ornburn and Richard J. LeBlanc, Jr., in Robert S. Arnold, Software Reengineering, IEEE Computer Society Press 1993.

"Understanding Interleaved Code," with Kurt Stirewalt and Linda Wills, in Reverse Engineering, Linda M. Wills and Philip Newcomb editors, Kluwer Academic Publishers, June 1996.

"Program Understanding," Allen Kent and James G. Williams editors, Encyclopedia of Computer Science and Technology, Marcel Dekker, Volume 35, Supplement 20, 1996.

"Using Declarative Descriptions to Model User Interfaces with MASTERMIND". Thomas Browne, David Davila, Spencer Rugaber, and Kurt Stirewalt. Formal Methods in Human Computer Interaction, Fabio Paterno and Philippe Palanque, editors, Springer-Verlag, 1997.

Edited Proceedings

Proceedings Third Workshop on Program Comprehension, with Aniello Cimitile, IEEE Computer Society, 1994.

Proceedings of the Sixth Working Conference on Reverse Engineering, with Michael Blaha and Francoise Balmas, IEEE Computer Society, 1999.

Conference Presentations

”Programming with Idioms in APL” with Alan J. Perlis, Proceedings of APL International Conference, Rochester, NY, 1979, pp. 232-235.

”The Impact of Verification of Reusable Components on Software Productivity” with W.M. McCracken and D.S. Guindi, Proceedings of the Hawaii International Conference on System Sciences, Kailau-Kona, Hawaii, January 1989.

”Reuse and the Software Lifecycle” with D.S. Guindi and W.M. McCracken, The Annual National Conference on Ada Technology, Atlantic City, NJ, March 1989.

”Fault Localization Using Execution Traces” with Margaret Francel, Proceedings of the 30th Annual ACM Southeast Conference, Raleigh, North Carolina, April 8-10, 1992, pp. 60-76.

”A Quick Tools Strategy for Program Analysis and Software Maintenance” with B. Johnson and S. Ornburn, Proceedings of the Conference on Software Maintenance, Orlando, Florida, November 1992

”Reverse Engineering: Resolving Conflicts between Expected and Actual Software Designs” with S. Ornburn, Proceedings of the Conference on Software Maintenance, Orlando, Florida, November 1992.

”The Representation Problem in Reverse Engineering” with Richard Clayton, Proceedings of the First Working Conference on Reverse Engineering”, Baltimore, Maryland, May 21-23, 1993.

”The Transition of Programs from COBOL to a Fourth Generation Language,” with Sri Doddapenini, International Conference Software Maintenance, September, 1993, Montreal, Canada.

”Domain Analysis and Reverse Engineering,” with Jean-Marc DeBaud and Bijith M. Moopen, Proceedings of the 1994 International Conference on Software Maintenance, Victoria, British Columbia, Canada, September 19-23, 1994, 326-335.

”Knowledge-based User Interface Migration,” with Melody Moore and Phil Seaver, Proceedings of the 1994 International Conference on Software Maintenance, Victoria, British Columbia, Canada, September 19-23, 1994, 72-79.

”Requirements Validation via Natural Language Parsing,” with Sastry Nanduri, Proceedings of the 28th Hawaii International Conference on System Sciences, Wailea, Maui, Hawaii, January 3-6, 1995. ”Reengineering Army Software to an Open Systems Environment,” with Melody Moore and Reginald Hobbs, Proceedings of the Seventh Annual Software Technology Conference, Salt Lake City, Utah, April 9-14, 1995.

”The Interleaving Problem in Program Understanding,” with Kurt Stirewalt and Linda Wills, 2nd Working Conference on Reverse Engineering, Toronto, Ontario, Canada, July 14-16 1995, 166-175.

”A Software Re-engineering Method Using Domain Models,” with Jean-Marc DeBaud, International Conference on Software Maintenance, Opio (Nice), France, October 16-20, 1995.

”Detecting Interleaving,” with Kurt Stirewalt and Linda Wills, International Conference on Software Maintenance, Opio (Nice), France, October 16-20, 1995, 265-274.

”Creating a Research Infrastructure for Reengineering,” with Linda Wills, 3rd Working Conference on Reverse Engineering, November 8-10, 1996, Monterey, California.

”MORALE-Mission Oriented Architectural Legacy Evolution”. Gregory Abowd, Ashok Goel, Dean F. Jerding, Michael McCracken, Melody Moore, J. William Murdock, Colin Potts, and Linda Wills.

Proceedings International Conference on Software Maintenance'97, Bari, Italy, September 29-October 3, 1997.

"Using Visualization for Architectural Localization and Extraction", with Dean Jerding. October 6-8, 1997, Proceedings of the Fourth Working Conference on Reverse Engineering Amsterdam, The Netherlands, IEEE Computer Society.

"Domain Analysis for Transformational Reuse" with Melody Moore, October 6-8, 1997, Proceedings of the Fourth Working Conference on Reverse Engineering, Amsterdam, Netherlands, IEEE Computer Society.

"Using Knowledge Representation to Understand Interactive Systems" with Melody Moore, May 28-30, 1997, Proceedings of the Fifth International Workshop on Program Comprehension, Dearborn, Michigan, IEEE Computer Society Press.

"A Case Study of Domain-based Program Understanding" with Richard Clayton, Lyman Taylor, and Linda Wills, May 28- 30, 1997, 5th International Workshop on Program Comprehension, Dearborn, Michigan.

"Dowsing: A Tools Framework for Domain-Oriented Browsing of Software Artifacts" with Richard Clayton and Linda Wills, May 1998, Proceedings of the Conference on Automated Software Engineering 1998, Honolulu, Hawaii.

"On the Knowledge Required to Understand a Program" with Richard Clayton and Linda Wills, October 1998, Fifth IEEE Working Conference on Reverse Engineering'98, Honolulu, Hawaii.

"Automating UI Generation by Model Composition" with Kurt Stirewalt, October 1998, Proceedings of Automated Software Engineering 1998, Honolulu, Hawaii.

"The Relationship of Slicing and Debugging to Program Understanding," with Margaret Francel, May 1999, Proceedings of the 7th International Workshop on Program Comprehension, Pittsburgh, Pennsylvania.

"Architectural Element Matching Using Concept Analysis," with Bob Waters and Gregory Abowd, to appear in the , Proceedings of Automated Software Engineering 1999, Cocoa Beach, Florida.

"A Tool Suite for Evolving Legacy Software," to appear in the Proceedings of the International Conference on Software Maintenance, Oxford, UK, August 1999.

Other

"Design Decision Analysis Research Project", with K. Kamper, GIT-SERC-90/01, Software Engineering Research Center, Georgia Institute of Technology, January 28, 1990.

"Joint Georgia Tech/BNR, Inc., Reverse Engineering Project", with K. Kamper and Software Engineering Research Center, School of Information and Computer Science, September l8, l990.

"ISA 97-Compliant Architecture Testbed (ICAT) Project Tools and Database Transition", with B. Johnson, and G. Pardun. SRC-TR-92/01, Software Engineering Research Center, Georgia Institute of Technology, January 1992.

"Knowledge Worker Platform Analysis Final Report", with Melody Moore and Hernan Astudillo, Center for Information Management Research, College of Computing, Georgia Institute of Technology, 1993.

”Transitioning to the Open Systems Environment (TRANSOPEN) Final Report”, with Melody Eidbo, Mostafa Ammar, Russ Clark, Rich Clayton, Srinivas Doddapaneni, Ron Dodge, Mike McCracken, Binh Nguyen, Webb Roberts, and Steve Rogers. Center for Information Management Research, College of Computing, Georgia Institute of Technology, April 14, 1993.

”NASA Final Report, The Detection and Extraction of Interleaving Code Segments”, with Kurt Stirewalt and Linda Wills, GIT-CC-95-49, College of Computing, Georgia Institute of Technology, December 20 1995.

”Domain Based Design Documentation and Component Reuse and their Application to a System Evolution Record Final Report”, with Linda Wills and Richard Clayton, October 31, 1997.

”MASTERMIND: Knowledgeable Development Environments for Human-Computer Interaction Software,” November 1998.

Research Honors and Awards

Outstanding Contribution over Time: 4th Working Conference on Reverse Engineering, 1997.

Best paper award: International Conference on Software Maintenance, 1995.

Best paper nominations: Hawaii International Conference on System Sciences, 1995; Automated Software Engineering Conference, 1998.

Best-submitted pattern: CHI Workshop on Patterns, 1997.

R. E. Kurt Stirewalt
Assistant Professor of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824

Degrees

Ph.D. in Computer Science, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, 1997.
Dissertation: *Automatic Generation of Interactive Systems from Declarative Models*
Thesis Advisor: Dr. Spencer Rugaber

M.S. in Information and Computer Science, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, 1991.

B.S. with honors in Information and Computer Science, Department of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia, 1989.

Professional Experience

Assistant Professor (January, 1998 to present), Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan.

Research Scientist I, (1995–1997), Graphics Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, Georgia.

Graduate Research Assistant, (1994–1995), College of Computing, Georgia Institute of Technology, Atlanta, Georgia.

Member of the Technical Staff, (1993–1994), The MITRE Corporation, McLean, Virginia.

Graduate Research Assistant, (1989–1992), College of Computing, Georgia Institute of Technology, Atlanta, Georgia.

Software development consultant/programmer, (1988), The Coca-Cola Company, Atlanta, Georgia.

Lead programmer, (1984-1986), National Financial Computer Systems, Atlanta, Georgia.

Honors and Fellowships

1999 Selected to participate in the 1999 Science and Engineering Education Scholars Program.
1998 Nominated best paper at the IEEE Conference on Automated Software Engineering.
1995 Awarded best paper at the International Conference on Software Maintenance.
1993 Awarded best paper in MITRE Journal.
1993 MITRE fellowship in residence.
1992 MITRE fellowship.
1991 MITRE fellowship.

Grants

NSF CCR-9901017, "Object-oriented development and analysis of embedded systems," (with Betty H. C. Cheng). 8/16/99-7/31/02 for $365,000.

MSU All University Research Initiation Grant (AURIG), "Integrating Protocol Constraints into a User-interface Toolkit", 1998-1999.

Publications

"The Model Composition Problem in user-interface generation," (with Spencer Rugaber), to appear in *Automated Software Engineering* 7(1) 2000.

"Separating concerns in direct manipulation user interfaces," Appears in *Proceedings of the IEEE International Conference on Automated Software Engineering*, October, 1999.

"Model based user-interface reengineering," (with Karmen Tucker), Appears in *Proceedings of the Sixth Working Conference on Reverse Engineering (WCRE'99)*, October, 1999.

"MDL: A language for binding user-interface models," Appears in *Proceedings of the Third International Conference on Computer-Aided Design of User Interfaces (CADUI'99)*, October, 1999.

"Layering direct-manipulation user interfaces with see-through components," Technical Report MSU-CSE-99-27, Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, 1999.

"Composition property analysis: a new strategy for model checking user-interface designs," (with G. D. Abowd), Technical Report MSU-CSE-99-30, Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, 1999.

"Automating UI generation by model composition," (with Spencer Rugaber), appears in *Proceedings of the IEEE International Conference on Automated Software Engineering*, November, 1998.

"Practical dialogue refinement," (with Gregory D. Abowd), appears in *Proceedings of the Fifth International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'98)*, June 1998.

"The MASTERMIND user-interface generation project," (with T. Browne, D. Davila, and S. Rugaber), appears in, *Applications of Formal Methods in Human Computer Interaction*, F. Paterno, G. Abowd, and P. Palanque (Editors), Springer–Verlag Publishers, June 1997.

Automatic Generation of Interactive Systems from Declarative Models, Ph.D. dissertation. Georgia Institute of Technology. December, 1997.

"Design and implementation of the MASTERMIND Toolkit," Technical Report GIT-GVU-97-23, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, Georgia, 1997.

"Understanding interleaved code," (with Spencer Rugaber and Linda Wills), appears in *Automated Software Engineering* vol. 3, no. 1, June 1996.

"Detecting Interleaving." (with Spencer Rugaber and Linda Wills), appears in *Proceedings of the International Conference on Software Maintenance* (awarded best paper), October 1995.

"The interleaving problem in program understanding," (with Spencer Rugaber and Linda Wills), appears in *Proceedings of the Second Working Conference on Reverse Engineering*, July 1995.

"Spatial-temporal analysis of program dependence graphs," (with Helen Gill, Tom Smith, Tom Gerasch, Carolyn McCreary, and John Warren), appears in *Journal of Parallel and Distributed Computing*, April 1993.

"On-the-fly generation of static single-assignment form from an incremental dependence framework," (with W. F. Appelbe and D. H. Gill), Technical Report GIT-CC-92/58, College of Computing, Georgia Institute of Technology, December, 1992.

"PATCH—A New Algorithm for Rapid Incremental Dependence Analysis," (with W. F. Appelbe and K. Smith), appears in *Proceedings of the International Conference on Supercomputing*, June 1991.

"Incremental Dependence Analysis for Interactive Parallelization," (with W. F. Appelbe and K. Smith), appears in *Proceedings of the International Conference on Supercomputing*, June 1990.

B Corporate Experience

See the resumés of the primary offeror and sub-contractor for details of industrial experience. See § 5.2 for details on previous DARPA funded projects in which the primary offeror and sub-contractor have participated.

C Facilities

The research project will also make use of existing computing facilities of the College of Computing at the Georgia Institute of Technology, and the Software Engineering and Network Systems Laboratory at Michigan State University.

Georgia Tech The College of Computing at the Georgia Institute of Technology provides an ideal environment for the proposed research because of the extensive computational facilities available. The existence of the Graphics, Visualization, and Usability Center (GVU) within the College provides strong evidence of the major commitment the Institute has made towards furthering research and education in Computing. The College maintains a variety of computer systems for general support of academic and research activities. These include more than fifty Sun, Silicon Graphics, and Intel systems used as file and compute servers, ten of which are quad-processor machines and more than 600 workstation class machines from Sun, Silicon Graphics, Intel, and Apple.

All of the College's facilities are linked via local area networks which can provide a choice of switched 10 mbps Ethernet, switched or shared 100 mbps Ethernet, 155 mbps ATM, 622 mbps ATM, or 1000 mbps Ethernet connections to most locations, including offices, labs, and classrooms. The College's network employs a high-performance OC12C ATM and Gigabit Ethernet backbone with connectivity to the campus backbone provided by an OC12C ATM link and a redundant OC3C ATM (155 mbps) link for failover. The primary campus Internet connection is provided by a direct FDDI (100 mbps) link to the service provider's Atlanta switching center, augmented by an OC3C ATM connection to the NSF vBNS (very high performance Backbone Network Service) research network. Georgia Tech is also leading efforts to establish a southern regional gigabit network as part of Internet2.

Michigan State The Department of Computer Science and Engineering (CSE) at Michigan State University presently operates six research laboratories: Advanced Computer Systems Laboratory, Autonomous Agents Laboratory, the Intelligence Systems Laboratory, High-Speed Networking and Performance Laboratory, Pattern Recognition and Image Processing Laboratory, and Software Engineering and Network Systems (SENS) Laboratory. Each of these research laboratories is equipped with software and hardware to facilitate problem-solving for a given research area; many of the laboratories were established or enhanced by NSF equipment grants.

The proposed DYNAMO project will be conducted in the Software Engineering and Network Systems (SENS) Laboratory. The Laboratory contains a testbed for mobile/distributed computing, which includes a cluster of symmetric multiprocessors obtained with a prior NSF/CISE Instrumentation Grant (CDA-9617310), comprising 14 Intel-based and Sun SMP computers, connected by a Cisco Catalyst 2924 Fast Ethernet switch. Most of the Intel systems in the cluster are configured for dual boot in both Windows NT and Linux. These systems are used collectively as a network of workstations (NOW) to support research in parallel processing, distributed computing, and networking, as well as individually to support research in software engineering, model-based user-interface generation, and formal verification. Since its establishment in 1997, the SMP cluster has been used to support a wide variety of research projects, including simulation of network protocols, testing of group communication libraries, parallel processing applications, and experimental testing of distributed object-oriented software. The Laboratory also contains six additional workstations and PCs purchased by the Department or under federal/industrial research grants – three are quite old and are effectively retired.

The SENS Laboratory testbed also includes several mobile computers, specifically three HP 660 hand held systems (running Windows CE) and three Dell laptop computers (bootable in either Windows NT or Linux). The SENS Laboratory is home to two wireless LANs, Lucent WaveLAN and Proxim RangeLAN2 (due to interference, usually only one of the networks is active at any given time). Both networks are 802.11 compliant and use CSMA/CA access control. The WaveLAN network uses direct sequence spread spectrum signaling and has a raw bit rate of 2 Mbps. The RangeLAN2 network uses frequency hopping and has a raw bit rate of 1.6 Mbps. In addition wireless PCMCIA cards for mobile computers, we also have several ISA cards for the PCs and workstations in the Laboratory (in the case of WaveLAN, enough to accommodate all our systems), enabling testing of wireless protocols on over 20 nodes. The combination of relatively slow and noisy wireless channels with fast and low-loss wired LANs, as well as the variety of computing platforms, provides an excellent testbed for studying network performance and software interoperability issues. In addition, the SENS Laboratory testbed is connected to an NSF-sponsored ATM testbed and several large instructional laboratories that can be used in research when not scheduled for courses. Finally, working with the College of Engineering, we are currently constructing an 11 Mbps multi-cell wireless LAN for our Engineering Library. In addition to its primary function in providing students with wireless access within the library, this facility will also be available for experimentation.

Note to reviewers: The subcontractor has recently submitted an instrumentation proposal to NSF/EIA in order to further enhance the SENS testbed. However, we emphasize that the research activities described in the this proposal are oriented towards our *existing* testbed and do not depend on the funding of the instrumentation proposal.

D Government Furnished Property

Not applicable