ITO 2001
PROJECT SUMMARY COLLECTION

Verify a Technical Report

Attention: Your report has not been accepted yet!

Please review your report for accuracy, grammar and spelling errors. If you are satisfied with your report then click the "Submit" button at the bottom of this page. If you wish to make changes to your report then click the "Back" button on your browser.

Project Title:	DYNAMO: DYNamo Assembly from MOdels
Organization:	Georgia Institute of Technology
AO Number:	K504
Contract Number:	F30602-00-2-0618
Start Date:	28 JUL 2000
End Date:	28 JUL 2002

Principal Investigator	
Name:	Spencer Rugaber
Address:	College of Computing Georgia Institute of Technology
City, State Zip:	Atlanta, GA 30332-0280
Phone:	404-894-8450
Fax:	404-894-9442
Email:	spencer@cc.gatech.edu
Level Of Participation - Billed:	33%
Level Of Participation - Unbilled:	0%
Name:	Kurt Stirewalt
Address:	Department of Computer Science and Engineering Michigan State University
City, State Zip:	East Lansing, MI 48824
Phone:	(517)-355-2359
Fax:	(517)-432-1061
Email:	stire@cse.msu.edu
Level Of Participation - Billed:	20%
Level Of Participation - Unbilled:	10%
Financial POC	
Name:	David Hendrix
Address:	Office Contact Administration 400 10th Street Room 246
City, State Zip:	Altanta, GA 30332-0420
Phone:	(404) 894-6939
Fax:	404-894-6956
Email:	dave.hendrick@oca.gatech.edu
Project URL:	http://www.cc.gatech.edu/dynamo

Objective:	The DYNAMO project is concerned with assembling high-assurance systems from components, specifically, with guaranteeing correct synchronization of sets of large, heterogeneous components. Several problems that must be overcome to provide such guarantees: 1. dealing with the sheer complexity of the individual components and their interoperation; 2. maintaining design integrity and information hiding of the individual components; 3. providing the desired guarantees, either statically (i.e., during design) or at run time; and 4. not compromising efficiency while accomplish the other goals.
Approach:	DYNAMO addresses these problems with several techniques: 1. a layered architecture limits complexity by reducing the amount of allowed interactions; 2. a declarative specification mechanism abstracts away low-level details, such as event broadcast/handling and variable updates; 3. static analysis of component designs are augmented with run-time status monitoring and gauges; and 4. generic (compile-time) component-wrapper generation removes expensive, inter-layer procedure calls. We apply each of these techniques simultaneously using a design abstraction called a mode component. A *mode component* is a hierarchical (i.e., layered) software component whose modular interface provides a continuously updated view of its current status. Clients (i.e., higher-level components) of a mode component can interact with it by directly observing and triggering reactions to changes in this status view. These facilities enable the designer to compose mode components in a way that supports correct, synchronous inter-operation. Such a specification is more abstract (and therefore easier to write and to reason about) than an equivalent specification phrased as a collection of message-passing event protocols. Mode components extend traditional modular interfaces with extra information, which is useful during static analysis. Consequently, designs built using mode components admit the detection of many errors statically, i.e., at deployment time as opposed to at run time. Moreover, the provision of continuously updated status simplifies the assembly and composition of run-time gauges, which allow even higher levels of assurance. In

	DYNAMO, designers are charged with thinking about the non-functional aspects of a component during component design, so that this information can be made available as status. We expect this to add discipline and rigor to the initial design of a component so that the designer will be contemplating "what to gauge" from the beginning. A mode component provides status information to components above it in a layered architecture. The state of those components is updated automatically when changes occur to any of the status information in the mode component. The nature of the update is expressed declaratively without requiring detailed specification of event synchronization. Client components can also make service requests to a mode component using traditional method calls. Mode components are implemented using C++ templates to wrap traditional class definitions. The templates are generated automatically from a design specification provided in ArgoUML. Use of templates and other advanced C++ features implements the specification without compromising the performance of the assembly of components.

Recent Accomplishments:	• Proof-of-concept example of mode components in various assemblies. The example is a text browser that can be controlled using a scroll bar, and various facts about which can be displayed using one of many reusable gauges. We demonstrated this example in Baltimore.
	• A product-line assembly tool (called **PLAT**) that supports the static (i.e., design- time) assembly of mode components. PLAT integrates a user-interface for component selection with a UML-based editor in which the designer may specify composition constraints. PLAT also supports some degree of static assurance checking by employing the SMV model checker to verify composition properties of mode-component assemblies.
	• Library of reusable mode-component gauges, some of which are output only (e.g., dials and textual displays), others of which allow input as well.
	• Several major pieces of tool infra-structure, that are useful not only in DYNAMO, but also in other products and projects:
	    ○ **Para:** XMI, the output from the ArgoUML modeling tool, is verbose and difficult to access from analysis and code-generation tools. Para is an intermediate representation and API that provides easy access to ArgoUML designs. It can be used independently of DYNAMO as a front end to any tool wishing access to designs expressed in XMI.
	    ○ **ParaGen:** ParaGen is a translator for converting XMI into the Para intermediate representation. It uses the

	Xerces XMI parser to analyze XMI designs. An API is provided for tools to access designs in Para format. ParaGen can also produce output into a text file for external perusal or scripted post-processing. ○ **Para2SMV:** DYNAMO provides design and run-time guarantees of synchronization behavior of assemblies of components. One example of design analysis is model checking. Para2SMV is a tool for converting a design expressed in Para into the input format of SMV, a symbolic model checker. ○ SmvModel: SmvModel is a collection of C++ classes that implement a meta-model of the SMV language. These classes can be used to help generate SMV input files. They are used internally to DYNAMO by Para2SMV, but may be used by any tool wishing to generate SMV input files.
Current Plan:	• Extend PLAT with the ability to process and generate composition code from designer-specified constraints. Currently, PLAT allows the designer to choose from among several "pre-fabricated" constraints. • Develop the dynamic assembler, which allows for the run-time composition of mode components, in accordance with designer-specified composition constraints that are not known until run time. • Extend our library of mode-component gauges.
Technology Transition:	We have spoken with two groups about the transition of DYNAMO technology to service organizations: Lydia Shen at the SPAWAR Systems Center and Mike Saboe of TACOM. We demonstrated our tools to both of them in Baltimore and are exploring the role that we might play in subsequent demonstration projects.
Comments / Questions / Anything else you need:	
Submit Technical Report	