Title: Support for Compiling High-level Access Interfaces for Multi-site Software (CHAIMS)

Submitted to Siemens <>

Technical contact: Professor Gio Wiederhold
Computer Science Department
Stanford University
Stanford CA 94305
phone: 415 725-8363; fax: 415 725-2588
email: gio@cs.stanford.edu

Administrative contact: Marianne Siroker
Computer Science Department
Stanford University
Stanford CA 94305
phone: 415 723-0872; fax: 415 725-2588
email: siroker@cs.stanford.edu

	FY1997
Total:	$ 35,000

prepared 19 May 1996.

A: Innovative Claims.

This proposal requests funding to complement mega-programming research efforts being funded by Defense Advanced Research Projects Agency (DARPA) starting in FY 96 and expected to continue to FY2000. The CHAIMS effort will develop a very-high level (*mega*-) programming language for software module composition. Its compiler will generate a variety of invocation sequences for current and developing standards for software interoperation, with a focus on multi-computer, distributed operation. The language will include the ability to set up module interfaces prior to executions, request performance estimates from modules prior to their invocation, schedule modules in parallel, monitor execution of invoked modules, interrupt inadequately performing modules, and provide data and meta-information to customer interface modules [WiederWC:92].

CHAIMS supports a paradigm shift which is already occurring: a move from coding as the focus of programming to a focus on composition. This shift is occurring invisibly to many enterprises, since there is no clear boundary in moving from subroutine usage to remote service invocation. There are hence few tools and inadequate education to deal with this change. Ten years ago, integration of large-scale software was performed by experienced groups in large companies, as in IBM, UniSys, Fujitsu, Arthur Andersen and the other `Big Five', and system contractors as SAIC, Lockheed, MITRE, Lincoln Labs, etc. Today, software composition has moved to UNIX and PC platforms, and an increasing fraction of the software workforce is engaged in composition. CHAIMS intends to overcome the gap in composition tools.

Many programming languages have had some composition facilities added to their basic capabilities, making them unwieldy. The CHAIMS mega-programming language will only serve module composition and scheduling and remain simple, although it has to introduce some significant concepts. A focus on the issues critical to system integration will allow a high capability to be attained without superseding important facilities provided by legacy and evolving languages. Modules written in C, C++, Ada, Fortran, etc., will need Interfaces (similar to APIs) to allow interoperation in a dynamic, multi-site setting. Such interfaces are specified by interoperation standards as CORBA, DCE, and KQML. These protocols provide, in effect, our machine languages and make the CHAIMS concept implementable today, whereas it was not, say, three years ago.

By focusing CHAIMS on interoperation in a multi-site environment, rather than on platform-specific code, we gain high-level support of mega-programming concepts:

1. Flexibility of interface standard choice, changeable late by only megaprogram recompilation.
2. Composition clarity by making instances of domain-model architectures obvious.
3. Testing of new and updated software without committing to specific configurations.
4. High system performance since asynchronous and parallel operation is the default.
5. Life-time maintenance as a seamless continuation of system development.
6. Adaptability of applications to evolving interface and product standards.
7. Optimization and adaptation to change of scale, modules, and network configuration.

Each of these claims is elaborated below in Section D. However, it must be realized that module performance, although not addressed directly by CHAIMS, continues to critically affect system performance. In a multi-site system composed via a CHAIMS mega-program, individual module performance can be improved by allowing a choice of platforms and module programming languages.

B: Deliverables.

This project will be executed in a sequence of prototype development cycles; there are deliverables with each cycle. Support from Siemens is requested for the first cycle. We hope that the results will be so that Siemens will be motivated to continue research funding as the CHAIMS project evolves.

We do not expect to invoke proprietary claims on the language or its prototype compiler at Stanford. We will publish the results of each cycle, including metrics obtained, rapidly on the worldwide-web, and subsequently in formal form. The most effective form of dissemination we can envisage are our graduates moving to companies that are not risk-adverse. The sponsors of this research will be given direct access to web pages which will document progress as it is evolving.

Specific Deliverables, based on an initiation of funding 1 Oct 1996:

<<to be matched to revised proposal >>

Cycle 1: 31 Dec.1996: Specification of the CHAIMS language. The report will be delivered to Siemens.

Cycle 2: 30 August 1997: CHAIMS initial demonstration. The analysis of the results will be delivered to Siemens
<<>>,

Professor Wiederhold will, at a mutually agreed time, present the result of this research to an audience selected by Siemens.

C: Statement of Work.

We will develop, assess, and demonstrate a mega-programming language, CHAIMS, whose function is to support integration of existing modules. The modules themselves may use a variety of interface standards as promulgated today, as CORBA, OLE, KQML, DOE, DCE, PCTE, etc. The binding of standards selection of interfaces can be delayed arbitrarily, and changed by recompilation of the CHAIMS program. The proposed language will allow composition of such modules operating on a variety of platforms in a distributed network of computing resources. Recompilation of a CHAIMS program will allow network configurations to be changed.

The components of CHAIMS are symbolically depicted in Fig.1. The topmost component is the (mega-) programmer, who selects the modules to be invoked. The mega-program, written by the mega-programmer, only manages the invocation of the modules, according to precedence constraints that typically represent dataflow dependencies. Any needed conversions to be performed in the dataflow are invoked externally to the mega-program, assuring that the mega-program is a clean representation of the architectural intent for this application. The mega-program is to be compiled by the CHAIMS compiler, the resulting executable program consists mainly of module invocations, expressed using the interface standard protocols that are appropriate for this module. The actual modules are either pre-existing or written to order in a suitable programming language. The autonomy of the modules encourages reuse [Tracz:95].

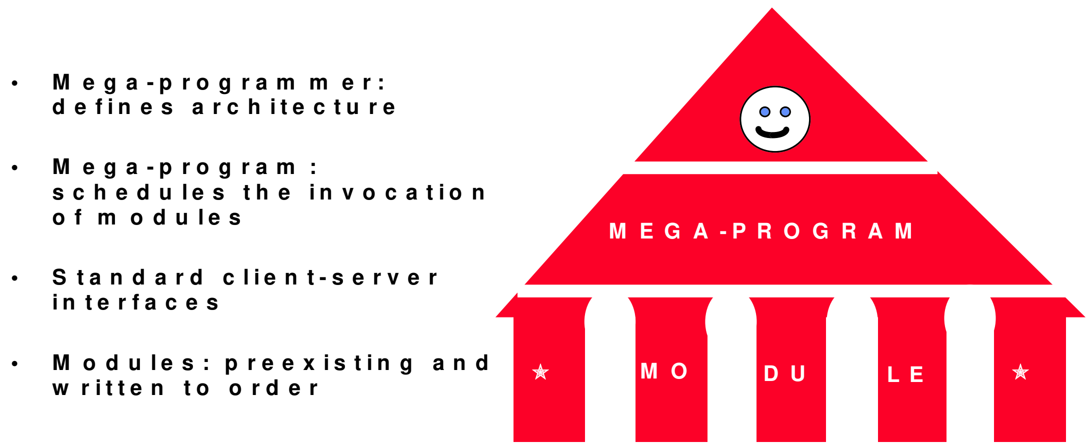

Figure 1: Components of the CHAIMS mega-programming paradigm.

The process of the CHAIMS mega-program generation and use is illustrated in Fig.2. Note that this proposal does not envisage automatic programming or knowledge-based techniques [Smith:90]. It simply provides a new high level tool for module composition. Neither does it provide novel interfaces. The result of a mega-program execution are simply shipped to an output module, written again in any language that is compatible with the end-user or customer platform. However, once a language-driven style of interoperation is established, some of the practical barriers which disable automation of large-scale computations will have disappeared.

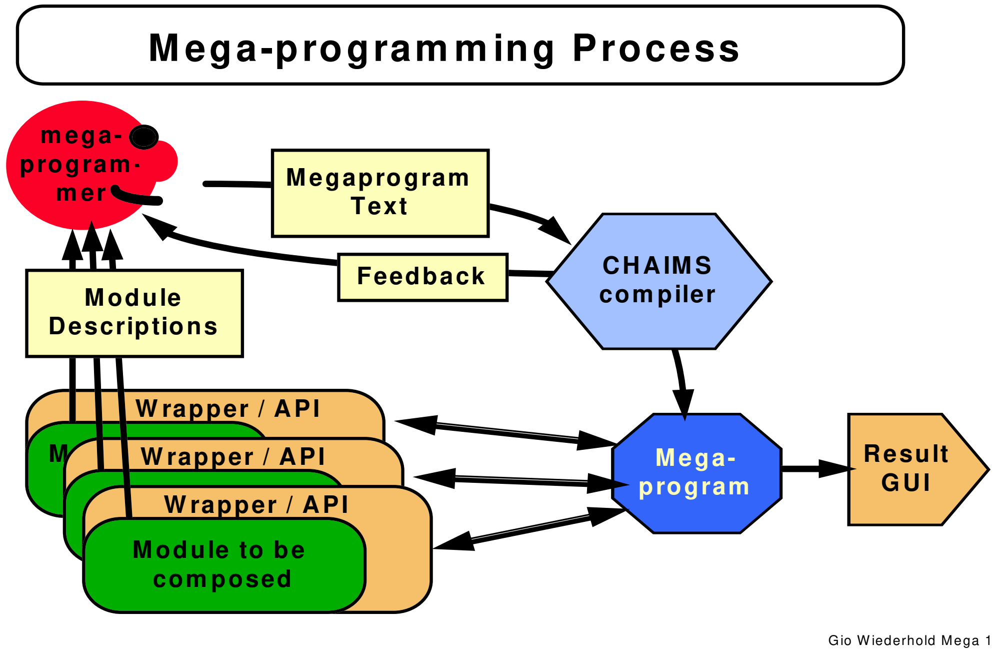

Figure 2: The CHAIMS Mega-programming Process.

The modules to be composed by CHAIMS can be written in any language that is able to create the required interfaces, or that generates code that can be suitably wrapped. We certainly will demonstrate modules in C and C++. The FORTRAN and Ada languages are high on our list as well. We intend to profit here from the many efforts in supporting interoperation interfaces currently in process, and likely to be accelerated by other proposals in the EDCS suite. Where required, glue-code to resolve semantic differences will be supplied by modules written in any suitable programming language that can generate a compatible interface. The CHAIMS compiler itself will not be burdened with having to generate such glue code, since here conventional languages perform already well.

The CHAIMS language will be designed to solely serve system integrators, an increasingly large fraction of the software community. CHAIMS will not obsolete specific languages to write modules, although its use will enable new module efficiencies due to greater flexibility in making platform assignments and specialization. Modules composed by a CHAIMS program can be written in the most suitable language and use the best representation for the given sub-task and solution approach.

.

H: Key personnel.

Gio Wiederhold is a Professor of Computer Science at Stanford University and has courtesy faculty appointments in Medicine and Electrical Engineering. He works on applying knowledge-based techniques to database management, information systems, security, and software construction and maintenance. His effort on this proposal will be supported by other funding.

Wiederhold has published 4 books and more than 250 publications in computing and medicine. He is a fellow of the ACMI, the IEEE, and the ACM, as well as a member of the AAAS, AAMI, AIAA, and AAAI.

At Stanford Gio Wiederhold is a member of the Computer Systems Laboratory, the Medical Information Science Program, and the Centers for Integrated Systems (Dep. of Elec. Eng.), Facility Engineering (Dep. of Civil Eng.) and the Stanford Computer Industry Project (SCIP) of its Business School. Past work on the application and development of knowledge-based techniques to database management, information systems, and software construction and maintenance, initiated with the KBMS project at Stanford in 1977, has now become an active research field in its own right. Results derived from this work are found in general system design, medicine, information systems, and planning applications. He has supervised over 25 PhD students. Complementary current research projects address Heterogeneity of Information and Trusted Interoperation of Healthcare Information. He currently teaches an experimental course on `Large-Scale Software Construction' (CS446).

From 1991 to 1994 he was the Program Manager for Knowledge-Based Systems at ARPA, initiating programs in Intelligent Integration of Information (I3) and in Persistent-Object Bases (POB). He also participated in establishing directions for a Software Foundations program, specifically in composition technology for large, multi-site systems.

Professor Wiederhold received a degree in aeronautical engineering in Holland in 1957 and shortly after came to the United States. After gaining 16 years of industrial experience, he returned to school and earned a PhD in 1976 in Medical Information Science from the University of California in San Francisco. He has been on the Stanford faculty since that time. He has consulted for many governmental and commercial enterprises, among them the United Nations Development Programme, the US Department of Health and Human Services, various US defense agencies, and, of course, with Silicon Valley innovators.

Professor Wiederhold's address is Department of Computer Science, Stanford University, Stanford, CA 94305-2140, or Gio@cs.stanford.edu.
His home page is at http://www-db.stanford.edu and includes a full vita.

K: Cost Breakdown.

	10/1/97-9/31/98
Salaries	
Gio Wiederhold, 0%	0
Student research assistant 50% AY only	16,759
Subtotal Salaries	16,759
Staff Benefits @ 26.2% FY97	4,391
Travel-domestic (East-coast, shared)	589
Subtotal, Direct Costs	**21,739**
Indirect Costs @ 61%	13,261
Total Budget	**35,000**