Proposal to
The Defense Advanced Research Projects Agency

BAA 00-23: Program Composition for Embedded Systems
Topic areas:
Aspect Suites for Embedded Systems
Program Representation and Analysis
Program Transformation System

Alchemy: Transmuting Raw Code into Robust Components

Technical contact: JAY LEPREAU

University of Utah
Department of Computer Science
50 Central Campus Drive Rm. 3190
Salt Lake City, UT 84112-9205
Phone: 801-581-4285, Fax: 801-585-3743
lepreau@cs.utah.edu

Administrative contact: LYNNE CHRONISTER

University of Utah
Office of Sponsored Projects
1471 Federal Way
Salt Lake City, UT 84102-1821
801-581-3003, Fax: 801-581-3007
ospawards@osp.utah.edu

Proposed duration: 36 months

January 23, 2000

A Innovative Claims

In the domain of low-level systems, the implementations of functional subsystems and cross-cutting aspects are inextricably mixed. Many parts of a system, such as the memory manager, the threads package, or the I/O buffer manager, exist in two dimensions: they are implemented by modules that should be locally replaceable with other implementations, but they also induce dependencies among components that span the entire system.

The goal of our work is to integrate the notions of component and aspect development into a single programming paradigm, resulting in a new model for component programming that takes into account low-level systems' requirements for assurance, performance, and use of legacy code, particularly with respect to cross-cutting dependencies such as concurrency, memory management, and execution time. More concretely, we will (1) build a new model for components, based on the state-of-the-art module language of *units* [28, 29], (2) extend the language of component interfaces to specify the interaction of cross-cutting aspects, such as concurrency and resource constraints, (3) develop a composition language for components that permits easy replacement and reuse of components while enforcing component dependencies, and (4) optimize across component boundaries to make the performance of componentized systems reasonable. The tools and techniques that we develop will be applicable to the huge body of existing legacy low-level systems code written in C, promising practical impact decades earlier than techniques that rely on advanced languages themselves.

We have a unique resource with which to begin our investigations: the OSKit [30, 34]. The OSKit is set of C-based components, many of them derived from existing operating systems such as Linux, that provide reusable building blocks for constructing low-level software, such as operating systems and embedded systems. It has proven useful to operating systems and language researchers worldwide. However, as both the OSKit and the demands on it have grown, we have seen serious limitations in the OSKit's ability to scale, due to the complexity of the components presented to the end programmer. We will attack these problems by applying our new component programming techniques to the OSKit.

Our research will yield both an approach to systems design and specific tools for realizing the designs, including a module language, ALCHEMY, and its compiler and component composition tool, ALCHEMIST. As we evaluate and demonstrate the applicability of our tools and techniques by applying them to the OSKit, our research will also produce an improved OSKit that is suitable for use in building embedded systems. In addition, by amortizing the cost of expensive verification techniques over its reusable components, such techniques become attractive to apply to the OSKit.

Because the OSKit component base is widely used by researchers, it provides our research with the real-world constraints of users and scale, as well as the benefit of their feedback. By using the OSKit as a basis, we will be guided toward component specification solutions that are practical and that can be integrated in an evolutionary manner into large, existing, production systems: not just in the OSKit, but in similar low-level and legacy systems as well.

We will test that claim by expanding our set of Linux-derived components and specifications to encompass the remaining major portions of the kernel. We expect the resulting Linux-compatible, aspect-aware, component base, together with our tools, to have broad impact.

B Proposal Roadmap

Main Goal of the Work **Pages 3–7**

Our goal is to develop tools and components for building high assurance and high performance low-level systems, through a language for system composition that integrates components—for local replacement and reuse, with aspects—for global reconfiguration of implementations.

Tangible Benefits to End Users **Pages 9, 17, 30**

We will produce a component-level language, ALCHEMY, its compiler and composition tool, ALCHEMIST, and a suite of low-level components for building operating systems and embedded systems. End users building new systems will benefit from access to new tools and reusable code.

Critical Technical Barriers **Pages 6–7**

The technical barriers are (1) developing a type and specification system that is adequate for verifying the composition of components and aspects, (2) breaking realistic/legacy systems implementations into appropriate components, and applying types and specifications to the components, and (3) optimizing across component boundaries to lower the cost of componentization.

Main Elements of the Proposed Research **Pages 9–16, 18–24**

The research consists in (1) developing our existing unit language for components to handle the cross-cutting dependencies found in low-level systems code, and (2) overcoming the semantic and performance barriers to decomposing low-level code into fine-grain components.

Confidence That Work Will Overcome Technical Barriers **Pages 7, 17**

Our investigations begin with both a theoretical asset and a practical one. The unit model of components supports an unprecedented combination of expressiveness and static analyzability. The OSKit components offer both control over numerous cross-cutting aspects and a realistic environment with actual users for developing the theory.

Nature of the Expected Results **Page 27**

We expect our research to culminate in both theoretical advances, in terms of the specification and verification of components at their boundaries, as well as concrete software advances, in terms of new components and tools for systems implementation.

Risk if the Work is not Done **Pages 5, 17**

If such work is not pursued, embedded and low-level systems will be more expensive and less robust for the next 15 years. Systems programmers will continue to struggle with *ad hoc* techniques for modularization, and to waste implementation effort building systems components that could have been reused from existing, validated implementations.

Criteria for Evaluating Progress and Capabilities **Pages 18–25**

We will evaluate ALCHEMY by testing it on the OSKit and Linux. The resulting components should be more flexible and easier to use, and systems constructed with the components should be smaller and perform better than with the original components.

C Technical Description

C.1 Technical Innovations

We propose to develop a new language and tools for componentizing systems software, with particular attention to the specification and verification of cross-cutting dependencies, or aspects, among components. Our strategy for the language, ALCHEMY, is to extend component interfaces with specifications, first along the lines of higher-order type systems, and later with constraints based on more advanced proof systems. At the same time, we will lower the performance cost of componentization through a component compiler, ALCHEMIST, that optimizes across component boundaries (which can be viewed as "weaving" the implementation of widely-used components into other components). To further encourage the production of small, highly-parameterized components, ALCHEMY will allow linking components together hierarchically, with support for the overriding of a component by another with a compatible interface.

Our previous work on *units* [28,29] provides the theoretical underpinning for ALCHEMY. The most important facet of the unit model (which has been implemented for Scheme) is that each unit specifies only the shape of its imports, rather than the source. The source of a unit's imports is determined later, by the programmer who links units together to create a program. This feature of units is crucial to ensuring that components are reusable and replaceable. Further, it provides a framework for exploring component specifications that go beyond simple type signatures, and for exploring component composition tools that support the implementation of aspects.

The OSKit [30, 34], developed by our group, will serve both as the primary test case for ALCHEMY, and as a deliverable set of components for writing embedded systems. The OSKit is an existing set of C-based components for constructing low-level software. Aspects of operating system implementation, such as virtual memory management, threading, locking, buffering, and device handling, are implemented by separate components in the OSKit. Many of the larger components are encapsulated "legacy" software derived with little or no modification from the parts of other operating systems such as Linux [68] and FreeBSD [64]. The OSKit currently runs on the Intel x86 and StrongARM hardware platforms. Execution environment aspects are sufficiently decoupled that with the addition of a Unix adaptor component, almost all OSKit kernels can run on Linux or FreeBSD instead of on the bare hardware.

Configuring combinations of OSKit components is certainly easier than writing new code from scratch, even in the absence of clear specifications or automated checking, but it is still complex and error-prone. The OSKit demonstrates that current component technologies are not well-suited to low-level systems, because current solutions (1) fail to support sophisticated component dependencies and requirements, and (2) frequently introduce run-time overheads that are unacceptable in low-level systems. Our research will focus on these critical issues. We expect that ALCHEMY will make the OSKit more reliable, easier to use, and more efficient, both for current users and for future projects as we expand the set of OSKit components. Beyond the OSKit, we will apply, develop, and demonstrate our new component model in the reengineering of other "legacy" systems software; in particular, a coarse decomposition of the Linux kernel.