

My current and future research interests involve facilitating software development for large-scale heterogeneous distributed computing systems. Technological "raw materials"—including powerful inexpensive computers, high-speed networks, and high-performance network protocols—are rapidly growing in power and profusion, and applications that require these resources abound. However, the right abstractions and environments that would avail applications developers of the resources have, until recently, been missing. Developing and enhancing "middleware" or "metasystem" environments is an exciting area of research with an abundance of new and interesting problems.

As a graduate student in the Legion research group at the University of Virginia, I have contributed to all aspects of an extremely successful experimental metasystems research project. When I arrived at Virginia, I was fortunate to join a relatively small group of four graduate students and three faculty, led by Andrew Grimshaw. The group was just beginning to work on an ambitious and exciting idea—creating software that would tie together vast amounts of geographically distributed heterogeneous computing resources, taking advantage of emerging gigabit wide area networks to provide a single worldwide virtual computer. In my first year, I helped design the system and I wrote the first detailed specification of Legion's components and their roles, and the first published paper that describes Legion. I also helped write two successful funding proposals, resulting in grants from DARPA and DOE, and another proposal that is still being considered by NSF.

In early 1996, we began to implement the Legion software. For the first year and a half of development I was among the two or three primary system architects. In particular, I implemented the Legion naming mechanism, the original class system, and many of the underlying system components. The Legion software, which includes much of my code at its core, runs successfully on many different platforms, and is currently deployed at universities, research laboratories, and government agencies across the United States and on other sites around the world, including in the Netherlands and Japan.

Whereas my contributions to Legion have been made while working within a talented and ambitious research group, my dissertation, titled *Dynamically Configurable Distributed Objects*, represents my own independent research. Dynamic configurability enables efficient and effective distributed object evolution using component-based software composition in the context of object-based distributed computing systems. I have developed a language-independent model that describes how shared, persistent, independent, active, distributed objects can change their interfaces and implementations on the fly, without deactivating any part of the system, including the objects being evolved. I have implemented the model within Legion, I have studied the impact that dynamic configurability can have on other parts of the system, and I have created evolution management strategies that define and restrict when and how objects can evolve.

My dissertation represents the right approach to the problem of enabling distributed object evolution, and my implementation of dynamic configurability within Legion is a useful artifact of the research. My contributions have opened up many exciting possibilities for future work in the area. In particular, I will address the following research questions related to my dissertation:

1. *How should programmers be able to specify evolution?* My dissertation provides mechanisms and approaches for on-the-fly evolution, but does not concentrate on how users initiate and invoke the mechanisms, which will be key to the success of dynamic configurability. Specifying evolution rules within makefiles and other project development environments will help. More complex approaches include language extensions that will allow programmers to

manipulate and change type definitions at run-time, and that will have those changes be reflected, via support from a specialized compiler, in the classes and objects in the underlying run-time system.

2. *How should programmers build evolvable objects?* My current implementation of dynamic configurability allows programmers to build implementation components by structuring their source code appropriately, by including code necessary for dynamic configurability, and by specializing a boiler-plate compilation mechanism. To become useful for a wide range of programmers and environments, the mechanism for building components must be more flexible and automatic. Pragmas or compiler directives will allow programmers to delineate components and will allow preprocessors or compilers to generate much of the necessary code automatically. Other strategies involve elevating the notion of an implementation component to a first-class entity in a programming language, as well as in the underlying run-time system.
3. *What are the right evolution management strategies for real applications, according to real programmers?* My dissertation research provides a set of evolution management strategies, but none have been tested on real applications with real users. Significant long-term user studies will help refine and extend the set of policies I have defined.

I will also work on extending the kinds of objects that can evolve dynamically. My current dynamic configurability model applies to objects with their own address spaces, one per object. Different mechanisms, algorithms, and policies may be necessary to support objects that are distributed across multiple address spaces, that share address spaces with one another, and that are replicated. I will also investigate approaches for evolving multiple different object types in tandem, since applications generally comprise several different object types that work in concert with one another, and that often need to evolve together.

In a related area, I plan to explore supporting mobile agents in heterogeneous multi-language environments. The key to doing so will be to define the representations of agents in a self-describing standard low-level form so that they can be incorporated into objects written in different languages. An agent should contain a description of (at least) its semantics and its implementation characteristics. This will allow different entities to host agents written in different languages with different syntax and semantics. I believe that the format and approaches I have used for implementation components in my thesis can be extended to work well for language-independent mobile agent systems.

In the longer term, I plan to continue to merge into a cohesive environment, the programming, compilation, and run-time aspects of distributed software development for heterogeneous high-performance computing environments. I will bring entities that typically exist in only one of the aspects, and give them roles in one or more of the others. For example, types, modules, and classes are traditionally programming language constructs that are used to build executable code, and that generally play a limited role at run-time. In my opinion, Legion's key enabling idea was to turn the class into an active entity that exists at run-time, and to empower it with system-level responsibility to manage its instances as it sees fit. Likewise, executables are active entities in the Legion run-time system, as are dynamic configurability's implementation components. As such, they can be manipulated like any other objects in the system, thereby enabling automatic binary management across multiple disjoint file systems, and distributed object evolution, respectively. I believe that other benefits can be reaped by exploring further in this direction.