The Legion Vision of a Worldwide Virtual Computer

Long a vision of science fiction writers and distributed systems researchers, the notion of a worldwide computer, now taking shape through the Legion project, distributes computation like the World-Wide Web distributes multimedia, creating the illusion for users of a very, very powerful desktop computer.

Andrew S. Grimshaw, Wm. A. Wulf, and the Legion team

TODAY'S DRAMATIC INCREASE IN AVAILABLE NETWORK BANDWIDTH WILL qualitatively change how the world computes, communicates, and collaborates. The rapid expansion of the World-Wide Web and the changes it has wrought are just the beginning. As high-bandwidth connections become available, they shrink distances and change our modes of computation, storage, and interaction. Inevitably, users will operate in a wide-area environment transparently consisting of workstations, PCs, graphics-rendering engines, supercomputers, and nontraditional computing devices, such as televisions. The relative physical locations of users and their resources is increasingly irrelevant.

Realization of such an environment, sometimes called a "metasystem," is not without problems. Today's experimental high-speed networks, such as the Very

MARTIN MAYO

High-speed Backbone Network Service (vBNS) connecting the NSF supercomputer centers and other sites and the I-Way, a one-time networking event, are a preview of both the promise and pitfalls of constructing a metasystem. There are many difficulties:

- Few approaches scale to millions of machines.
- The tools for writing applications are primitive.
- Faults abound and mechanisms to handle them are not available.
- Issues of security are treated in a patchwork manner.
- Site autonomy—the ability to control one's own resources while playing in the global infrastructure—is not addressed.

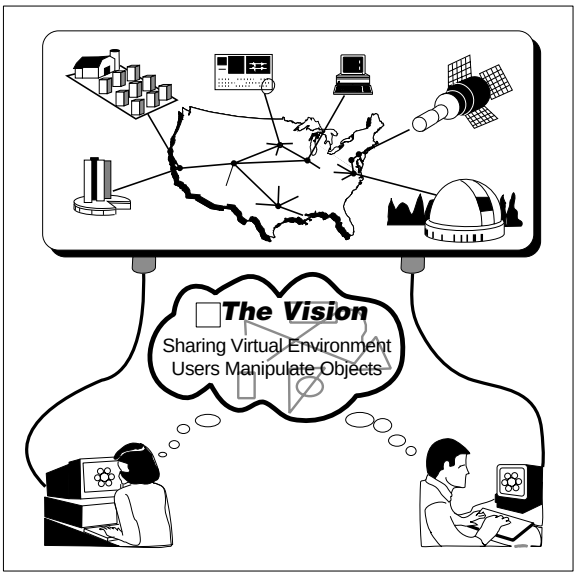

Figure 1. Legion vision

Software is the fundamental difficulty, arising from an inadequate conceptual model. Confronted by a flood of new hardware, the computer science community has tried to stretch an existing paradigm—interacting autonomous hosts—into a regime for which it was not designed. The result is a collection of partial solutions, some quite good in isolation but generally lacking coherence and scalability, making development of even a single wide-area application demanding at best.

Thus, the challenge to computer scientists is to provide a solid, integrated conceptual foundation on which to build applications that unleash the potential of so many diverse resources. The foundation must do the following:

- Hide the underlying physical infrastructure from users and from the vast majority of programmers.
- Support access, location, fault transparency, and construction of larger integrated components using existing components.
- Enable interoperability of components.
- Provide a secure environment for resource owners and users.
- Scale to millions of autonomous hosts.

The technology to meet these criteria largely exists in the form of:

- Parallel compilers that support execution on distributed memory machines.
- Distributed systems software that manages complex distributed environments.
- General acceptance of the object-oriented paradigm because of its encapsulation and reuse properties.
- Cryptography and cryptographic protocols.

Legion is a metasystem software project at the University of Virginia. Begun in 1993, its goal is a highly usable, efficient, and scalable system based on solid computing principles. We have been guided by our own work in object-oriented parallel processing, distributed computing, and security, as well as by decades of research in distributed computing systems. When complete, Legion will provide a single, coherent virtual machine that addresses such issues as scalability, programming ease, fault tolerance, security, and site autonomy. Legion is a conceptual base for the sort of metasystem we seek.

Our vision of Legion is a system consisting of millions of hosts and billions of objects co-existing in a loose confederation united through high-speed links (see Figure 1). Users will have the illusion of a very powerful desktop computer through which they can manipulate objects. When we refer to a *terminal*, we use the term in its most liberal sense, so terminal could mean anything from a workstation to an immersive environment, such as a head-mounted display, a Cave Automatic Virtual Environment (CAVE), or a portable personal digital assistant. The objects being manipulated will represent data resources (such as digital libraries and video streams), applications (such as teleconferencing and physical simulations), and physical devices (such as cameras, telescopes, and linear accelerators). Naturally, these objects may be shared with other users, allowing construction of shared virtual workspaces.

Legion is responsible for supporting the abstraction presented to the user, transparently scheduling application components on processors; managing data migration, caching, transfer, and coercion; detecting and managing faults; and ensuring that the users' data and physical resources are adequately protected.

Design Objectives

Realizing this vision is a daunting task. We have distilled 10 design objectives central to the success of the project:

- **Site autonomy.** Not a monolithic system, Legion will be composed of resources owned and controlled by an array of organizations. These organizations, quite properly, will insist on having control over their own resources (e.g., specifying how much of a resource can be used, when it can be used, and who can and cannot use it).
- **Extensible core.** Because we cannot know the future or all of the many and varied needs of users, mechanism and policy must be realized via extensible, replaceable components. An extensible core will permit Legion to evolve over time and allow users to construct their own mechanisms and policies to meet specific needs.
- **Scalable architecture.** Because Legion will consist of millions of hosts, it must be a scalable architecture. There can be no centralized structures; the system must be totally distributed.
- **Easy-to-use, seamless computational environment.** Legion must mask the complexity of the hardware environment and of the communication and synchronization of parallel processing (e.g., machine boundaries should be invisible to users). As much as possible, compilers, acting in concert with run-time facilities, must manage the environment for the user.
- **High performance via parallelism.** Legion must support easy-to-use parallel processing with large degrees of parallelism, including task and data parallelism and their arbitrary combinations.
- **Single, persistent name space.** One of the most significant obstacles to wide-area parallel processing is the lack of a single name space for file and data access. The existing multitude of disjoint name spaces makes it extremely difficult to write applications spanning multiple sites.
- **Security for users and resource owners.** Because we cannot replace existing host operating systems, we cannot significantly strengthen existing operating system protection and security mechanisms. However, we must ensure that Legion does not weaken existing mechanisms. Further, we must provide mechanisms for users to manage their own security needs; Legion should not define the security policy or require a "trusted" Legion.
- **Management and exploitation of resource heterogeneity.** Clearly, Legion must support interoperability between heterogeneous hardware and software components. In addition, some architectures are better than others at executing particular applications (e.g., vectorizable codes). These affinities and the costs of exploiting them must be factored into scheduling decisions and policies.
- **Multiple language support and interoperability.** Legion applications will be written in a variety of languages. It must be possible to integrate heterogeneous source language application components in much the same manner as heterogeneous architectures. Interoperability also means we must be able to support legacy codes.
- **Fault tolerance.** In a system as large as Legion, it is certain that at any given instant, several hosts, communication links, and disks will have failed. Thus, dealing with failure and dynamic reconfiguration is a necessity for Legion itself and for its applications.

Users will have the **illusion** of a very powerful desktop computer through which they can manipulate objects.

In addition to these goals, our design must deal with several constraints, including:

- **It can't replace host operating systems.** Organizations will not permit their machines to be used if their operating systems must be replaced. Operating system replacement would require them to rewrite many of their applications and retrain many of their users, and would possibly make these applications incompatible with other systems in their organizations. Our experience with Mentat [6], an earlier system developed at the University of Virginia, indicates that it is sufficient to layer a system on top of an existing host operating system.
- **It can't legislate changes to the interconnection network.** We must initially assume that the network resources and protocols in use are a given. Much as we must accommodate operating system heterogeneity, we must live with available network resources. However, we can layer better protocols over existing ones, and we can state that performance for a particular application on a particular network will be poor unless the protocol is changed.
- **It can't require Legion to run as "root" (or equivalent).** Indeed, to protect themselves, most Legion users will want Legion to run with the least possible privilege.

Legion's Object Foundation

The common framework enabling a coherent solution to these problems is object orientation. In Legion, all compo-

Problem	Tools Available
Writing parallel applications	CWVC-aware PVM, parallel C++, Fortran wrappers
Multiple separate file systems	Federated file system for transparent file access
Heterogeneous resources	Automatic scheduling, binary selection and migration, application specific scheduling tools
Multiple resource owners	Owner control of resource consumption, detailed resource consumption accounting
Debugging parallel programs difficult	Post-mortem playback using off-the-shelf debuggers, (e.g., dbx)
Host/network failures	Automatic system reconfiguration and limited application fault tolerance

Table 1. Campus Wide Virtual Computer toolset

nents of interest to the system are objects and all objects are instances of defined classes. Thus, users, data, applications, and even class definitions are objects. Use of an object-oriented foundation, including the object-oriented paradigm's encapsulation and inheritance properties, makes accessible a variety of benefits often associated with the paradigm, including software reuse, fault containment, and complexity reduction. The need for the paradigm is particularly acute in a system as large and complex as Legion.

Objects, written in either an object-oriented language or such sequential and parallel languages as C, Fortran, and High-Performance Fortran (HPF), will encapsulate their implementation, data structures, and parallelism and will interact with other objects via well-defined interfaces. In addition, they may also have associated inherited timing, fault, persistence, priority, and protection characteristics. Naturally, these characteristics may be replaced to provide different functionality depending on class. Similarly, a class may have multiple implementations with the same interface.

While we are committed to the object-oriented paradigm, we recognize that Legion needs to support applications written in a variety of languages in order to support existing legacy codes, permit organizations to use familiar languages (e.g., Ada, C, and Fortran), and support a variety of parallel processing languages and tools. We intend to provide multilanguage support and interoperability among user objects written in different languages in three ways:

- By generating object "wrappers" for code written in such languages as Ada, C, and Fortran.
- By exporting the Legion run-time system interface and retargeting existing compilers and tools.
- By using a combination of the first two ways.

System Philosophy

Complementing our use of the object-oriented paradigm is a philosophical theme: We cannot design a system that satisfies every user's needs. We must design Legion to allow users and class implementers the greatest flexibility in the semantics of their applications. Therefore, we resist the temptation to provide "*the* solution" to a wide range of system functions. Users should be able, whenever possible, to select both kind and level of functionality, making their own trade-offs between function and cost (see Figure 2).

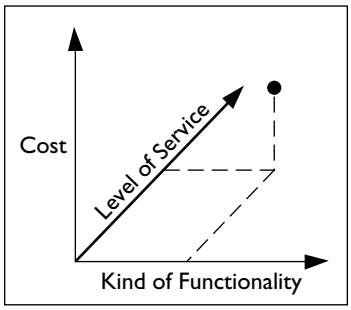

Figure 2. Calculating trade-offs between Legion functions and their system and financial costs

Neither the kind nor the level of functionality is linearly ordered; a simplistic model is a multidimensional space. The needs of users will dictate where users need to be or can afford to be in this space; we—the designers of the supporting conceptual system—have no way of knowing what those needs are or what they will evolve into in the future. Indeed, if we were to dictate a systemwide solution to almost any of the issues on our list of objectives, we would preclude large classes of potential users and uses.

For example, consider security with respect to both kind and level of functionality. Some users are mostly concerned with privacy; others, especially banks and hospitals, with the integrity of their data. Some users are content with password authentication, others might feel the need for stronger user identification through signature analysis, fingerprint verification, or other techniques. These examples illustrate the differences in security functionality. The size of the cryptographic key, on the other hand, is an issue of the level of security. Without changing the basic nature of the security provided, users can get a greater degree of security by paying the higher cost of using a longer key or a stronger algorithm.

The Legion approach, rather than providing a fixed security mechanism, with the result that no one is completely satisfied, lets users choose their own trade-offs by

implementing their own policies or by using existing policies via inheritance. Some users may want a policy requiring that every method invocation has all of its parameters encrypted, that the caller be separately authenticated, and that the user on whose behalf the call is being made be fully authenticated as well. Such a policy is expensive (in CPU power, network bandwidth, and time). Alternatively, an application requiring low overhead cannot afford such a policy and should not be forced to use it. Such an application could instead choose a lightweight policy that simply checks whether the caller is its parent (creator) without any authentication or encryption or perhaps does not check anything at all.

Next, consider consistency semantics in a distributed file system. To achieve good performance, it is often desirable to make copies of all or portions of a file. If updates to the file are permitted, the different copies may begin to diverge. There are many ways to attack this problem, including not permitting replication of writable files, using a cache-invalidate protocol, using lazy updates to a master copy, and so on. Each has an associated cost and semantics. Some applications don't require all copies to be the same; others require strict "reads deliver the last value written" semantics, and others know that the file is read-only so that consistency protocols are a waste of time; still others may need different semantics for the file in different regions of the application. Independent of the file semantics, some users may need frequent automatic backups and archiving, others may not. The point is that the system should not make such decisions for users, who should be able to select the kind and level of service they require.

This philosophy has been extended into the system itself. The Legion object model specifies the composition and functionality of Legion's core objects—those objects cooperating to create, locate, manage, and remove other objects from the Legion system. Legion specifies the functionality (not the implementation) of the system's core objects. Therefore, Legion's core will consist of extensible, replaceable components. The Legion project will provide implementations of the objects the core comprises, but users will not be obligated to use them. Instead, Legion users will be encouraged to select or construct objects that implement mechanisms and policies meeting the users' specific requirements.

The object model provides a natural way to achieve such flexibility. Files, for example, are not part of Legion itself. Any programmer or developer may define a new class whose general semantics we would recognize as those of a file but whose specifics match the particular semantics meeting the user's needs. We (the Legion team) need to provide an initial collection of file classes reflecting the most common needs, but we do not have to anticipate all possible future requirements.

Beyond Prototypes

In summer 1995, we released our first prototype Legion implementation—the Campus Wide Virtual Computer (CWVC)—based on the earlier object-oriented parallel processing system Mentat [6]. Mentat was originally designed to operate in homogenous, dedicated environments but has been extended to operate in an environment with heterogeneous hosts, disjoint file systems, local resource autonomy, and host failure. We could have continued to stretch Mentat but felt that one can transform a system only so far before it begins to show signs of the stress; it is often better to design from the ground up so the resulting system has a clean, coherent architecture rather than a patchwork of modifications based on a solution for a different problem.

The CWVC is a direct extension of Mentat onto a larger scale, is a prototype for the nationwide Legion system, and reflects the idea that a university is a microcosm of the world. The computational resources at a university are operated by many different departments; there is no shared name space, and sharing of resources is currently rare.

Even though the CWVC is much smaller and the components much closer together than in the envisioned worldwide Legion, it still presents many of the same challenges. The processors are heterogeneous; the interconnection network is irregular, with orders-of-magnitude differences in bandwidth and latency; and the machines are currently in use for on-site applications that cannot be incapacitated in any way. Further, each department at a university operates essentially as an island of service, with its own network file system mount structure and trusting only machines on the island.

The CWVC, which is both a prototype and a demonstration project, aims to accomplish several goals:

- Demonstrate the usefulness of network-based, heterogeneous, parallel processing to university computational science problems;
- Provide a shared high-performance resource for university researchers;
- Provide a given level of service (as measured by turnaround time) at reduced cost; and
- Act as a testbed for the nationwide Legion.

The CWVC consists of more than 100 workstations and an 18-processor IBM SP2 parallel-processing supercomputer in six buildings using two completely disjoint underlying file systems. We have developed a suite of tools to address a number of common problems (see Table 1). In collaboration with domain scientists at the University of Virginia and elsewhere, we also developed a set of applications that exploit the CWVC (see Table 2).

In addition to the local production environment, we have

demonstrated the CWVC on wide-area systems. For example, during the Supercomputing'95 conference in San Diego, we ran the CWVC on the I-Way. The connections were at DS-3 (45MB/sec) and OC-3 (155MB/sec) rates. The CWVC was installed at three sites using seven hosts on three different architectures. At the National Center for Supercomputer Applications (NCSA) in Urbana, Ill., we used four Silicon Graphics Power Challenge shared-memory multiprocessors (8 to 16 processors each) and a Convex Computer Exemplar shared-memory multiprocessor. At the Cornell Theory Center (CTC) in Ithaca, N.Y., and at Argonne National Laboratory (ANL) in Argonne, Ill., we used IBM SP2s.

Once the IP routing tables were properly configured, moving the CWVC to the wide-area environment was relatively simple. We copied the CWVC to the platforms, adjusted the tables to use IP names that routed through the high-speed network, and tested the system. As expected, files could be accessed in a location-transparent fashion, executables were transparently copied from one location to another as needed, the scheduler worked, and the system automatically reconfigured on host failure. Utilities and tools, such as the debugger, also migrated easily. The real bonus was that user applications required no changes to run in the new environment.

Discipline	Application
Biology	DNA and protein sequence comparison
Computer Science	Parallel databases and I/O, genetic algorithms
Electrical Engineering	Automatic test pattern generation, VLSI routing,
Engineering Physics	Trajectory and range of ions in matter
Physics	2D electromagnetic finite element mesh

Table 2. Sample Campus Wide Virtual Computer applications

For our demonstration, we used our utilities and ran one of our applications, called complib, on the I-Way. Complib compares two DNA or protein sequence databases using one of several selectable algorithms [5]. The first database was located at ANL, the second at NCSA. The application transparently accessed the databases using the Legion file system, while the underlying system schedulers placed application computation objects throughout the three-site system (third site at CTC). Legion transparently handled all communication, placement, synchronization, code, and data migration.

Since Supercomputing'95, we have repeated the demonstration several times and are now constructing a more permanent prototype to span NCSA and the San Diego Supercomputer Center (SDSC) and will operate it as a part of the DARPA-funded Distributed Object Computation Testbed (see http://www.sdsc.edu/doct/).

Related Work

The vision of a seamless metacomputer is not novel; worldwide computers have been the vision of science fiction writers and distributed systems researchers for decades. However, to our knowledge, no other project has a scope and goals as broad and ambitious as Legion. Fortunately, it is not necessary to develop all of the required technology from scratch. A large body of relevant research in distributed systems, parallel computing, fault tolerance, management of groups of workstations, and pioneering wide-area parallel processing projects provide a strong foundation on which to build.

Related efforts, such as OSF DCE [7] and CORBA [2], are rapidly becoming industry standards. Legion and DCE share many of the same objectives, drawing on the same heterogeneous distributed computing literature for inspiration. Consequently, both projects use many of the same techniques (e.g., an object-based architecture and model, interface description languages [IDLs] to describe object behavior, and wrappers to support legacy code). However, Legion and DCE differ in several fundamental ways. First, DCE does not target high-performance computing; its underlying computation model is based on blocking remote procedure calls between objects. Further, DCE does not support parallel computing, instead emphasizing client/server-based distributed computing. Legion, on the other hand, is based on a parallel computing model, and one of our primary objectives is high performance via parallel computation. Another important difference is that Legion specifies very little about implementation. Users and resource owners are permitted—even encouraged—to provide their own implementations of "system" services. Our core model is completely extensible, providing choice at every opportunity—from security to scheduling to fault tolerance. Similarly, CORBA [2] defines an object-oriented model for accessing distributed objects. CORBA includes an IDL and a specification for the functionality of run-time systems that enable access to objects. But like DCE, CORBA is based on a client/server model rather than a parallel computing model and places less emphasis on such issues as object persistence, placement, and migration.

Other projects share many of the objectives but not the

scope of Legion. For example, Nexus [4] provides communication and resource management facilities for parallel-language compilers; unlike Legion, Nexus does not address fault tolerance, security, site autonomy, or extensibility. Castle [3] is a set of related projects that aims to support scientific applications, parallel languages and libraries, and low-level communication issues; unlike Legion, Castle does not address heterogeneity, persistence, fault tolerance, security, and autonomy. The Network of Workstations (NOW) project [1] provides a somewhat more unified strategy for managing networks of workstations but is intended to scale only to hundreds of machines, not millions. And Globe [9] is an architecture for supporting wide-area distributed systems but does not yet seem to address such important issues as security and site autonomy.

In its intended application for distributed collaboration and information systems, Legion might be compared to the Web. In particular, the object-oriented, secure, platform-independent remote execution model afforded by the Java language [8] has added more Legion-like capabilities to the Web. The most significant differences between Java and Legion are Java's lack of a remote method invocation facility, lack of support for distributed memory parallelism, and interpreted nature, which even in the presence of just-in-time compilation leads to significantly lower performance than can be achieved using compilation. Furthermore, the security and object placement models provided by Java are rigid and a poor fit for many applications.

The Future

Legion is an ambitious middleware project that will provide a solid, integrated conceptual foundation on which to build applications. One could argue that Legion is perhaps too ambitious, that there are just too many different complex issues to address. The number of different issues is certainly a risk. On the other hand, Legion-like metasystem software will be developed eventually; such software is a necessary condition for a large-scale digital society. The real issue is whether it will come about by design in an organized and coherent fashion or by pasting together various solutions. Legion's strength is that its object model was designed from project inception both for the intended environment and for extensibility. We feel these attributes will permit Legion to readily adapt to an ever-changing worldwide computing infrastructure.

Legion—as defined by our objectives—is not yet a reality. While we have a prototype, its purpose is to demonstrate the feasibility of constructing a wide-area system and to permit application and tool development to occur concurrently with system implementation. The prototype is not designed to evolve directly into a complete Legion implementation. For more information, see http://www.cs.virginia.edu/legion.

In March 1996, we began implementation of the core Legion object model. Unlike the existing prototype, this implementation incorporates mechanisms for security, fault tolerance, application-directed scheduling, autonomy, scalable binding, and more. The complete implementation will reuse many of the prototype's components, including the compiler and debuggers, although for the most part, it is being written from the ground up. We expect to have a usable, documented system available for public use in mid-1997. The system and sources will be publicly available.

Acknowledgements

The Legion team at the University of Virginia includes Steve Chapin, James C. French, Paul F. Reynolds, Jr., Charlie Viles, and Alfred C. Weaver, as well as research scientist Mark Hyett and graduate students Adam Ferrari, John Karpovich, Darrell Kienzle, Mike Lewis, Anh Nguyen-Tuong, and Chenxi Wang.

References

1. Anderson, T.E., Culler, D.E., Patterson, D.A., and the NOW team. A case for NOW (Networks of Workstations). IEEE Micro. 15, 1 (Feb. 1995) 54–64.
2. Ben-Naten, R. *CORBA: A Guide to the Common Object Request Broker Architecture*. McGraw-Hill, New York, 1995.
3. The Castle Project. University of California, Berkeley (see http://http.cs.berkeley.edu/projects/parallel/castle/castle.html).
4. Foster, I., Kesselman, C., and Tuecke, S. Nexus: Runtime Support for Task-Parallel Programming Languages. Argonne National Laboratories (see http://www.mcs.anl.gov/nexus/paper/).
5. Grimshaw, A.S., West, E.A., and Pearson, W.R. No pain and gain! Experiences with Mentat on biological applications. *Concurrency: Prac. & Expe. 5*, 4 (June 1993), 309–328.
6. Grimshaw, A.S., Ferrari, A.J., and West, E.A. Mentat in *Parallel Programming Using C++*. G. Wilson, Ed. MIT Press, Cambridge, Mass., 1996, pp. 383–427.
7. Lockhart, H.W., Jr. *OSF DCE Guide to Developing Distributed Applications*. McGraw-Hill, New York, 1994.
8. Sun Microsystems. *The Java Language Specification, Version 1.0*, Beta, Oct. 30, 1995.
9. Van Steen, M., Homburg, P., Van Doorn, L., Tanenbaum, A.S., and deJonge, W. Towards object-based wide area distributed systems. In *Proceedings of the International Workshop on Object Orientation in Operating Systems*, L.F. Carbrera and M. Theimer, Eds. (Lund, Sweden, Aug. 1995).

Andrew Grimshaw *(grimshaw@virginia.edu) is an associate professor of computer science and director of the Institute for Parallel Computation at the University of Virginia.*

Wm. A. Wulf *(wulf@mail.cs.virginia) is the AT&T Professor of Computer Science at the University of Virginia. He is on leave for the 1996-97 academic year serving as President of the National Academy of Engineering*

This work is partly supported by DOE grant DE-FG02-96ER25290, DOE contract Sandia #LD-9391, and NSF grant ASC-9201822.