Legion

The next logical step toward the world-wide virtual computer

Andrew S. Grimshaw, Wm. A. Wulf

and the whole Legion team

1 The Opportunity

The dramatic increase in ubiquitously available network bandwidth will qualitatively change how the world computes, communicates, and collaborates. The rapid expansion of the world-wide web, and the changes that it has wrought are just the beginning. As high bandwidth connections become available they "shrink" distance" and change our modes of computation, storage and interaction. Inevitably, users will operate in a wide-area environment that transparently consists of workstations, personal computers, graphics rendering engines, supercomputers, and non-traditional computing and rendering devices: TVs, toasters, etc. The relative physical location of the users and their resources will become increasingly irrelevant.

The realization of such an environment, sometimes called a "metasystem", is not without problems. Today's experimental high speed networks such the vBNS and the I-way preview both the promise and pitfalls. 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, and site autonomy — controlling ones own resources while still playing in the global infrastructure —þis not addressed.

As usual, the fundamental difficulty is software — specifically, we believe the problem is an inadequate *conceptual model*. In the face of the onrush of hardware, the 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 lacking coherence and scalability — that make the development of even a single wide-area application demanding at best.

Thus, the challenge to the computer science community 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 at least hide the underlying physical infrastructure from users and from the vast majority of programmers, support access, location, and fault transparency, enable inter-operability of components, support construction of larger integrated components using existing components, provide a secure environment for both resource owners and users, and it must scale to millions of autonomous hosts.

The technology to meet this challenge largely exists: (1) parallel compilers that support execution on distributed memory machines, (2) advances in distributed systems software that manage complex distributed environments, (3) the widespread acceptance of the object-oriented paradigm because of its encapsulation and reuse properties, and (4) advances in cryptography and cryptographic protocols.

Legion is a metasystems software project at the University of Virginia[1]. Begun in the Fall of 1993, our goal is a highly usable, efficient and scalable system based on solid 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 each of the issues raised earlier: scalability, programming ease,

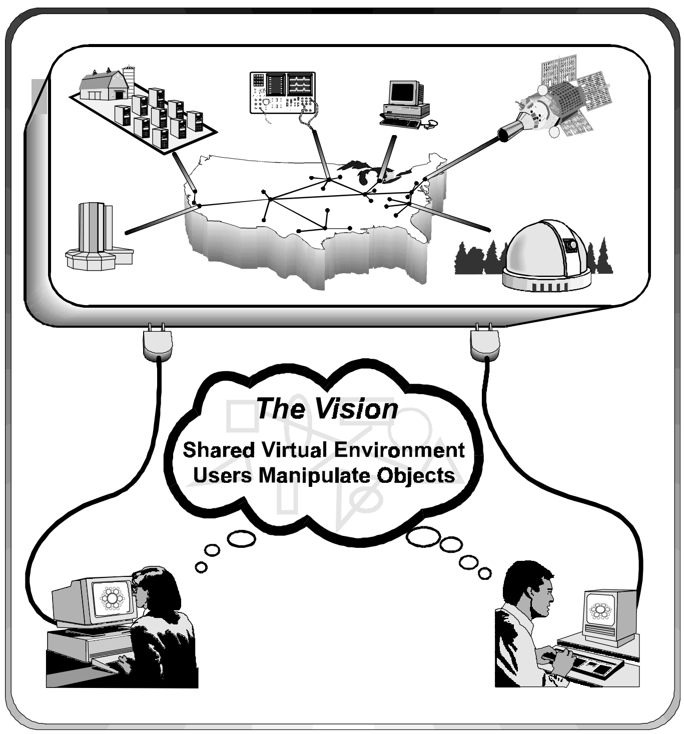
---

1. We can provide only an introduction to Legion in this short paper. For more information, including all referenced technical reports, see the Legion home page at http://www.cs.virginia.edu/~legion/.

fault tolerance, security, site autonomy, etc. In short, we believe Legion is a conceptual base for the sort of metsystem we seek.

Our vision of Legion is of a system consisting of millions of hosts and billions of objects co-existing in a loose confederation tied together with high-speed links. The user will have the illusion of a very powerful computer on her desk. She will sit at her terminal and manipulate objects. We use terminal in its most liberal sense. Terminal could mean anything, a workstation, an immersive environment such as a head-mounted display or CAVE^{TM}, or a portable Personal Digital Assistant. The objects she manipulates will represent data resources such as digital libraries or video streams, applications such as teleconferencing or physical simulations, and physical devices such as cameras, telescopes, or linear accelerators. Naturally the objects being manipulated may be shared with other users – allowing the construction of shared virtual workspaces.

It is Legion's responsibility to support the abstraction presented to the user, to transparently schedule application components on processors, manage data migration, caching, transfer, and coercion, detect and manage faults, and ensure that the users data and physical resources are adequately protected.



2 Project Objectives

To realize our vision is a daunting task. We have distilled ten design objectives that are central to the success of the project: site autonomy; an extensible core; scalability; an easy-to-use, seamless computational environment; high performance via parallelism; a single persistent name space; security for both users and resource providers; management and exploitation of resource heterogeneity; multi-language support and inter-operability; and fault tolerance.

- *Site autonomy*: Legion will not be a monolithic system. It 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 resource can be used, when it can be used, and who can and cannot use the resource.

- *Extensible core*: We cannot know the future or all of the many and varied needs of users. Therefore, mechanism and policy must be realized via extensible, replaceable, components. This will permit Legion to evolve over time and will 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 must 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 communication and synchronization of parallel processing — for example, achene 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. This includes 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 writing applications that span sites extremely difficult.
- *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 existing mechanisms are not weakened by Legion. Further, we must provide mechanism 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 inter-operability 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 inter-operability*: 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 that heterogeneous architectures are integrated. Inter-operability also means that 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 re-configuration is a necessity — both for Legion itself, and for applications.

In addition to these goals, several constraints restrict our design—for example:

- *We cannot 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, retrain many of their users, and possibly make them incompatible with other systems in their organization. Our experience with Mentat [ref] indicates that it is sufficient to layer a system on top of an existing host operating system.
- *We cannot legislate changes to the interconnection network*: We must initially assume that the network resources, and the protocols in use, are a given. Much as we must accommodate operating system heterogeneity, we must live with the 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.
- *We cannot require that Legion run as "root" (or the equivalent).* Indeed, quite the contrary — to protect themselves, most Legion users will want it to run with the least possible privileges.

2.1 Legion's Object Foundation

The common framework that enables a coherent solution to these problems is object-orientation. In Legion all components 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 paradigm's encapsulation and inheritance properties, will make accessible a variety of the benefits often associated with the paradigm, including, software reuse, fault containment, and reduction in complexity. The need for the paradigm is particularly acute in a system as large and complex as Legion.

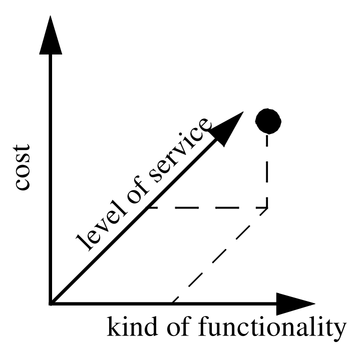
Objects, written in either an object-oriented language or other languages such as HPF Fortran, will encapsulate their implementation, data structures, and parallelism, and will interact with other objects via well-defined interfaces. In addition they may also have associated inherited timing, fault,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

While we are committed to the object-oriented paradigm we recognize that Legion will need to support applications written in a variety of languages in order to support existing legacy codes, permit organizations to use familiar languages (C, ADA, Fortran), and support a variety of parallel processing languages and tools. We intend to provide multilanguage support, and interoperability between user objects written in different languages in three ways, by generating object "wrappers" for codes written in languages such as Fortran, ADA, and C; by exporting the Legion run-time system interface and retargeting existing compilers and tools; and by a combination of the two.

2.2 System philosophy

Complementing our use of the object-oriented paradigm is one of our driving philosophical themes—*we cannot design a system that will satisfy every user's needs*. We must design Legion to allow users and class implementors the greatest flexibility in the semantics of their applications; We must, therefore, resist the temptation to provide "the solution" to a wide range of system functions. Users should be able, whenever possible, to select both the *kind* and the *level* of functionality, and make their own trade-offs between these and cost.

Neither the "kind" nor the "level" of functionality are linearly ordered, but a simplistic model is that of a multi-dimensional space. The needs of users will dictate where they need to be and/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 to be in the future. Indeed, if we were to dictate a system-wide "solution" to almost any of issues raised in our list of objectives we would preclude large classes of potential users and uses.



Consider security with respect to both kind and level of functionality. Some users are mostly concerned with privacy, while others are more concerned with the integrity of their data — both banks and hospitals are in the later category for example. Some users are content with password authentication, while others might feel the need for stronger user identification — signature analysis, fingerprint verification, or whatever. Both of these are examples of differences in the *kind* of security functionality. The size of cryptographic key, on the other hand, is an issue of the degree, or 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.

In the Legion approach, rather than providing a fixed security mechanism, with the result that no one is completely satisfied, users may choose their own trade-offs by implementing their own policies or by using existing policies via inheritance. Some users may require a policy that requires every method invocation to have 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 will be expensive (CPU, bandwidth, time). Alternatively, an application that requires low overhead cannot afford such a policy and should not be forced to use it. Such an application could instead choose a light-weight policy that simply checks if 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, don't replicate writable files, use a cache invalidate protocol, use 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 a strict "reads deliver the last value written" semantics, others know that the file is read only so that consistency protocols are a waste of time, while others may need different semantics for the file in different regions of the application. Independent of the file semantics, some users may need automatic backup and archiving frequently, while others may not. The point is that they system should not make such decisions for users, they should select the kind and level of service they require.

The philosophy has been extended into the system itself. The Legion object model specifies the composition and functionality of Legion's core objects—those objects that cooperate to create, locate, manage, and remove objects from the Legion system. Legion specifies the functionality, not the implementation, of the system's core objects. Therefore, the core will consist of extensible, replaceable components. The Legion project will provide implementations of the objects that comprise the core, 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 that meet the users' own specific requirements.

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

3 Experiences - The CWVC and the I-way

In the summer of 1995 we released our first prototype Legion implementation, the *Campus Wide Virtual Computer* (CWVC). This first implementation is based on an earlier object-oriented parallel processing system, Mentat [5]. 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 only transform a system so far before it begins to show signs of the stress; it is often better to design from the ground up so that the resulting system has a clean coherent architecture, rather than a patchwork of modifications based on a solution for a different problem.

The campus-wide virtual computer is a direct extension of Mentat onto a larger scale, and is a prototype for the nationwide system and reflects the fact that the university is a microcosm of the world. The computational resources at the 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 nationwide 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 must not be negatively impacted. Further, each department operates essentially as an island of service, with its own NFS mount structure, and trusting only machines in the island.

The CWVC is both a prototype and a demonstration project. The objectives are to:

- 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 turn-around time) at reduced cost,
- act as a testbed for the nationwide Legion.

The CWVC consists of over one hundred workstations and an IBM SP-2 in six buildings using two completely disjoint underlying file systems. We have developed a suite of tools to address common problems encountered (Table 1). In collaboration with domain scientists both at the University of Virginia and elsewhere we have also developed a set of applications that exploit the environment (Table 2).

TABLE 1. Campus Wide Virtual Computer Toolset

Problem	Tools available
Writing parallel application	CWVC-aware PVM, parallel C++, Fortran wrappers
Multiple separate file systems	Federated file system - 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 is hard	Post-mortem playback using off-the-shelf debuggers, e.g., dbx.
Host/network failure	Automatic system reconfiguration and limited application fault-tolerance

TABLE 2. Sample of existing CWVC applications

Discipline	Application
Biology	DNA&protein sequence comparison
Computer Science	Parallel databases & 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

In addition to the local production environment we have also demonstrated the CWVC on wide-area systems. During Supercomputing ‘95 in San Diego we ran the CWVC on the I-Way, an experimental network connecting the NSF supercomputer centers, several of the DOE and NASA labs, and a number of other sites. Many of the connections were at DS-3 (45 mb/sec) and OC-3 (155 mb/sec) rates. The CWVC was installed at three sites using seven hosts of three different architectures. At NCSA (Urbana) we used for SGI Power Challenges and the Convex Exemplar. At the CTC (Cornell) and ANL (Argone) we used IBM SP-2’s.

Once the IP routing tables had been 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 though was that user applications required no changes to run in the new environment.

For our demonstration excercised our utilities, and ran one of our applications, *complib*, on the I-way. *Complib* compares two DNA or protein sequence databases using one of several selectable algorithms [6]. The first database was located at ANL, while the second was located 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. All communication, placement, synchronization, and code and data migration was handled completely transparently by Legion.

Since Supercomputing we have repeated the demonstration several times, and are now in the process of constructing a more permanent prototype. The new prototype will span NCSA and SDSC and will opperate as a part of the DARPA funded Distributed Object Computation Testbed.

4 Related work

The vision of a seamless metacomputer such as Legion is not novel; worldwide computers have been the vision of science fiction authors and distributed systems researchers for decades. However, to our knowledge no other project has the same broad scope and ambitious goals of 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 workstation farms, 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, and draw upon 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, IDL's 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 RPC between objects. Further, DCE does not support parallel computing; instead, the emphasis is on client-server based distributed computing. Legion, on the other hand, is based upon 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 the implementation. Users and resources owners are permitted—even encouraged—to provide their own implementations of "system" services. Our core model is completely extensible and provides 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 Interface Description Language, and a specification for the functionality of runtime systems that enable access to objects (ORB's). But like DCE, CORBA is based on a client-server model rather than a parallel computing model, and less emphasis is placed on issues such as object persistence, placement, and migration.

Other projects share many of the same objectives but not the scope of Legion. Nexus[4] provides communication and resource management facilities for parallel language compilers. Castle[3] is a set of related projects that aims to support scientific applications, parallel languages and libraries, and low-level communications issues. The NOW[1] project provides a somewhat more unified strategy for managing networks of workstations, but is intended to scale only to hundreds of machines instead of millions. Globe[9] is an architecture for supporting wide area distributed systems, but does not yet seem to address important issues such as security and site autonomy.

In its intended application for distributed collaboration and information systems, Legion might be compared to the World Wide 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 lie in Java's lack of a remote method invocation facility, lack of support for distributed memory parallelism, and its 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 are a poor fit for many applications.

5 Summary and the Future

Legion is an abitious middleware project that will provide a solid, integrated, conceptual foundation on which to build applications. One could argue that Legion is perhaps too abititious, that there are just too many different complex issues to addrerss. The number of different issues is certainly a risk. On the other hand, eventually there will be Legion-like metasystem software; it 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 different solutions. Legion's strength is that its object model that was designed from its very inception both for the intended environment and for extensibility We feel that these attributes will permit Legion to readily adapt to an ever changing world.

Legion – as defined by by our objectives, is not yet a reality. While we have a prototype, the purpose of the prototye is to demonstrate the feasibility of construcing a wide-area system and to permit application and tool development to occur concurrently with system implementation. It is not designed to evolve directly into a complete Legion implementation.

In March of 1996 we began our implementation of the core Legion object model. Unike the existing prototype the new implementation incoporates mechanism for security, fault-tolerance, application directed scheduling, autonomy, scalable binding, etc. This "full blown" implementation re-uses many components of the prototype, e.g., the compiler, debuggers, and so on, but for the most part is being writen from the ground up. We expect to have a useable, documented, system available for public use in mid 1997. The system, and sources, will be pubically available.

6 References

[1] T.E. Anderson, D.E. Culler, D.A. Patterson, and the NOW team, "A Case for NOW (Networks of Workstations)," to appear in *IEEE MIcro*.

[2] Ron Ben-Naten, *CORBA: A Guide to the Common Object Request Broker Architecture*, McGraw-Hill, 1995.

[3] The Castle Project, University of California, Berkeley, http://http.cs.berkeley.edu/projects/parallell/castle/castle.html.

[4] I. Foster, Carl Kesselman, Steven Tuecke, "Nexus: Runtime Support for Task-Parallel Programming Languages," Argonne National Laboratories, http://www.mcs.anl.gov/nexus/paper/.

[5] A. S. Grimshaw, A. J. Ferrari, and E. A. West, "Mentat" *Parallel Programming Using C++*, Editor: Greg Wilson, MIT Press, 1996.

[6] A. S. Grimshaw, E. A. West, and W.R. Pearson, "No Pain and Gain! - Experiences with Mentat on Biological Application," *Concurrency: Practice & Experience*, pp. 309-328, Vol. 5, issue 4, June, 1993.

[7] H.W. Lockhart, Jr., *OSF DCE Guide to Developing Distributed Applications*, McGraw-Hill, Inc. New York 1994.

[8] Sun Microsystems, "The Java Language Specification," Version 1.0 Beta,Oct. 30, 1995

[9] M. van Steen, P. Homburg, L. van Doorn, A.S. Tanenbaum, and W. de Jonge. "Towards Object-based Wide Area Distributed Systems". In L.-F. Carbrera and M. Theimer, (eds.), *Proceedings International Workshop on Object Orientation in Operating Systems*, pp. 224-227, Lund, Sweden, August 1995.