2K: A Distributed Operating System for Dynamic Heterogeneous Environments*

Fabio Kon† Roy H. Campbell
M. Dennis Mickunas Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
{f-kon,rhc,mickunas,klara}@cs.uiuc.edu

Francisco J. Ballesteros
Systems and Communications Group
Rey Juan Carlos University, Madrid
nemo@gsyc.escet.urjc.es

Abstract

The first decades of the new millennium will witness an explosive growth in the number and diversity of networked devices and portals. We foresee high degrees of mobility, heterogeneity, and interactions among computing devices connected to global networks. While previous research in distributed operating systems solved many problems related to resource management, they seldom addressed the problems of heterogeneity and dynamic adaptability. On the other hand, middleware solutions, like CORBA and Java/Jini, solve part of the heterogeneity problem by permitting seamless communication among different platforms. But, they do not address dynamic resource management and adaptability for applications requiring high-performance distributed computing.

This paper presents 2K*, an integrated operating system architecture that addresses the problems of resource management in heterogeneous networks, dynamic adaptability, and configuration of component-based distributed applications.*

1. Introduction

Modern computing environments are characterized by a high level of dynamism. Two major kinds of dynamic changes occur frequently. The first refers to structural changes such as hardware and software upgrades, protocol and API updates, and operating system evolution. The second refers to dynamic changes in the availability of memory, CPU, network bandwidth and, in mobile systems, connectivity and location. Drastic changes may occur in a few seconds, impacting the performance of user applications profoundly. Existing operating systems offer very little support for managing, adapting, and reacting to these changes; all the work is left to the applications or to users and system administrators who must take care of them manually. Since large corporate and academic networks tend to be heterogeneous, the configuration work is multiplied by the number of supported platforms.

This scenario is further aggravated as mobile systems become more common and digital computing becomes ubiquitous. Users accessing the global network from anywhere in the world would like to have prompt access to their computing environment irrespective of their locations.

Thus, we need a flexible and adaptable architecture that permits the dynamic instantiation of customized user environments at different locations in the distributed system with proper care for dependencies. Existing system architectures, however, do not provide proper management of the dependencies among system and application components, which makes it difficult to support automatic configuration of component-based environments in a reliable way. It is hard to create robust and efficient systems if the dynamic dependencies between components are not well understood. For those reasons, proper dependence management is a major requirement for the next generation middleware and operating systems.

Management of dynamism and dependencies is also crucial for computationally intensive distributed applications. In many cases, the dynamic variations in the environment are so high that the distribution overhead becomes larger than the speedup obtained with the additional parallelism.

In this paper, we present an overview of *2K*, a novel network-centric operating system that extends the previous work on configurable operating systems. It redesigns the system API to provide an integrated solution to the problems mentioned above, namely, the existence of an integrated environment that supports dynamic changes, auto-

*This research is supported by the National Science Foundation, grants CCR 96-23867, 98-70736, 99-70139, and EIA 99-72884EQ.

†Fabio Kon is supported in part by a grant from CAPES-Brazil, process 1405/95-2.

matic configuration, heterogeneity, and distributed resource management.

2. Resource Management in Heterogeneous Environments

The basic task of both centralized and distributed operating systems is to manage the resources of a machine (or a collection of machines) and safely export them to their users. Conventional operating systems, however, are not able to manage the resources of collections of heterogeneous machines.

CORBA and Java/Jini emerge as powerful technologies for interoperability in heterogeneous environments. But they both lack the notion of a "user" and do not provide support for dynamic resource management either in a single machine or in a distributed environment.

Our approach combines the benefits of CORBA with those of distributed operating systems. It provides management of distributed resources while being able to handle different hardware platforms and different underlying, single-node operating systems.

The *2K* system supports a completely object-oriented view of the distributed computing environment; distributed hardware and software resources are encapsulated as CORBA objects while distributed operating system services (e.g. file, naming, and execution services) are exported as CORBA services. Applications run within this relatively homogeneous environment built on top of highly heterogeneous distributed environments.

To achieve optimal application performance in a dynamic environment of distributed resources, the middleware must be configurable and able to adapt to dynamic changes in resource availability and in the software and hardware infrastructures. As described in section 3.5, *2K* uses a dynamically configurable reflective ORB [12] to provide a high-level of flexibility to applications that can benefit from it by tuning the CORBA implementation to their specific needs. But it also keeps the complexity away from applications that prefer to use the CORBA distributed object model without worrying about the underlying details.

The reflective ORB solves the problem of *how* to adapt the system to the application needs but it does not address the problem of *when* and *what* to adapt. We do that by maintaining an explicit representation of the dynamic dependencies among system and application components and by allowing the inspection and monitoring of the dynamic state. *2K* enhances resource management with algorithms for QoS provision including admission control, negotiation, reservation, and renegotiation. Application programmers have then complete access to the system's dynamic state and are able to implement application-specific adaptations while the system guarantees that QoS is preserved.

3. *2K* System Model

2K adopts a *network-centric* model in which all entities, users, software components, and devices exist in the network and are represented as CORBA objects. Each entity has a network-wide identity, a network-wide profile, and dependencies upon other network entities. When a particular service is instantiated, the entities that constitute that service are assembled.

In contrast to existing systems where a large number of non-utilized modules are carried along with the basic system installation, our philosophy is based upon a "What You Need Is What You Get" (WYNIWYG) model. The system configures itself automatically and loads a minimal set of components required for executing the user applications in the most efficient way.

As shown in Figure 1, this philosophy is realized by leveraging standard CORBA services such as Naming, Trading, Security, and Persistence and extending the CORBA model with the addition of services for QoS-Aware Resource Management, Automatic Configuration, and Code Distribution.

3.1. Automatic Configuration Service

To address the problems described in the previous sections, the *2K Automatic Configuration Service* manages two distinct kinds of dependencies:

1. *prerequisites*, i.e., the requirements for loading an inert component into the runtime system, and
2. *dynamic dependencies* among loaded components in a running system.

As long as the system has access to the requirements for installing and running a software component, the installation and configuration of new components can be automated. As a byproduct of this knowledge, component performance can be improved by analyzing the dynamic state of system resources, by analyzing the characteristics of each component, and by configuring them in the most efficient way.

3.1.1. Prerequisites

The prerequisites for a particular inert component specify any special requirement for properly loading, configuring, and executing that component. Prerequisites specify 1) the type and share of hardware resources that a component needs and 2) the software services (i.e. other components) it requires.

The first kind of prerequisites lets the QoS-aware Resource Management Service determine where, how, and

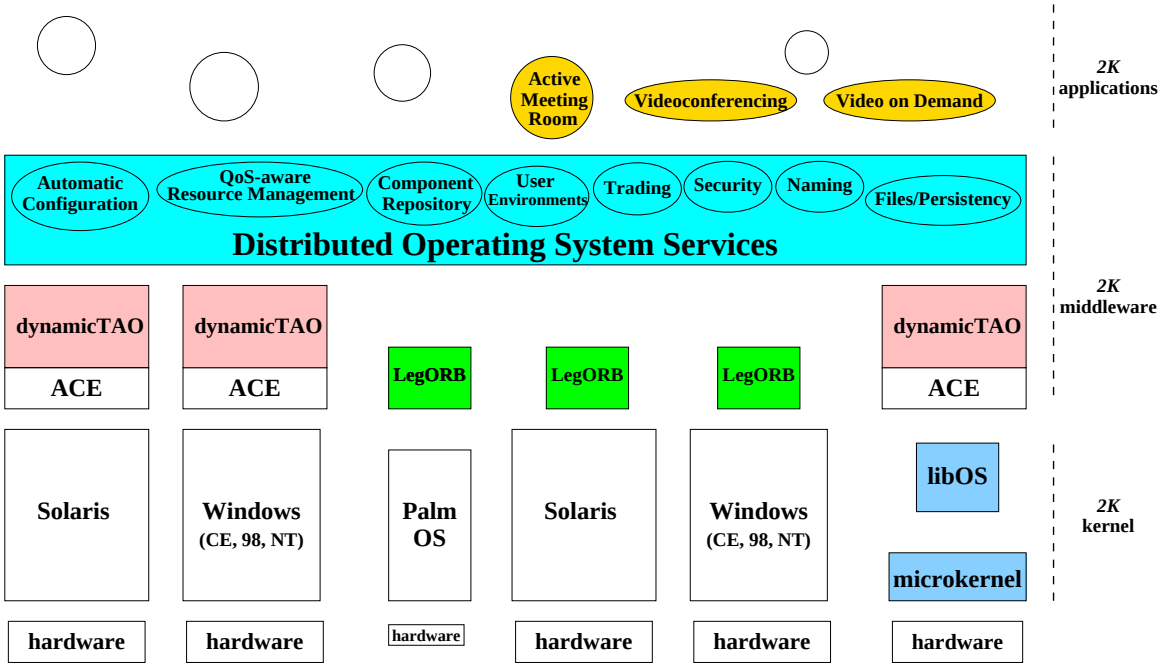

Figure 1. The *2K* overall architecture

when to execute each component. It uses this data to enable proper admission control, resource negotiation, reservation, and scheduling.

The second kind of prerequisites determines which auxiliary components must be loaded and which other software services must be located. As the Automatic Configuration Service parses the software prerequisite specifications, it verifies whether it is necessary to create new instances of the required components in the *2K* runtime. If necessary, it contacts the Component Repository, fetches the component binary code compiled for that specific platform and dynamically loads it.

As of now, the prerequisite specifications are created manually by component developers. In the future, we expect that this task will also be automated.

3.1.2. Dynamic Dependencies

While the Automatic Configuration Service parses the prerequisite specifications, fetches the required components from the Component Repository, and dynamically loads their code into the runtime, it uses the information in the prerequisite specifications to create a runtime representation of inter-component dependencies. This representation uses CORBA objects called *ComponentConfigurators* (see Figure 2). These objects store the dependencies as lists of CORBA Interoperable Object References (IORs), pointing to other component configurators, forming a dependence graph of distributed components.

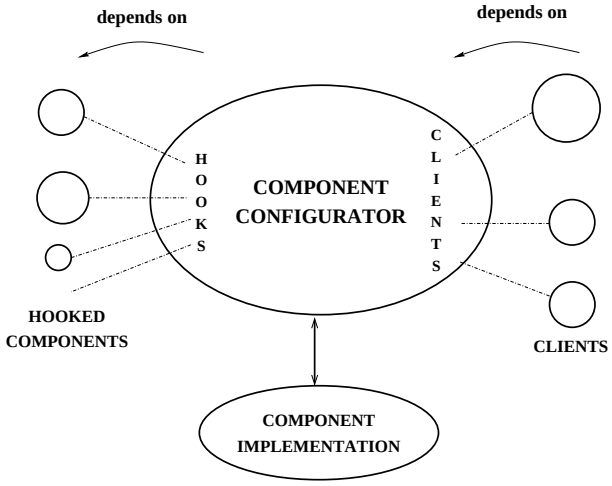

Figure 2. Reification of component dependence

With information about its runtime dependencies, applications can refer to its own requirements, selecting different components to fulfill their needs in different environments and at different times. In addition, the underlying system can manipulate the application dependencies in order to optimize performance or to adapt to dynamic changes in the

environment.

When a *2K* component fails, the system inspects its dependencies and informs the proper components about the failure. Applications can customize the system by implementing specialized instances of the *ComponentConfigurator*, for example, to recover from a failures by replacing the faulty component with a new one. The same mechanism can be used for adapting the system and its components to changing parameters such as network bandwidth, CPU load, resource availability, and user access patterns.

3.2. Mobile Configuration Agents

In addition to the *pull*-based approach for code distribution described in 3.1.1 where the system fetches components from the Component Repository, *2K* also supports a *push* mechanism based on mobile agents. In many cases, this alternate mechanism can improve system performance by distributing code updates in a scalable way.

The reflective ORBs are organized as a distribution network through which system administrators or applications can send *configuration* and *inspection* agents [11]. Agents may contain both configuration commands (to change the configuration of the ORBs and the applications running on top of them) and new implementations for system and application components (in the form of dynamically loadable libraries or Java bytecode).

The combination of these mechanisms provides a flexible infrastructure for dynamic software updates. By working on an environment that requires less manual administration, users and developers can concentrate on more important tasks and improve their productivity.

3.3. QoS-Aware Distributed Resource Management

The *2K* Resource Management Service [29] relies on Local Resource Managers (LRMs) present in each node of a *2K* cluster and whose task is to export the hardware resources in that node to the whole distributed system. LRMs send periodic updates of the state of their resources to the Global Resource Manager (GRM), a replicated service that maintains an approximate view of the cluster resource utilization state. The GRM then utilizes this information as a hint for performing QoS-aware load distribution within its cluster. In the future, we intend to combine groups of GRMs hierarchically to provide hardware resource sharing across multiple clusters connected through the Internet.

The LRMs are also responsible for performing QoS-aware admission control, resource negotiation, reservation, and scheduling of tasks on a single node. This is achieved with the help of a Dynamic Soft Real-Time Scheduler (DSRT) [16] that runs as a user-level process in conventional operating systems like Solaris and Windows. It is able to use the system's low-level real-time API to provide QoS guarantees to applications with soft real-time requirements.

2K uses a CORBA Trader [18] to supply resource discovery services, which allow applications to request resources based on QoS specifications. In this way, the system helps parallel and distributed applications achieve the best performance with the available resources.

3.4. Dynamic Security

Access to *2K* services is restricted to controlled CORBA interfaces. For that, we utilize the OMG Standard Security Service [18] that comprises authentication, access control, auditing, object communication encryption, non-repudiation, and administration of security information.

Our prototype implementation of the CORBA Security Service utilizes the *Cherubim* security framework [4] to support dynamic security policies [21]. The reflective ORB allows on-the-fly reconfiguration of the Security Service, facilitating the adoption of situation-specific policies and mechanisms for authentication and encryption. The implementation currently supports various access control models including Discretionary Access Control (DAC) and Mandatory Access Control (MAC) [24]. We are now extending it to support Role-Based Access Control (RBAC) [22], which will be the basis for security in large-scale *2K* environments.

The possibilities for dynamically configuring the security subsystem that *2K* provides are very useful for a wide range of applications in several situations. As an example, consider a computationally intensive application that runs initially in a single cluster and later expands itself to use the processors of several clusters connected via the public Internet. It may be acceptable to use lightweight encryption and soft access control in the intranet but it may be necessary to apply strong encryption and very tight access control policies when communicating over the public Internet. The same happens with mobile computers and PDAs that may need to use different mechanisms and policies as they move from one domain to another.

3.5. Reflective ORBs

One of the major constituent elements of *2K* is *dynamicTAO* [12], a CORBA-compliant reflective ORB. *dynamicTAO* is an open source extension of the TAO ORB [25] that enables on-the-fly reconfiguration of the ORB internal engine and of applications running on top of it. In *dynamicTAO*, we used the *ComponentConfigurator* model described in section 3.1.2 to represent the dependence relationships between ORB components and between ORB and application components. The current version supports

safe dynamic reconfiguration of the strategies that control aspects such as concurrency, security, and monitoring. *dynamicTAO* exports an interface for loading and unloading modules into the system runtime, and for inspecting and changing the ORB configuration state.

After our experience in developing applications with both open source and commercial ORBs, we came to the conclusion that typical applications utilize just a very small fraction of the services and functionalities offered by common ORBs. Besides, one of the criticism that CORBA often receives is that it is too heavyweight to be used in small devices and embedded systems. Although *dynamicTAO* is configurable dynamically, its memory footprint is never less than a few megabytes, which makes it inappropriate for environments with limited resources and for applications with stringent resource requirements. This motivated our group to develop a new ORB architecture called *LegORB* [23]. It can be dynamically customized to adapt to resource availability and to accommodate the requirements of different applications and devices at different moments.

Unlike TAO, *LegORB* is designed with componentization and dynamic reconfiguration as a fundamental premise. Careful design and implementation has allowed us to achieve surprising results in terms of code size. A minimal configuration of *LegORB* that is able to send simple CORBA requests to standard ORBs occupies only around 6Kbytes on a PalmPilot running PalmOS. The development of *LegORB* is still in its early stages, but the preliminary results indicate that it will be not only a good choice for embedded systems and PDAs, but also for high-performance workstations where *LegORB* can perform even faster than highly-optimized commercial ORBs.

4. Lessons Learned

In the past three years of work on the design and implementation of *2K*, our research group has learned a number of lessons that we consider significant.

It is unlikely that a large number of users would be willing to adopt a completely new research operating system to use on a daily basis. Thus, we decided that *2K* would have to be able to run on top of other operating systems and, if necessary, co-exist with traditional applications. In that manner, users of traditional systems can extend the functionality of their machines by using the QoS-awareness, network-centrism, code distribution, and dynamic configuration properties of *2K* to manage their conventional system. The users that need the extra control and performance offered by a customizable microkernel can choose to boot their machines with the *2K* microkernel [1].

The use of a standard platform like CORBA provides two important benefits. First, we have the opportunity of re-using a large number of distributed services and applications that were developed by the CORBA community, saving us a lot of development time. Second, our system services become available to a wide community and accessible by any CORBA client. Since the interfaces are defined using OMG IDL, *2K* can be used by applications built on a completely different code base; all the applications need to interact with *2K* is a standard CORBA ORB. In addition, the use of IDL interfaces among distributed and even co-located system components improves system organization considerably.

On the other hand, the possibility of changing the implementation of the different aspects of the CORBA middleware through the use of a reflective ORB opens new possibilities in terms of code re-use. If a scientific application requires a special underlying protocol or a particular optimization, the programmer can implement it as a reflective ORB module, making it available to other applications with similar needs. The reflective architecture allows the deployment of these different protocols and optimizations without modifications to the application code.

Finally, by implementing CORBA interfaces for heterogeneous computers and devices, one can deal with the details of their specific protocols only once. After the CORBA wrapper is completed, the device becomes part of the distributed system and can be accessed easily by any other entity present in the network.

5. Performance Considerations

Our reflective ORB is an extension of TAO [25], a CORBA-compliant ORB that optimizes inter-object communication by using different protocols depending on the location of the objects. Calls to co-located servers can be as fast as virtual method calls on a C++ object. The general impression that CORBA was too large and slow corresponds to first-generation brokers. Recent performance measurements [20] suggest that contemporary CORBA implementations are efficient and that even faster implementations will appear.

On Linux running on a single 450MHz Pentium II with 256M of RAM, it takes 236 μs for TAO to perform a cross-domain method invocation with a single parameter [13] which is an acceptable figure for a wide-range of applications. When a better performance is required, applications may customize the ORB to optimize the system. TAO supports pluggable protocols [19], which allow specific transports to be used to maximize application performance.

5.1. Mobile Configuration Agents

We measured the performance of our infrastructure for dynamic configuration based on mobile agents (see section 3.2) by injecting configuration and inspection agents into

a network of six ORBs running on Sun Ultra-2 machines connected by a 100Mbps Ethernet. The inspection agent carried code to collect information about the state of the six reflective ORBs, bringing it back to the administrator. The total average time for sending, processing, and returning the agent was 101 milliseconds.

The configuration agent carried instructions to load a 30Kbyte component to the runtime of the six ORBs and attach the new component to a running application in each of the ORBs. It took 265 milliseconds, on average, to complete its task and return the results to the administrator.

Although these numbers can be improved significantly with more tuning and optimizations, they show that it is possible to carry out dynamic configuration of a collection of distributed components in few tenths of a second.

In another experiment, we measured the performance of our infrastructure in a wide-area system composed of nine nodes, three in the USA, three in Brazil, and three in Spain. Figure 3 shows a comparison between the performance of our agent-based approach and a conventional point-to-point approach as the size of the component being uploaded increases. Each value is the average of ten runs of the experiment, the vertical bars represent the standard deviation.

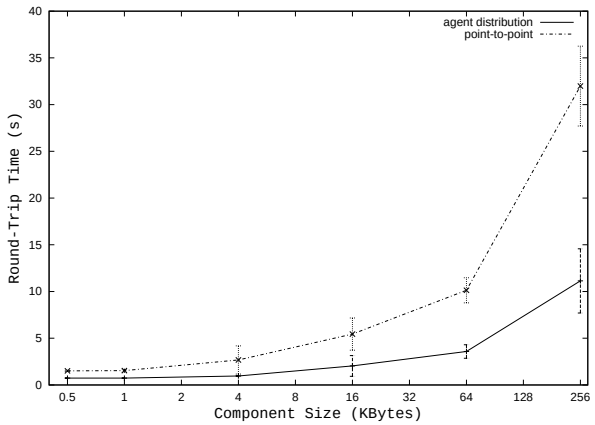

Figure 3. Agent uploading a new component to nine nodes

In [11] we present a more detailed analysis of our mobile agent infrastructure and discuss the improvements that agents can bring to the management and reconfiguration of wide-area distributed systems.

5.2. Automatic Configuration

The *2K* Automatic Configuration Service is implemented as a library that can be linked to any application. A program enhanced with this service becomes capable of fetching components from a remote Component Repository and dynamically loading and assembling them into its local address-space. The library requires only 157Kbytes of memory on Solaris 7, which makes it usable even on machines with limited resources such as a PalmPilot.

Figure 4 shows the total time for the service to load from one to eight components of 19.2Kbytes each. The time includes fetching the component code from the remote Component Repository, saving it in a local cache, parsing the component prerequisites, and dynamically linking the code in the local address-space. The experiments were carried out on two Sparc Ultra-60 machines running Solaris 7 and connected by a 100Mbps Ethernet. The Component Repository was executed on one of the machines and a test application with the Automatic Configuration Service on the other. Each value is the average of five runs of the experiment.

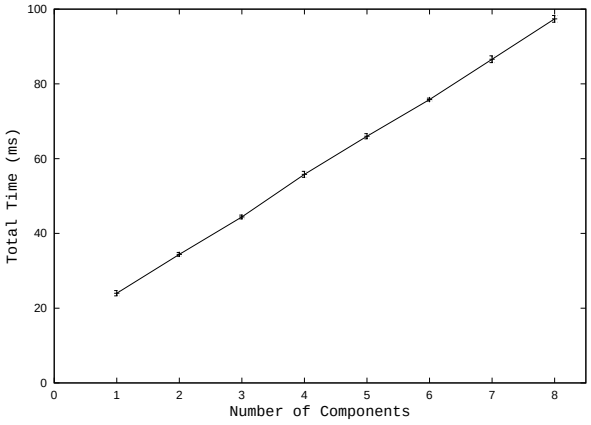

Figure 4. Automatic Configuration Service performance

Although there is still much room for improvements and performance optimizations in the protocols used by the Automatic Configuration Service, the results presented here are very encouraging. They demonstrate that it is possible to carry out automatic configuration of a distributed component-based application, in a local network, within a tenth of a second, which is an acceptable delay for a wide range of applications.

6. Related Work

Our work builds on previous and ongoing research in a number of different areas including operating system architecture, computational grids, configurable middleware, mobile agents, dynamic security, dynamic configuration, and software architecture.

SPIN [2] and VINO [26] are adaptable systems which load code into the kernel to allow system extensions. We build on their work, and employ code downloading (through

the network) to install new components into *2K* nodes. Choices and its derivatives [3] implement operating system services by means of a collection of object-oriented frameworks. These systems, however, do not address heterogeneous distributed systems.

Spring [15] is an object-oriented, distributed operating system which also uses IDL-based interfaces for system services. *2K* takes the ideas introduced by Spring a step further by adopting the CORBA communication model and standard CORBA services as the glue to connect *heterogeneous* hardware and software platforms and by representing and managing inter-component dependence.

Systems like Condor [14] are targeted to high performance computing on clusters of workstations. They rely on a central resource manager that starts processes on workstations with spare cycles.

The Globus project [6] provides a "computational grid" [7] integrating heterogeneous distributed resources in a single wide-area system. It supports scalable resource management based on a hierarchy of resource managers similar to those of *2K*. But, unlike *2K*, Globus is tailored to computationally-intensive applications such as large-scale simulations and teleimmersive applications.

Although the Globus design includes a service for managing application code (the Globus Executable Management service or GEM), the current implementation does not yet include it as a separate service. The name of this service, however, implies that Globus views an application as a single executable, rather than a collection of components that can be dynamically instantiated. Since Globus is one of the most important projects in this area, we hope that their system evolves to incorporate support for dynamic, component-based applications.

Recent research targeted at using the Internet as "the computer" has led to systems like Globe [28], Legion [9, 8], and WebOS [27]. Although some may be customizable, they do not consider adaptability, dependence management, and automatic configuration as a primary requirement. To the best of our knowledge, none of the systems mentioned above include a model enabling automatic configuration of component-based systems on distributed, heterogeneous environments.

Legion is the system that shares most similarities with *2K* as it also builds on a distributed, reflective object model. However, the Legion researchers focused on developing a new object model from scratch. Legion applications must be built using Legion-specific libraries, compiler, and runtime system (the Legion's ORB). In contrast, we focused on leveraging CORBA technology to build an integrated architecture that could provide the same functionality as Legion, while still preserving complete interoperability with other CORBA systems. In addition, our work emphasizes automatic configuration and dependence management, which are not addressed by Legion.

One of the major contributions of our work is to combine some of the important research results provided by WebOS, Globe, Legion, and Globus into a completely standard environment based on CORBA objects and standard CORBA services. It also brings automatic configuration – previously limited to isolated tools for application development – to the core of a distributed, object-oriented operating system. With the evolution of mechanisms for automatic configuration, such as the ones available in *2K*, we foresee a bright future for distributed operating systems for high-performance computing that will be easy to manage, comfortable to use, and extremely powerful.

7. Conclusions

We expect great changes in the environments for high performance distributed computing in the first decade of the new millennium, including higher degrees of dynamism, mobility, heterogeneity, and interactions among heterogeneous computing devices connected to global networks. Traditional middleware and operating system architectures are not prepared to provide efficient resource management for these highly dynamic heterogeneous environments.

In this paper, we presented an integrated operating system architecture for managing distributed heterogeneous resources using recent advances in software systems technology. *2K* provides support for code distribution, configuration of component-based distributed applications, and QoS-aware distributed resource management. *2K* can run both as "middleware" on top of traditional operating systems and as an integrated architecture with our customized microkernel directly on top of the hardware.

2K uses and offers services based on the CORBA standard which opens a wide variety of possibilities for integration with other systems and applications.

Our ongoing work in *2K* includes a flexible and adaptable distributed data management system [10], a service for managing user environments in mobile and wide-area systems [5], an infrastructure for managing active (physical) spaces, and mechanisms for enabling dynamic instantiation of distributed QoS-aware multimedia applications [17].

Availability

Documentation and source code for the *2K* microkernel, middleware, and distributed services can be found at `http://choices.cs.uiuc.edu/2K`.

References

[1] F. J. Ballesteros, C. Hess, F. Kon, S. Arévalo, and R. H. Campbell. Object Orientation in Off++ - A Distributed

Adaptable μKernel. In *ECOOP'99 Workshop on Object Orientation and Operating Systems*, pages 49–53, Lisbon, June 1999.

[2] B. N. Bershad et al. Extensibility, Safety and Performance in the SPIN Operating System. In *Proc. of the 15th SOSP*. ACM, December 1995.

[3] R. Campbell, N. Islam, P. Madany, and D. Raila. Designing and Implementing Choices: an Object-Oriented System in C++. *Communications of the ACM*, 36(9):117–136, Sept. 1993.

[4] R. Campbell and T. Qian. Dynamic Agent-based Security Architecture for Mobile Computers. In *Proceedings of the Second International Conference on Parallel and Distributed Computing and Networks (PDCN'98)*, pages 291–299, Australia, December 1998.

[5] D. Carvalho, F. Kon, F. Ballesteros, M. Román, R. Campbell, and D. Mickunas. Management of execution environments in 2k. In *Proceedings of the Seventh International Conference on Parallel and Distributed Systems (ICPADS'2000)*. IEEE Computer Society, July 2000.

[6] I. Foster and C. Kesselman. The Globus Project: A Status Report. In *Proceedings of the IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4–18, 1998.

[7] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, 1999.

[8] A. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey. Legion: an Operating Systems for Wide-area Computing. Technical Report CS-99-12, University of Virginia, March 1999.

[9] A. Grimshaw, W. Wulf, et al. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1), January 1997.

[10] C. K. Hess, F. J. Ballesteros, and R. H. Campbell. An Adaptable Distributed File Service. In *Proceedings of the ECOOP PhD Workshop on Object Oriented Systems (PHDOOS'00)*, Cannes, France, June 2000.

[11] F. Kon, B. Gill, M. Anand, R. H. Campbell, and M. D. Mickunas. Secure Dynamic Reconfiguration of Scalable CORBA Systems with Mobile Agents. In *Proceedings of the IEEE Joint Symposium on Agent Systems and Applications / Mobile Agents (ASA/MA'2000)*, Zurich, September 2000.

[12] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. C. Magalhães, and R. H. Campbell. Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, number 1795 in LNCS, pages 121–143, New York, April 2000. Springer-Verlag.

[13] D. L. Levine, S. Flores-Gaitan, and D. C. Schmidt. An Empirical Evaluation of OS Support for Real-Time CORBA Object Request Brokers. In *Proceedings of the International Symposium on Distributed Objects and Applications DOA99*, Edimburgh, Scotland, September 1999.

[14] M. Litzkow, M. Livny, and M. W. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, 1988.

[15] J. Mitchell et al. An Overview of the Spring System. In *Proceedings of Compcon 'Spring 1994*, Feb. 1994.

[16] K. Nahrstedt, H. hua Chu, and S. Narayan. QoS-aware Resource Management for Distributed Multimedia Applications. *Journal of High-Speed Networking, Special Issue on Multimedia Networking*, 7:227–255, 1998.

[17] K. Nahrstedt, D. Wichadakul, and D. Xu. Distributed QoS Compilation and Runtime Instantiation. In *Proceedings of the IEEE/IFIP International Workshop on QoS (IWQoS'2000)*, Pittsburgh, June 2000.

[18] OMG. *CORBAservices: Common Object Services Specification*. Object Management Group, Framingham, MA, 1998. OMG Document 98-12-09.

[19] C. O'Ryan, F. Kuhns, D. C. Schmidt, O. Othman, and J. Parsons. The Design and Performance of a Pluggable Protocols Framework for Real-time Distributed Object Computing Middleware. In *Proceedings of the IFIP/ACM Middleware'2000*, New York, April 2000.

[20] I. Pyarali, C. O'Ryan, D. Schmidt, N. Wang, A. S. Gokhale, and V. Kachroo. Using Principle Patterns to Optimize Real-Time ORBs. *IEEE Concurrency*, 8(1):16–25, January-March 2000.

[21] T. Qian. *Dynamic Authorization Support in Large Distributed Systems*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Nov. 1999.

[22] Ravi S. Sandhu and Edward J. Coyne and Hal L. Feinstein and Charles E. Youman. Role-based Access Control Models. *IEEE Computer*, 29(2):38–47, Feb. 1996.

[23] M. Román, D. Mickunas, F. Kon, and R. H. Campbell. LegORB and Ubiquitous CORBA. In *Proceedings of the IFIP/ACM Middleware'2000 Workshop on Reflective Middleware*, pages 1–2, Palisades, NY, April 2000.

[24] R. S. Sandu and P. Samarati. Access Control: Principles and Practice. *IEEE Communications Magazine*, 32(9):40–48, Sept. 1994.

[25] D. C. Schmidt and C. Cleeland. Applying Patterns to Develop Extensible ORB Middleware. *IEEE Communications Magazine Special Issue on Design Patterns*, 37(4):54–63, May 1999.

[26] C. Small and M. Seltezer. VINO: An Integrated Platform for Operating System and Database Research. Technical report, Computer Science Laboratory, Harvard University, Cambridge, MA 02138, 1994.

[27] A. Vahdat, T. Anderson, M. Dahlin, D. Culler, E. Belani, P. Eastham, and C. Yoshikawa. WebOS: Operating System Services For Wide Area Applications. In *Proceedings of the Seventh Symposium on High Performance Distributed Computing*, July 1998.

[28] M. van Steen, P. Homburg, and A. S. Tanenbaum. Globe: A Wide-Area Distributed System. *IEEE Concurrency*, 7(1):70–78, January 1999.

[29] T. Yamane. The Design and Implementation of the 2K Resource Management Service. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, February 2000.