2K: A Component-Based Network-Centric Operating System for the Next Millennium

Manoj B Patel
Computer Science and Engineering
University of Texas at Arlington
mpatel@cse.uta.edu

March 11, 2002

Abstract

After thirty years of explosive growth in computing and network technology, significant advances are made in the area of distributed operating systems. Even after all these advances, we have today's market place littered with devices such as PDAs, mobile phones, laptops, pagers, etc. which are not interconnected. Existing Distributed Operating Systems do solve many problems related to resource management, but they lack in the problem of heterogeneity, and dynamic adaptability, which are essential for dynamic heterogeneous environments. Besides Operating Systems, middleware solution, such as CORBA and Java/Jini, solve part of the heterogeneity problem by providing communication protocol between different platform but fail to address dynamic, distributed resource management and adaptability. The 2K Operating System, lead by group of scientists and graduate students at University of Illinios at Urbana-Champaign, aims to manage changes and seamlessly integrate new and old technologies in our dynamic heterogeneous world. The 2K Operating System is built to provide high degree of mobility, heterogeneity, and interactions among computing devices connected to the global network. The 2K provides both the middleware solution, which runs on top of the existing Operating Systems, and the full-scale operating system equipped with its own *Off*++ microkernel. The purpose of this paper is to present 2K, an adaptable, distributed, network-centric operating system.

1 Introduction

In our modern era of computing, there are hundreds if not thousands of dynamic operating systems available. Each one is designed for some specific purpose. Some dynamic operating systems manage good security policies, while other Operating Systems are better at performance. But from all these there is basic question emerges, why not make a "perfect" operating system, which is based on ideas from existing operating systems, to run on any kind of hardware and support any application most effectively and without reducing the performance. The answer to this question would most certainly be close to the 2K Operating System. The 2K Operating System builds on previous and ongoing research in a number of different areas including operating system architecture, middleware, mobile agents, dynamic security, dynamic configuration, and software architecture.

The rest of the paper is organized in following fashion: Section 2 gives background information on the network-centric operating system, JINI [1], and CORBA [2]. Section 3 gives overview of the 2K Operating System Model and its services. Performance analyses that were conducted on various 2K Operating System components are discussed in section 4. Finally, Section 5 gives discussion on how the 2K Operating System meets its goals and ongoing work on 2K.

2 Background

2.1 Network-centric Operating System

Network-centric computing is where computation is accomplished by collaboration between a large numbers of widely distributed peer nodes. Network-centric Operating System is a meta-OS that provides support for any kind of dynamic, heterogeneous environment.

2.2 Interoperability in heterogeneous environments

Heterogeneous environment consists of vast range of computers from different manufacturers running a variety of operating systems and managing data from a variety of sources. Such environments support not only have different hardware devices but also different software applications which are, very often, not compatible with each other for interoperability [3]. The very challenge of interoperability is to allow users, application programs, and computer systems to share information and work together despite differences. To assist in the interoperability in heterogeneous environments, JINI and CORBA had emerged as a powerful tool for developers.

2.2.1 JINI

Jini-based technology defines mechanism to support the federation of devices and software components into a single, dynamic distributed system. Figure 1 shows the categorized Jini Architectural overview.

	Infrastructure	**Programming Model**	**Services**
Base Java	Java VM RMI Java Security	Java APIs Java Beans …	JNDI Enterprise Beans JTS …
Java + Jini	Discovery/Join Distributed Security Lookup	Leasing Transactions Events	Printing Transaction Manager JavaSpaces Service …

Figure 1 Jini Architecture Segmentation

The components of the Jini system are divided into three categories: *infrastructure, programming model,* and *services*. The infrastructure is the set of components that enables building a federated Jini system, while the services are the entities within the federation. The programming model is a set of interfaces that enable the construction of reliable services, including those that are part of the infrastructure and those that join the federation. Distinction between these three categories is not clear because they are intertwined with each other. Jini is not discussed any further in this paper because main focus of this paper is on 2K Distributed Operating System, which uses CORBA instead of JINI-based technology. More information on Jini can be found in [1].

2.2.2. OMG CORBA

The Object Management Group (OMG) was formed in 1989 to develop, adopt, and promote standards for the development and deployment of applications in distributed heterogeneous environment [2]. OMG has developed a high-level vision of a completely distributed environment called Object Management Architecture or OMA. OMA, as shown in Figure 2, consists of system oriented components (Object Services and Object Request Broker (ORB)), and application oriented components (Application Interfaces, Domain Interfaces, and Common Facilities). Of these components ORB is responsible for setting foundation for OMA and manages all communication between its components. CORBA or common ORB architecture details the interfaces and characteristics of the ORB component of the OMA. The main features of CORBA are ORB Core, ORB Interface, CORBA IDL (Interface Definition Language) stubs and skeletons, Dynamic Invocation Interface (DII), Dynamic Skeleton Interface (DSI), and Object Adapter. Most of these are illustrated in Figure 3.

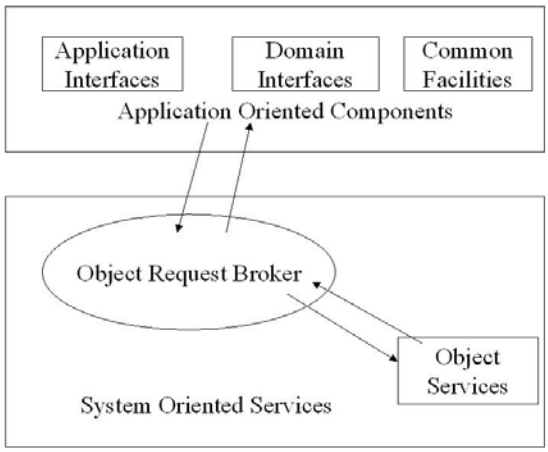

Figure 2. OMG Reference Model Architecture

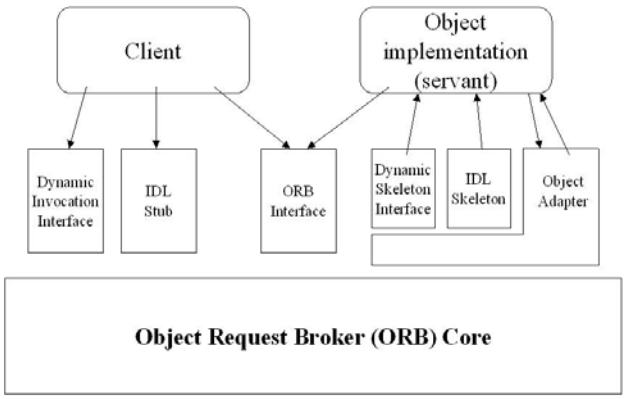

Figure 3. CORBA ORB Architecture

Object is the CORBA programming entity that consists of an *identity*, an *interface*, and an *implementation*, which is known as a *Servant.* Servant is an implementation programming language entity that defines the operations to support a CORBA IDL interface. Servants can be written in a C, C++, Java, Smalltalk, or Ada programming language.

Client invokes an operation on an object implementation. This invocation of a remote object is transparent to the caller and is as simple as calling a method on an object.

The ORB delivers requests to objects and returns any responses to the clients making the request [2]. The ORB provides a mechanism for a transparent communication between client and target object. When a client invokes an operation, the ORB is responsible for finding the object

implementation, transparently activating, delivering the request to the object, and returning any responses from object to client.

ORB interface's responsibility is to decouple applications from implementation details. This interface provides various helper functions such as converting object references to strings, and creating argument lists for requests made through the dynamic invocation interface. Interfaces for objects are defined in the IDL. CORBA IDL stubs and skeletons bind the client and server applications, respectively, to the ORB. An IDL compiler automates the transformation between CORBA IDL definitions and the target programming language.

In addition to the static invocation via stubs and skeletons, CORBA supports two dynamic invocation interfaces: *Dynamic Invocation Interface* (DII), and *Dynamic Skeleton Interface* (DSI) [2]. DII allows a client to directly access the underlying request mechanisms provided by an ORB. Applications can use DII to dynamically issue requests to object without requiring IDL interface specific stubs. DSI is the server side's analogue to the client side's DII. DSI provides dynamic dispatch to objects. The DII and DSI can be viewed as a generic stub and generic skeleton, respectively and neither is dependent upon the IDL interfaces of the object being invoked.

The last but not least subcomponent of CORBA is the Object Adapter (OA). As its name implies OA serves as the adapter or communication link between object implementation and the ORB core. OA assists the ORB with delivering requests to the object and with activating the object. OA can be specialized to provide system specific tasks.

Above is a brief overview of JINI and CORBA. To learn detail specifications about each system please refer to the [1] and [2] reference. The 2K OS, however, uses CORBA components extensively for its design and implementation. Jini is introduced here as an alternate to CORBA that is also available.

3. 2K OS Model

3.1. 2K overview

The 2K Operating System (OS) model is primarily based on "What You Need Is What You Get" (WYNIWYG) philosophy. In contrast to existing systems, 2K OS does not carry a large number of non-utilized modules with the basic system installation. It is designed to configure applications automatically and load a minimal set of components required for executing tasks in the most efficient way possible.

2K is a reflective OS since it is built on top of a reflective ORB called dynamic TAO [4], which will be discuss later. The 2K OS adopts a network-centric model in which all the entities including users, software components, and devices exist in the network and are represented as CORBA objects. It reuses many standard CORBA services such as Naming, Trading, and Security [5]. In addition, the 2K OS team extended 2K middleware to include other services for QoS-Aware Resource Management [6], Automatic Configuration, and Code Distribution. 2K

OS team also developed *7+* [7], which is a specially customized microkernel for the 2K OS based on earlier *Off* microkernel.

Following Figure 4, the 2K overall architecture, each component is briefly explained.

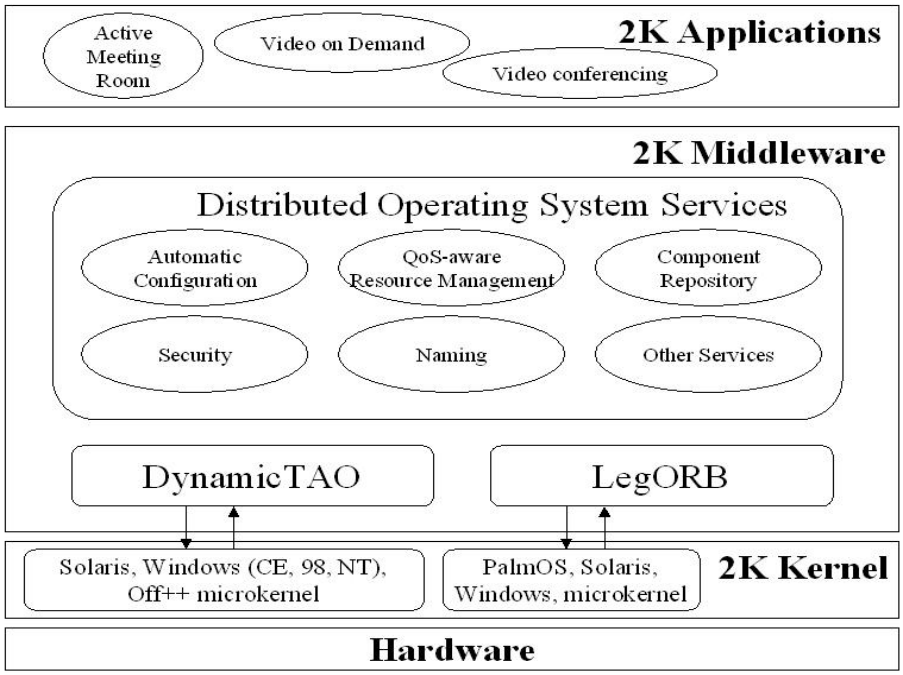

Figure 4: 2K Architecture

3.2. 2K middleware

As shown in Figure 4, 2K middleware consists of distributed OS services, and reflective ORB.

3.2.1. Services

Distributed OS services include some standard CORBA services and some 2K specific services.

3.2.1.1.Automatic Configuration

Automatic Configuration Service [14] gathers the knowledge of loading an inert component into the runtime system and defines dynamic dependencies among loaded components. For this purpose, 2K Automatic Configuration Service manages two distinct kinds of dependencies: *prerequisites*, and *dynamic dependencies*.

The prerequisites for a particular inert component specify any special requirement for properly loading, configuring, and executing that component. Prerequisites specify the type and share of hardware resources that a component needs and the software services it requires. The QoS-aware Resource Management Service [6] determines where, how, and when to execute each

component using component's resource requirements. It also uses this data to enable proper admission control, resource negotiation, reservation, and scheduling. Software requirements that are specified in the prerequisites determine which auxiliary components must be loaded and other software services must be located. As of now, the prerequisite specifications are created manually by the 2K OS component developers, which they expect to automate in the future.

The 2K Configuration Service represents their dependencies on other system and application components using CORBA objects called *ComponentConfigurators* as shown in Figure 5. Dependencies are stored as CORBA Interoperable Object References (IORs), pointing to other component configurators, forming a dependence graph of distributed components.

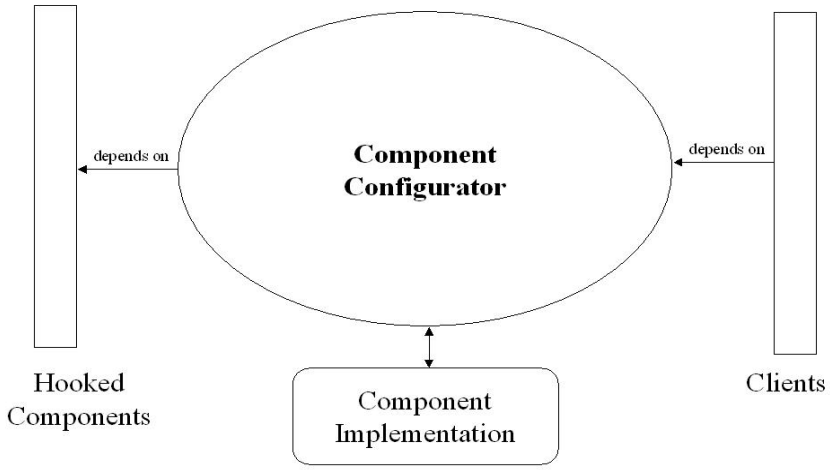

Figure 5: ComponentConfigurator overview

Runtime dependencies of components allow applications running in 2K OS to select different components dynamically to fulfill their needs in different environments and at different times. To optimize performance or to adapt to dynamic changes in the environment, system can manipulate the application dependencies. If a faulty component is encountered, the system replaces it with the new one and informs proper components of its failure by inspecting failed components dependencies. Applications can customize the system by implementing specialized instance of the *ComponentConfigurator* to recover from a failure by replacing the faulty component with a new one.

3.2.1.2.QoS-Aware Distributed Resource Management

QoS-Aware Distributed Resource Management [6] relies on Local Resource Managers (LRM) and Global Resource Management (GRM) in a 2K cluster to manage resources. Each 2K node executes a LRM to export the state and functionality of the hardware resources in that machine to the whole-distributed system through a common IDL interface. The LRMs are also responsible for performing QoS-aware admission control, resource negotiation, reservation, and scheduling of task on a single node.

GRM receives periodic updates from the LRM about its state of local resources and is responsible for maintaining an approximate view of the LAN resource utilization state. The GRM is responsible for QoS-aware load distribution within its LAN. Although not supported now, the 2K intends to combine groups of GRMs hierarchically to provide hardware resource sharing across multiple LANs.

LRMs and GRMs use a CORBA Trader [5] to supply resource discovery services, which allow applications to request resources with certain QoS specifications without knowing the providers of the resource beforehand.

3.2.1.3 Dynamic Security

Dynamic Security service model is based on already established OMG Standard Security Service [5]. This service comprises authentication, access control, auditing, object communication encryption, non-repudiation, and administration of security information. The 2K OS prototype implementation of the CORBA Security Service utilizes the *Cherubim* security framework [15]. Dynamic reconfiguration of the Security Service, facilitating the adoption of situation-specific policies and mechanisms for authentication and encryption, is provided by reflective ORB. Currently, 2K OS implements access control models including Discretionary Access Control (DAC), Double DAC, and Mandatory Access Control (MAC) [16]. Implementation of Role-Based Access Control (RBAC) [17] is still in works by 2K OS team, which will be the basis for security in the large-scale 2K environments.

3.2.1.4.Naming Service

CORBA compliant distributed Name Service is naturally used for 2K OS. This service supplies the symbolic names and maps them into IORs. Through a mechanism known as *Junction*, the 2K Naming Service [18] makes the underlying DNS and POSIX file system names readily available for 2K applications.

3.2.2. Reflective ORBs

Conventional ORBs are static and are not suitable for dynamic system such as 2K. 2K OS team developed a CORBA compliant reflective ORB model based on TAO ORB [4], which allows dynamic configuration. Two such models are considered for 2K: *dynamicTAO* [19], and *LegORB* [20].

DynamicTAO is an extension of the TAO ORB, which enables on-the-fly reconfiguration of the ORB internal engine and of applications running on top of the ORB. DynamicTAO uses ComponentConfigurator to represent the dependence relationships between ORB components and between ORB and application components. DynamicTAO exports an interface that supports transfer of components across the distributed systems, loads and unloads components, and inspects and modifies the configuration of the ORB (and application running on top of it). After experiencing with dynamic ORBs and commercial ORBs, 2K OS team came to conclusion that most applications utilize just a fraction of the services and functionalities offered by ORBs and

they are too large for small devices. A lightweight version of dynamicTAO was developed by 2K OS team call LegORB with minimal functionalities.

LegORB is prime example of how 2K OS is adopting to What You Need Is What You Get (WYNIWYG) philosophy. LegORB can be dynamically customized to adapt to resource availability and to accommodate the requirements of different applications and devices at different moments. A minimal configuration of LegORB that is able to send simple CORBA requests to standard ORBs occupy only around 6Kbytes on a PalmPilot running PalmOS compare to few megabytes required by dynamicTAO. Other than its size, LegORB performs much faster than highly customized commercial ORBs.

3.3. Mobile Configuration Agents

The use of active networking, mobile agents, and push and pull technologies available today to improve performance, provide intelligent resource management, fault tolerance, and reduce bandwidth requirements on distributed and mobile environments. The 2K apply the active networking model [21] to the middleware level, using mobile agents for Code Distribution. Configuration and inspection agents [22] are sent through the reflective ORBs to distribute code. These mobile configuration agents may contain executable code in the form of dynamically loadable libraries or bytecode for interpretation by a virtual machine. Also, they may contain references to network-centric components to be fetched from remote repositories.

Upon receiving an agent, a node processes its content and forwards it to the next node in the distribution network. When an application requests a new component through the ORB reflective interface, the system checks whether there is an appropriate, local version of this component. If one is not available locally, then it fetches its code from a remote repository and dynamically links it to the running ORB. The same mechanism could be followed to upgrade local services that node provides. The combination of these mechanisms provides a flexible infrastructure for automatic software updates.

3.2. Microkernel

2K OS is design to run at the middleware level, above conventional OS like UNIX and Windows. However, users willing to adopt the 2K philosophy in all system layers can get additional benefits by bootstrapping their machines with *Off*++ [9]. The *Off*++ microkernel is an object-oriented redesign of the old *Off* microkernel, which includes new features for architectural awareness and QoS awareness. The *Off*++ distributed adaptable kernel is a minimal kernel whose only task is to safely multiplex and export the distributed hardware present in the network [7]. The *Off*++ extends the functionalities provided by *Off* kernel in order to provide basic support for the 2K OS. 2K OS can run on top of the *Off*++ microkernel as shown in Figure 2.

The *Off*++ microkernel provides low-level architectural awareness for 2K through inspector objects, which are associated with each relevant kernel object. Middleware code can navigate through system components and inspect component properties using a common interface. 2K needs architectural awareness to let its applications know when and how to adapt. More

information about the design and implementation of the *Off*++ microkernel can be found in [8, 9].

4. Performance and Results

Several experiments were conducted to analyze and compare different aspects of 2K Operating System, which include reflective ORB, microkernel, mobile configuration agents, and automatic configuration.

In reflective ORB, calls to co-located servers are found to be as fast as virtual method calls on C++ objects. Performance measurements on CORBA [10] suggest its implementations are efficient and on going research will yield faster implementation.

The *Off*++ microkernel performance was compare with the performance measurements from traditional systems such as the Ultrix monolithic kernel [11] and the Mach microkernel [12]. The result was that *Off*++ microkernel outperforms in all operations except the protection-mode change in which the performance is closely matched with Ultrix.

Dynamic reconfiguration was tested using mobile agents (reconfiguration and inspection agents). System administrators sent Mobile agents into a network of six ORBs running on Sun Ultra-2 machines connected by a 100Mbps Ethernet. The time it took inspection agents to collect the state of the six reflective ORBs and bring it back to the administrator is 101ms. It took 265ms for reconfiguration agent to load a 30Kbytes component and attach the new component to a running application in each of the six ORBs. These numbers can be improved significantly with more tuning and optimization. In addition, experiment on wide-area system composed of nine nodes in different countries was conducted to show the comparison between the performance of 2K mobile configuration agents and a conventional point-to-point approach. The results shows that the round-trip time for mobile configuration agent is significantly less than point-to-point approach as the component size to be uploaded to the target system increases.

Performance of the 2K Automatic Configuration Service was measured by conducting an experiment in which this service is to load components of size 19.2Kbytes. The total time for 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 was less than 100ms for eight components.

Although the unit tests conducted on the performance of the 2K Operating System are encouraging, there need to be more tests on 2K to check for its handling of fault tolerance, QoS, security, etc.

5 Discussion & Conclusion

The main design goal of the 2K Operating System is to facilitate management of dynamic, heterogeneous computing environments for users, administrators, and developers. To achieve this goal, the 2K Operating System should support minimum of following:

- Automatic configuration of components.
- Dynamic reconfiguration of components.
- Data distribution and management between objects at run-time.
- Dynamic resource management.
- Adapt to existing operating systems.
- Support different programming languages.
- Provide secure communication between objects in the network.
- Maintain quality of service (QoS).
- Able to run on small devices such as PDAs, microwave, "high-tech" watches, etc.

To determine if the 2K Operating System has true potential in the 21st century marketplace, lets examine it for above criteria.

The 2K Operating System has automatic configuration service at the middleware to configure system components. Automatic configuration service manages two types of dependency information on components to be loaded on the system dynamically. First, it provides *prerequisite specifications* that specify the needs of each component to properly load, configure, and execute on different hardware. Second, it uses CORBA objects called *ComponentConfigurators* to store dynamic dependencies among loaded components in the system using information in specifications.

Dynamic reconfigurations is required for changing system parameters at runtime, or for replacing components at runtime to fix bugs, to update functionality, or to adapt to changes in the environment. Automatic configuration service uses *pull*-based approach for dynamic reconfigurations and code distribution in the network system. In addition, the 2K Operating System team has developed mobile reconfiguration agents for *push*-based approach to dynamic configuration and code distribution. Mobile reconfiguration agent is based on existing mobile agents technology.

A flexible and adaptable distributed data management system is not support by the 2K Operating System. It is an ongoing research area and the 2K Operating System team plans to incorporate that in the future. For now, data management is left to the individual Operating System on which 2K Operating System runs as a middleware.

The 2K Operating System is able to run as a middleware layer on top of the existing Operating Systems, and it can adapt to any operating system through dynamicTAO and legORBs. The 2K uses CORBA IDL to provide seamless communication between client and server application in different programming languages. The 2K OS utilizes OMG Standard Security Service to provide restricted access to its services. Currently the *Cherubim* security framework [15] is used in the prototype to support dynamic security policies [13]. Dynamic reconfiguration for security service is provided by reflectiveORB. In this prototype, 2K only supports Discretionary Access Control (DAC) and Mandatory Access Control (MAC) models. Research is ongoing in this area to support various other access control models for future implementation including Role-Based Access Control (RBAC).

QoS-aware load distribution and sharing of resources are provided through QoS-Aware Distributed Resource Management Service. It uses Local Resource Managers (LRMs) in conjunction with Global Resource Manager (GRM) to maintain quality of service by providing admission control, resource negotiation, reservation, and scheduling of tasks on a single node and within the cluster.

Finally, the 2K operating system must be able to run on devices with limited memory and limited CPU power. The 2K OS uses "What You Need Is What You Get" (WYNIWYG) approach. This is demonstrated by legORB, which is lightweight model of dynamicORB. DynamicORB has memory footprint of few megabytes or more whereas LegORB that is able to send simple CORBA requests to standard ORBs occupies only around 6Kbytes of memory. Initially LegORB only has minimal operation capability but it dynamically customizes itself on the need basis.

By looking at what all the services that it has to offer and its dynamic approach, the 2K Operating System can be marketable. But there are still many issues to be incorporated in this operating system such as fault tolerance, and data management in distributed environment. Also, there has to be extensive unit level testing in variety of environments and determine if results are efficient for modern applications and future support. Besides this critical issues, one of the major contribution made by the 2K Operating System 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.

6 References

[1] Sun microsystems, Jini Architectural Overview. Technical White Paper. January 1999.

[2] Steve Vinoski. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. IEEE Communications Magazine, Volume: 35 Issue: 2, Page(s): 46 -55, Feb. 1997.

[3] Hank Shiffman. Windows NT and IRIX Interoperability. http://www.sgi.com/developers/feature/1998/inteop.html.

[4] 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.

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

[6] 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.

[7] F. J. Ballesteros, C. Hess, F. Kon, S. Arevalo, and R. H. Campbell. Object Orientation in Off++ - A Distributed Adaptable Microkernel. In *ECCOP'99 Workshop on Object Orientation and Operating Systems*, pages 49-53, Lisbon, June 1999.

[8] Francisco J. Ballesteros, Fabio Kon, and Roy H. Campbell. A Detailed Description of Off++, a Distributed Adaptable Microkernel. Technical Report UIUCDCS-R-2035, University of Illinois at Urbana-Champaign, August 1997.

[9] Francisco J. Ballesteros, Christopher Hess, Fabio Kon, and Roy H. Campbell. The Design and Implementation of the off++ and voff++ μkernels. Technical Report UIUCDCS-R-98-2086, Department of Computer Science, University of Illinois at Urbana-Champaign, March 1999.

[10] I. Pyarali, C. O'Ryan, D. Schmidt, N. Wang, and V. Kachroo. Applying Optimization Principle Patterns to Design Real-Time ORBs. In *Proceedings of the 5th USENIX COOTS*, San Diego, CA, May 1999.

[11] Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr. Exokernel: An operating system architecture for application-level resource management. In *proceedings of the Fiftennth Symposium on Operating Systems Principles*, December 1995.

[12] J. Liedtke. Improving IPC by Kernel Design. In 14th SOSP, pages 175-188, Ashville (NC), 1993. ACM.

[13] Tin Qian. Dynamic Authorization Support in Large Distributed Systems. PhD thesis, Department of Computer Science, University of Illionois at Urbana-Champaign, November 1999.

[14] Fabio Kon, Dulcineia Carvalho, and Roy Campbell. Automatic Configuration in the 2K Operating System. In *Proceedings of the ECOOP'99 Workshop on Object Orientation and Operating Systems*, pages 10-14, Lisbon, June 1999.

[15] Roy Campbell and Tin 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.

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

[17] R. S. Sandu, E. J. Coyne, H. L. Feinstein, and C. E. ouman. Role-based Access Control Models. *IEEE Computer*, 29(2): 38-47, February 1996.

[18] M. Z. Hydari. Design of the 2K Naming Service. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, February 1999.

[19] F. Kon, M. Roman, P.Liu, J. Mao, T. Yamane, L. C. Magalhaes, 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. SpringerVerlag.

[20] M. Roman, 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.

[21] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A Survery of Active Network Research. *IEEE Communications Magazine*, 35(1): 80-86, January 1997.

[22] 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.