

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
Rey Juan Carlos University, Madrid
nemo@gsyc.escet.urjc.es

<http://choices.cs.uiuc.edu/2K>

Report No. UIUCDCS-R-99-2132, UILU-ENG-99-1751

December, 1999

*This research is supported by the National Science Foundation, grants 98-70736 and 99-70139.

2K: A Distributed Operating System for Dynamic Heterogeneous Environments

Abstract

The first decades of the new millennium will witness an explosive growth in the number and diversity of networked devices. We foresee high degrees of mobility, heterogeneity, and interactions among computing devices connected to global networks. Conventional operating system architectures are inadequate for the wide-area, heterogeneous computing environments of the new millennium. 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 JINI, solve part of the heterogeneity problem by permitting seamless communication among different platforms. But, still, they do not address dynamic, distributed resource management and adaptability.

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. We discuss our rationale, describe our prototype implementation, and point to future directions.

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 patches. System administrators spend excessive time just keeping their systems from becoming obsolete; treating the numerous incompatibilities among different software versions is a frequent administrator's nightmare. The second kind refers to dynamic changes in the availability of memory, CPU, network bandwidth and, in mobile systems, connectivity and location. In the latter case, drastic changes may occur in a few seconds, impacting the performance of user applications profoundly.

Unfortunately, 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 work is multiplied by the number of supported platforms. This heterogeneity also brings additional incompatibility problems. Research in interoperable systems and protocols has ameliorated many static heterogeneity problems, but it does not address many of the problems encountered in rapidly changing systems.

This scenario is further aggravated as mobile systems become more common and digital computing becomes ubiquitous. It is then necessary to deal with users accessing the global network from anywhere in the world.

Thus, we need a very flexible and adaptable architecture that permits the dynamic instantiation of customized user environments at different locations in the distributed system. This can be achieved with the help of new component technologies that allow for dynamic instantiation and configuration of component-based systems. However, existing architectures do not provide proper management of the dependencies among system and application components, which makes it difficult to support configuration in a reliable way. Structural and dynamic property changes in a particular component do affect other components. 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 next generation middleware and operating systems.

Previous research has demonstrated that different applications require different system optimizations. One-size-fits-all operating systems are inadequate. Research in customizable operating systems [CIMR93, LTC96], microkernels [Her97] and the exokernel [K⁺97] has shown that one can achieve significant performance improvements by exporting, to the applications, low-level interfaces for simple system services and by building customized high-level abstractions on a per-application basis.

There is now a need to redefine system APIs to represent the needs of users in the new form of distributed environments. Computing in the next millennium will be largely ubiquitous and mobile. Operating systems and middleware will have to provide new services and modify existing ones to adjust to this new reality.

In this paper, we present an overview of *2K*, a novel network-centric operating system that extends the previous work on configurable operating systems, providing an integrated solution to the problems mentioned above, namely, the existence of an integrated environment that supports dynamic changes, heterogeneity, distributed resource management, and customizable system abstractions.

2K provides support for dynamic configuration through a reflective middleware layer based on the CORBA standard. Distributed operating system services are exported to applications through IDL interfaces, extending the ideas that emerged with Spring [M⁺94]. A configuration service takes care of inter-component dependence and assists applications on providing fault-tolerance and adaptation to changes in the environment. Although *2K* can run as middleware on top of traditional operating systems, it can also achieve optimal performance by running as an integrated system with our specially designed microkernel that provides customized system abstractions.

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. Existing operating systems, however, are not able to manage the resources of collections of heterogeneous machines.

CORBA emerges as a powerful standard for interoperability in heterogeneous environments. But, at the present moment, it still lacks the notion of a *user* and does 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 network resources while being able to handle different hardware platforms and different underlying, single-node operating systems.

The *2K* system exports 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.

This design, however, could lead to inefficiencies if we were limited to a static implementation of the CORBA object model. Most existing middleware systems are not flexible; CORBA implementations, for example, are typically limited to TCP/IP-based communication in the form of IIOP [OMG98a]. This is known to be a suboptimal protocol for a wide range of applications such as multimedia streaming [TCC⁺96]. If applications have to build their abstractions on top of a fixed middleware layer, they are forced to use inadequate mechanisms and re-implement the same abstraction at different levels.

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. In *2K*, we combine the use of a dynamically configurable reflective ORB with an adaptable microkernel. This architecture provides all the 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 of components in each system layer. In *2K*, resource management is enhanced with algorithms for Quality of Service (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 Quality of Service is preserved.

In section 3, we discuss how the ideas described in the previous paragraphs are reflected in the architecture of *2K*. In section 4 we present the *2K* microkernel, which provides additional support for customization, dynamic adaptation, and architectural awareness.

3 *2K* System Model

The limitations of current operating systems point to the need for adopting a network-centric approach in operating system design. We now describe how our network-centric model supports code distribution, automatic configuration, distributed resource management, and dynamic security.

3.1 Network-Centrism

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 configured, 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. In other words, the system configures itself automatically and loads the minimum set of components required for executing the user applications in the most efficient way. These components can be downloaded from the network so only a small subset of system services are needed to bootstrap a node, leading to the construction of “network-computers”.

In order to achieve that, *2K* reifies inter-component dependence. System and application components contain an explicit representation of the requirements that must be fulfilled before their execution. A Web Browser, for example, could specify that it depends upon components implementing a Window Manager, a local File Service, the TCP/IP protocol, and a Java Virtual Machine version 1.2 or later.

2K extends the models for representing dependencies among shared libraries in traditional systems. In Solaris [Sun97], for example, the system keeps track of the dependencies among shared libraries that run within one process. *2K* manages dependencies between arbitrary components executing in different processes under different users possibly in different machines.

Inter-component dependence characterizes exactly the software requirements for a service. Thus, the system can download and execute only the minimal set of components that are needed to provide the required services. Moreover, dependencies can be satisfied in alternate ways, leading the way to adaptations to different environments.

This dependence-awareness allows, for example, the configuration of a user desktop environment no matter where the user is and which computer he or she is using. The *2K* middleware can download only the software that is compatible with the underlying operating system and select the version that is optimized for it. In addition, as the reified dependencies are available at runtime, system and application components can process them to take care of mobility and dynamism.

In summary, a careful representation of the dependencies among network-centric components can (1) facilitate system maintenance; (2) provide the basis for automated software updates, (3) enable the implementation of “users” as network-centric entities, and (4) support a WYNIWYG model in which minimal, optimized configurations are assembled.

3.2 Code Distribution

Several research groups are investigating the use of active networking, mobile agents, and push and pull technologies to improve performance, provide intelligent resource management, fault tolerance, and reduce bandwidth requirements on distributed and mobile environments. In particular, by transforming *push* and *pull* [ZF97] into each other as needed, by utilizing combinations of multicast and unicast, and with the aid of mobile agents, we can optimize bandwidth utilization and response time of network-wide operations.

2K applies the active networking model [TSS⁺97] to the middleware level, using mobile agents to distribute code. The reflective ORBs are organized as a distribution network through which system administrators can send *reconfiguration* and *inspection* agents [KCS⁺99]. Agents may contain executable code in the form of dynamically loadable libraries or bytecode for interpretation by a virtual machine. They may also 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. Routing within the distribution network can be based on system state and on data and code included in the agents.

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, it fetches its code from a remote implementation repository and dynamically links it to the running ORB.

Clients may also request the installation of a new service by giving, to the ORB, a reference to a remote component. Using the same code retrieval mechanism, the reflective ORB will then fetch, dynamically link, and execute the code to provide the requested service. Should this new component depend upon other non-available components, the system automatically downloads them from remote repositories and makes sure they are installed.

The combination of these mechanisms provides a flexible infrastructure for automatic software updates. The reification of inter-component dependence lets the system automatically fulfill component requirements, manage multiple component versions, and perform garbage collection, deleting components that are no longer used. 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 Reifying Inter-Component Dependence

To address the problems described in the previous sections, *2K* has a Configuration Service that 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, analyzing the characteristics of each component, and by configuring them in the most efficient way.

Also, if the system knows what the dynamic dependencies among running components are, it can (1) better handle exceptional behavior that could potentially trouble component operation, and (2) support dynamic reconfiguration of large systems by replacing individual components on-the-fly.

3.3.1 Prerequisites

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 (i.e. other components) it requires.

The QoS-aware Distributed Resource Management Service uses the information about the resource requirements to determine where, how, and when to execute each component. The QoS management subsystem [XWN00] uses this data to enable proper admission control, resource negotiation, reservation, and scheduling.

The portion of the prerequisites that specify the software requirements determines which auxiliary components must be loaded and which other software services must be located. This kind of prerequisite is equivalent to the *require* clause in architecture description languages like Darwin [MTK97]. But *2K* uses the CORBA Trader constraint language [OMG98b] to enable the definition of more elaborated specifications.

3.3.2 Dynamic Dependencies

As components are fetched from persistent storage and their code is loaded into the runtime system, the *2K* Configuration Service represents their dependencies on other system and application components using CORBA objects called *ComponentConfigurators* (see figure 1). Dependencies are stored as CORBA Interoperable Object References (IORs) to other component configurators, forming a dependence graph of distributed components.

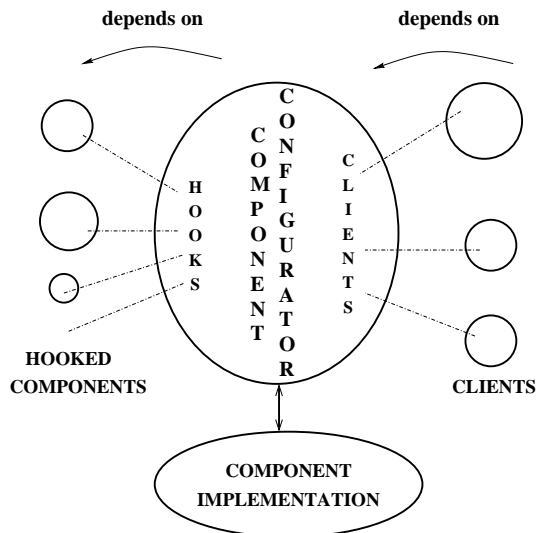


Figure 1: Reification of component dependence.

With information about its runtime dependencies, an application such as the web browser discussed in section 3.1, can refer to its own requirements, selecting different components to fulfill its 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. For example, when a mobile computer is disconnected from a local network, the system can detect that the component implementing wired network communication has failed and that the application has to be automatically redirected to a component implementing wireless communication. The application and the underlying system are aware of their own runtime architecture and are able to manipulate their dependencies to achieve the new configuration.

When a *2K* component fails, the system inspects its dependencies and informs the proper components about the failure. The system may alternatively recover from a failure 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, user access patterns, and the like.

3.3.3 The CORBA Component Model

The CORBA Component Model (CCM) adopted by OMG on November, 1999, specifies a standard framework for building, packaging, and deploying CORBA components [OMG99]. As opposed to the *ComponentConfigurator* model, which focuses on prerequisites and dynamic dependencies, the CORBA Component Model concentrates on defining an XML vocabulary and an extension to the OMG IDL to support the specification of component packaging, customization, and configuration. Thus, we believe that our model and CCM complement each other and could be integrated. CCM provides a static description of component needs and interactions while our model manages the runtime dynamics.

Although CCM was already approved by OMG, publicly available ORBs do not support it yet. Once this happens, we intend to work towards the integration of the two models.

3.4 QoS-Aware Distributed Resource Management

Each *2K* node executes a Local Resource Manager (LRM) whose task is to export the state and functionality of the hardware resources in that machine to the whole distributed system through a common IDL interface. 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 LAN resource utilization state. The GRM then utilizes its information as a hint for performing QoS-aware load distribution within its LAN. Groups of GRMs can be combined hierarchically to provide hardware resource sharing across multiple LANs.

The LRMs are also responsible for performing QoS-aware admission control, resource negotiation, reservation, and scheduling of tasks on a single node. A detailed description of our QoS framework is beyond the scope of this paper and is available in [XWN00].

Resource allocation and deallocation is the responsibility of resource providers which are encapsulated by the LRMs. Applications may still bypass the GRM facilities and request resources directly from the LRM of a certain machine. LRMs and GRMs use a CORBA Trader to supply resource discovery services which allow applications to request resources with certain QoS specifications without knowing the providers of the resources beforehand.

3.5 Naming

The use of CORBA leads naturally to two kinds of names in *2K*: symbolic names and IORs. A CORBA-compliant distributed Name Service supplies the symbolic names and maps them into IORs.

Through a mechanism known as *Junction*, the *2K* Naming Service [Hyd99] makes the underlying DNS and POSIX file system names readily available for *2K* applications. DNS domains and file system directories are mapped to CORBA `NameContexts` transparently. Per-application and per-user name spaces can be defined using a mount table that is kept within application and user profiles.

For efficiency, applications use CORBA IORs as resource names whenever possible. Calls to IOR-named objects within a single protection domain are executed with a performance comparable to a C++ virtual function call (see section 7). But, the use of IORs makes the references still valid across process and machine boundaries.

3.6 Dynamic Security

Access to *2K* hardware and software resources is restricted to controlled CORBA interfaces. For that, we utilize the OMG Standard Security Service [OMG98b] that comprises authentication, access control, auditing, object communication encryption, non-repudiation, and administration of security information.

Our implementation of the CORBA Security Service utilizes the *Cherubim* security framework [CQ98] which is based on active capabilities with support for dynamic security policies [Qia99]. Our 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), Double Discretionary Access Control (DDAC), and Mandatory Access Control (MAC) [SS94a]. We are now extending it to support Role-Based Access Control (RBAC) [Rav96], 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 mobile computer moving from a corporate intranet towards a wireless satellite network. It may be acceptable to use light-weight encryption and soft access control in the intranet but it may be required to apply strong encryption and very tight access control policies when switching to the wireless network.

Although *2K*'s high level of code mobility clearly promotes significant benefits, it also brings additional security concerns. Nevertheless, we minimize this problem by restricting the sources of mobile native code to trusted, authenticated entities and by confining untrusted code to sand boxes, using Java.

4 2K Microkernel

2K is able to run at the middleware level, above conventional operating systems 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++*, a specially customized microkernel that we built using the OSKit [Bry97].

Like Exokernel [K⁺97], the *Off++* microkernel exports hardware resources to user applications and to the middleware. However, as we are dealing with distributed applications, their abstractions may be deployed on a large set of heterogeneous nodes. Thus, exporting raw hardware would force applications to be aware of the specific characteristics of every machine architecture involved.

Unlike Exokernel, that exports a platform-specific interface, *Off++* exports the hardware resources through common, abstract interfaces. For instance, physical memory is exported using an abstract **Seg** interface, which is the same no matter what the machine architecture is. The middleware can ask a **Seg** for its size, its properties (e.g. does it support DMA?), and the memory bank in which it resides.

As opposed to the Exokernel, *Off++* exports *networked* hardware resources to a whole collection of distributed machines. It does not matter whether a resource is operated locally or remotely. In the physical memory example, an application can allocate a collection of remote page frames by asking a remote memory bank to create a **Seg**. As the page frames are allocated, the microkernel furnishes the application with capabilities that enable it to install address translations to those frames.

The microkernel exports and multiplexes the hardware in such a way that physical resources are still valid on remote nodes. The middleware sees the whole network as a source of available resources which creates opportunities for optimizing the implementation of the CORBA middleware. The system, in its turn, uses the local machine as a cache for remote resources, in the spirit of the Cache Kernel [CD94].

4.1 *Off++* architecture

We developed the *Off++* microkernel as an object-oriented, adaptable system. It is organized as a set of (nested) resource containers that export hardware resource units. Most of the kernel code comprises a framework that supplies different resource unit allocators and a hierarchy of resource containers. The kernel is simply a collection of resource containers derived from this framework. It associates physical resources such as page frames, I/O ports, and processor time slots with containers that support both resource allocation and resource operation. For example, **IOBanks** are resource containers that contain **IOPorts**; together they allow the middleware to allocate I/O ports and execute I/O operations.

The kernel interface is simply the set of interfaces implemented by resource containers and their respective resource units.

4.2 Architectural Awareness and Resource Discovery

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. For example, one can navigate a node to locate memory banks, then ask for the value of the “DMA-capable” property, and locate chunks of memory supporting Direct Memory Access).

We made extensive use of the composite pattern [GHJV95] to structure system components, which led to a simple implementation of resource navigation and inspection. One may operate on large containers (even a whole machine) in the same way that one operates on concrete resource units such as page frames.

2K needs architectural awareness to let its applications know *when* to adapt and *how* to adapt. The 2K middleware uses *Off++* architectural awareness to let applications know what resource containers are available in the surroundings and what their functionalities and load are. The 2K Global Resource Manager (see section 3.4) complements the microkernel by giving the user a high-level interface to discover resources in the network.

More information about the design and implementation of the *Off++* microkernel as well as information about its deployment in the *2K* environment can be found in [BKC97, BHKC99].

5 Implementation

We have completed the implementation of various subsystems and frameworks that compose the *2K* distributed operating system. In this section, we describe the completed pieces and discuss ongoing work.

5.1 Dependence Management

We implemented the *ComponentConfigurator* model described in 3.3.2 for single-process applications in C++ and Java, and for distributed applications using CORBA. This framework has been used in a number of systems developed in our research group including the reflective ORB and *2K* distributed services such as the Persistent Object Service [Iye99] and the Automatic Configuration Service [KCC99].

5.2 Reflective ORBs

One of the major constituent elements of *2K* is *dynamicTAO*, a CORBA-compliant reflective ORB. *dynamicTAO* is an open source extension of the TAO ORB [SC99]. It enables on-the-fly reconfiguration of the ORB internal engine and of applications running on top of the ORB. In *dynamicTAO*, we used the *ComponentConfigurator* model described in section 3.3.2 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. A detailed description of the mechanisms for dynamic configuration in *dynamicTAO* can be found in [RKC99]. *dynamicTAO* also contains an infrastructure for building, transmitting, and processing configuration and inspection agents on a network of distributed ORBs [KCS⁺99].

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 heavy-weighted to be used in small devices and embedded systems. Although *dynamicTAO* can be dynamically configurable, its memory footprint is never less than a Megabyte. It would be extremely difficult, if not impossible, to run it on a PDA such as the PalmPilot III. This motivated us to develop a new ORB architecture called *LegORB*. 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. Our goal is to run *LegORB* not only on workstations but also on embedded systems and PDAs. 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 are promising.

5.3 Automatic Configuration

We have recently completed a prototype implementation of the *2K* Automatic Configuration Service [KCC99] using the prerequisite model presented in section 3.3.1. The prototype is based on a skeleton into which different kinds of *prerequisite parsers* and *prerequisite resolvers* can be plugged, allowing for different specification languages and different prerequisite resolution policies. The prototype uses a Simple Prerequisite Description Format (SPDF) that supports the different kinds of prerequisites described in section 3.3.1. Figure 2 shows a typical SPDF description.

:hardware requirements	
machine_type	SPARC
native_os	Solaris
min_ram	5MB
optimal_ram	40MB
cpu_speed	>300MHz
cpu_share	10%
:software requirements	
FileSystem	IR:/sys/storage/DFS1.0 (optional)
TCPNetworking	IR:/sys/networking/BSD-sockets
WindowManager	IR:/sys/WinManagers/simpleWin
JVM	IR:/interp/Java/jvm1.2 (optional)

Figure 2: A Simple Prerequisite Description

In turn, the prerequisite resolver fetches component implementations from remote CORBA implementation repositories and caches them locally. We are currently extending the prototype to support XML specifications and to interact with the CORBA Trader.

5.4 *Off++* Microkernel

After two years of work, we recently completed the implementation of the *Off++* prototype for Intel x86 machines [BHKC99]. Our experience with this prototype led to the design of *Off++v2*, which we are currently implementing. The new version will improve performance by including the device drivers inside the kernel and exporting physical memory segments instead of individual page frames. In addition, it will use a single, simple interface called *box* [BA99] to export all resources.

5.5 Active Spaces

One of the scenarios in which we are applying *2K* is in the context of *active spaces* such as smart rooms. These rooms contain computers, printers, video cameras, projectors, microphones, digital white boards, as well as other kinds of electric and electronic devices. Our idea is to integrate smart rooms, mobile computers, and PDAs into the distributed system by using the *2K* infrastructure together with CORBA and JINI services to support naming, discovery, communication, and QoS-aware resource management in a scalable way.

In our preliminary experiments [HKC⁺99], we “CORBARized” some devices by implementing IDL interfaces to control video cameras, light switches, and even a microwave oven. By using well-defined interfaces and CORBA as a common communication substrate, we were able to integrate all these highly heterogeneous devices into the distributed system and interact with them using both powerful workstations running full CORBA implementations and hand-held PalmPilot computers running our minimal ORB.

6 Lessons Learned

In the past two years of our 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 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 customized microkernel can choose to boot their machines with the *Off++* microkernel.

By using a platform like CORBA, 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. In addition, the use of IDL interfaces among distributed and even co-located system components improves system organization considerably.

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 an application requires a special underlying protocol or a special optimization, the programmer can implement it as a reflective ORB communication module, making it available to other applications with similar needs. In addition, the reflective architecture allows the deployment of these different protocols and optimizations without modifications to the application code.

Configurability must be taken into consideration from the early design stages. Although it is possible to add dynamic configurability later, as we did with *dynamicTAO*, only an architecture like *LegORB* can provide all the flexibility we need.

Finally, we learned that by implementing CORBA interfaces for heterogeneous computers and devices such as video cameras, digital white boards, and microwave ovens, 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 by any other entity present in the network.

7 Performance Considerations

Our reflective ORB is an extension of TAO [SC99], 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++ object¹. The *Off++* environment exploits these optimizations by adopting a vertical integration approach where system services are implemented in libraries. Thus, many *2K* server objects can be invoked with the overhead of a local method call avoiding the need to enter and leave the kernel.

TAO has protocol adaptors which allow specific transports to be used. On native *Off++* environments, that feature allows the use of native portal calls (the *Off++* IPC mechanism) to perform remote CORBA invocations. TAO allows the use of zero-copy schemes, which are readily available on our microkernel.

On a traditional UNIX system (Linux) running on a single 450MHz Pentium II with 256M of RAM, it takes 236 μ s for TAO to perform a cross-domain method invocation with a single parameter [LFGS99]. Applying the *Off++* model, we can optimize this by avoiding the scheduling overhead (using the client thread on the server side).

Contrary to what one could expect, the added support for adaptability and the object-oriented design of the *Off++* microkernel does not compromise the performance of the basic kernel functionality. Table 1 shows the performance of several operations on *Off++*, running on a 100 MHz Intel Pentium-S computer with 64 Mbytes of main memory.

To compare with traditional systems, we scaled the performance numbers obtained in previous experiments with the Ultrix monolithic kernel [EKJ95] and the Mach microkernel [Lie93]. The result was that *Off++* performs better in all operations except the protection-mode change in which the performance is virtually the same as Ultrix. Note that the fastest IPC reported in the literature, that of L4 [Her97], is still one order of magnitude faster than ours. L4 is a non-object-oriented, non-adaptable system implemented in machine language.

The “light call” test measures the time to perform an extremely light-weight system call. In *Off++*, we used a call that returns the number of memory banks in the local node; on Ultrix, we used `getpid()`.

In *Off++*, system calls that simply read part of the kernel state are implemented through user-level libraries – in our model, most of the kernel memory is exported (read-only) to applications. The “r.o.”

¹The general impression that CORBA is big and slow corresponds to first-generation brokers. Recent performance measurements [POS⁺99] suggest that contemporary CORBA implementations are efficient and that even faster implementations will appear.

	light call	r.o.	prot	trap
<i>Off++</i>	7.5	0.7	13.0	44.7
Ultrix (scaled)	11.23	n/a	12.96	58.59

	IPC
<i>Off++</i>	32.8
Mach (scaled)	51.0

Table 1: Performance of *Off++* operations (μs)

column in the table corresponds to the time to issue this kind of (read-only) system call. All the system calls related to architectural awareness (see section 4.2) fall into this category.

The “prot” test corresponds to a protection-mode change for an address translation; “trap” is the time to handle a page protection-fault. “IPC” is the time to perform one-way inter-process communication.

Comparing scaled performance numbers is not enough to have a precise measurement of *Off++*’s performance gains. However, these numbers are enough to show that our microkernel performs reasonably well, especially when taking into consideration that unlike the other kernels we mentioned, *Off++* provides support for the floating-point unit and for SMP (Symmetric Multiprocessing), which introduces extra overhead for locking, synchronization, and context switch. With further tuning and optimizations we will obtain even better performance.

Nevertheless, most potential performance improvements offered by *Off++* come from the control that applications and the *2K* middleware have over expensive operations, and not because of the performance of individual system calls. *Off++* gives the middleware the same degree of freedom that is provided by exokernels. Performance gains due to optimization of kernel primitives are often irrelevant when compared to gains due to the extra control given to the middleware and to applications [K⁺97]. But note that, unlike exokernels, *Off++* allows the use of collections of heterogeneous resources in an object-oriented fashion.

Finally, we measured the performance of our infrastructure for dynamic reconfiguration based on mobile agents (see section 3.2) by sending reconfiguration 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.

For its part, the reconfiguration 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 six ORBs. It took 265 milliseconds, on average, to complete its task and return the results to the administrative tool.

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

8 Related Work

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

Significant research has focused on adaptable operating systems. The term *prerequisite* we adopted was introduced with the SOS operating system [SGH⁺89], but was not significantly explored at that time. Exokernel [K⁺97] and Off [BF97] export hardware interfaces as a means for extensibility. We build on that work by adopting the same model for the network nodes willing to run a new operating system. However, those approaches do not cope with legacy systems nor do they provide a clean model to organize how, when, and what should be adapted.

SPIN [B⁺95] and VINO [SS94b] are adaptable systems which download code into the kernel to allow system extensions. We also build on their work, and employ code downloading (through the network) to install new components into *2K* nodes.

We also borrowed concepts from object-oriented operating systems. Choices and its derivatives [CIMR93] implement operating system services by means of a collection of object-oriented frameworks. *μ*choices [LTC96] allows code-downloading for extensibility. However, neither addresses large-scale, heterogeneous distributed systems.

Spring [M⁺94] is an object-oriented, distributed operating system which also uses clean, IDL-based interfaces for system services. Although the Spring subcontract model [HPM93] supports customization of object invocations, Spring is not as adaptable as *2K* running on *Off++*. The Spring Virtual Memory Manager [KN93], for example, is kept outside the kernel, but it is still a privileged part of the system. Differently from *Off++* applications, Spring applications have no control on how the file server allocates and revokes page frames when accessing application data. When considering a higher-level of abstraction, however, *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.

OMOS [OM92] provided an object meta server on top of Mach, implementing an object-oriented view of originally non-object-oriented components. The OMOS server resolves object names, supplying their code and data, and allows dynamic replacement of system objects. *2K* is similar in that it adds meta-information to keep track of the system structure and deals with mobile code. However, it differs from OMOS because the latter does not include information about inter-component dependence and does not address the question of how to let applications provide their own abstractions efficiently.

Systems like Condor [LLM88] are targeted to High Performance Computing on Clusters of Workstations. They rely on a central resource manager that starts processes on workstations with cycles to burn. Condor uses additional techniques like checkpointing to avoid losing long running jobs due to crashes. Unlike Condor, *2K* uses a hierarchy of resource managers to scale up to the whole Internet.

Globus [FK97] aims to provide a “computational grid” integrating distributing resources in the same system. It also provides scalable resource management based on a hierarchy of resource managers, as we do. But, unlike Globus, *2K* deals with resource interdependencies explicitly using component configurators.

Recent research targeted at using the Internet as “the computer” has lead to systems like Globe [vSHT99], Legion [GW⁺97], and WebOS [VAD⁺98]. 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 as *2K* does. Finally, they do not consider exporting networked resources through well-defined IDL interfaces to support the management of distributed, heterogeneous resources.

Systems based on the research in software architecture [SG96] and dynamic configuration [PCS98] generally assume that the operating system is an omnipresent, monolithic black box that can be left out of the discussion; they concentrate on the architecture of individual applications. We believe that, rather than conflicting with such approaches, our vision complements them by reasoning about all the dependencies that may affect reliability, performance, and quality of service.

The Darwin architectural description language was used with CORBA to specify the overall structure of component-based applications [MTK97]. A Darwin specification defines all the components of an application and the communication interactions between them. At application start time, the middleware loads all the application components and establishes the links between them. This model does not represent dependencies of application components upon system components, other applications, or services available in the distributed environment. Our approach differs from theirs in the sense that, for each individual component, we specify its dependencies on all different kinds of environment components and we maintain and use these dynamic dependencies at runtime.

The Distributed Multimedia Research Group at the Lancaster University has proposed a reflective architecture for next generation middleware [BCRP98, CB99]. They developed a prototype, using the Python interpreted language, in which the programmer is able to inspect and change the implementation at runtime. The level of reflection is much higher than in *dynamicTAO* since, in Python, it is possible to add or remove methods from objects and classes dynamically and even change the class of an object at runtime.

In contrast, our research concentrates on a simpler reflective model, focusing on high performance. In our model, the reflective mechanisms are not included in the normal flow of control, they are only invoked when needed.

One of the major contributions of our work is to combine these important research results using 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. We believe that the mechanisms for automatic configuration in the *2K* operating system point to a bright future for distributed operating systems that will be easy to manage, comfortable to use, and extremely powerful.

9 Conclusions

The role of computers in our lives has changed drastically in the last decade and will continue to change in the first decades of the new millennium. We will witness high degrees of mobility, heterogeneity, and interactions among heterogeneous computing devices connected to global networks. Traditional middleware and operating systems architectures are not prepared to provide efficient resource management in this highly dynamic heterogeneous environments.

In this paper, we presented an integrated operating system architecture for managing distributed heterogeneous resources using recent advances in the technology of software systems. *2K* provides support for dynamic adaptability, configuration of component-based distributed applications, and QoS-aware distributed resource management. Our high-level distributed services are available 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 as well as collaborations with other research groups.

Availability

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

References

- [B⁺95] B. N. Bershad et al. Extensibility, Safety and Performance in the SPIN Operating System. In *Proc. of the 15th SOSP*. ACM, December 1995.
- [BA99] Francisco J. Ballesteros and Sergio Arévalo. The Box: A Replacement for Files. In *IEEE 7th Workshop on Hot Topics in Operating Systems (HotOS-VII)*, Rio Rico, Arizona, March 1999.
- [BCRP98] Gordon Blair, Geoff Coulson, Philippe Robin, and Michael Papathomas. An Architecture for Next Generation Middleware . In *Proceedings of Middleware '98*, Lake District, England, November 1998.
- [BF97] Francisco J. Ballesteros and Luis L. Fernández. The Network Hardware is the Operating System. In *Proceedings of the 6th Hot Topics on Operating Systems (HotOS-VI)*, Cape Cod, MA, May 1997.
- [BHKC99] Francisco J. Ballesteros, Christopher Hess, Fabio Kon, and Roy H. Campbell. The Design and Implementation of the Off++ and vOff++ μ kernels. Technical Report UIUCDCS-R-98-2086, Department of Computer Science, University of Illinois at Urbana-Champaign, March 1999.

- [BKC97] Francisco J. Ballesteros, Fabio Kon, and Roy H. Campbell. A Detailed Description of Off++, a Distributed Adaptable Microkernel. Technical Report UIUCDCS-R-97-2035, University of Illinois at Urbana-Champaign, August 1997. Also available at <http://choices.cs.uiuc.edu/2k/off++>.
- [Bry97] Bryan Ford and Godmar Back and Greg Benson and Jay Lepreau and Albert Lin and Olin Shivers. The Flux OSKit: A Substrate for Kernel and Language Research. In *Proceedings of the Sixteenth Symposium on Operating Systems Principles*, Saint Malo, France, October 1997. ACM.
- [CB99] Fabio Costa and Gordon Blair. A Reflective Architecture for Middleware: Design and Implementation. In *Proceedings of the ECOOP'99 Workshop for PhD Students in Object Oriented Systems*, Lisbon, June 1999.
- [CD94] D. Cheriton and K. Duda. A caching model of operating system kernel functionality. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pages 179–193, November 1994.
- [CIMR93] Roy Campbell, Nayeem Islam, Peter Madany, and David Raila. Designing and Implementing Choices: an Object-Oriented System in C++. *Communications of the ACM*, 36(9):117–136, September 1993.
- [CQ98] 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.
- [EKJ95] 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 Fifteenth Symposium on Operating Systems Principles*, December 1995.
- [FK97] I. Foster and C. Kesselman. The Globus Project: A Status Report. *International Journal of Supercomputer Applications*, 11(2):115–118, 1997.
- [GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. Elements of Object-Oriented Software*. Addison-Wesley, 1995.
- [GW⁺97] Andrew S. Grimshaw, Wm. A. Wulf, et al. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1), January 1997.
- [Her97] Hermann Härtig and Michael Hohmuth and Jochen Liedtke and Sebastian Schönberg and Jean Wolter. The Performance of μ -Kernel-Based Systems. In *Proceedings of the 16th Symposium on Operating Systems Principles*, Saint Malo, France, October 1997. ACM.
- [HKC⁺99] Christopher K. Hess, Fabio Kon, Roy H. Campbell, Manuel Román, Dulcinea Carvalho, and Luiz Magalhães. Dynamic Resource Management for Smart Environments: The 2K Approach. In *Inter-agency Workshop on Smart Environments*, Atlanta, Georgia, July 25-26 1999. Georgia Institute of Technology.
- [HPM93] G. Hamilton, M. L. Powell, and J. J. Mitchell. Subcontract: A Flexible Base for Distributed Programming. *Proceedings of the 14th Symposium on Operating Systems Principles*, pages 69–79, Dec 1993.
- [Hyd99] Muhammad Ziauddin Hydari. Design of the 2K Naming Service. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, February 1999.
- [Iye99] Arjun Chandrasekar Iyer. Persistent Object Service Framework Using Component Configuration Model. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, July 1999.

- [K⁺97] M. Frans Kaashoek et al. Application Performance and Flexibility on Exokernel Systems. In *Proc. 16th SOSP*, Saint Malo, France, October 1997. ACM.
- [KCC99] Fabio Kon, Dulcinea 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.
- [KCS⁺99] Fabio Kon, Roy H. Campbell, Balaji Srinivasan, Ramesh Chandra, and Arun Viswanathan. Dynamic Reconfiguration of Scalable Internet Systems with Mobile Agents. Technical Report UIUCDCS-R-99-2105, Department of Computer Science, University of Illinois at Urbana-Champaign, March 1999.
- [KN93] Yousef A. Khalidi and Michael N. Nelson. The Spring Virtual Memory System. Technical report, Sun Microsystems Laboratories Inc., 2550 Garcia Avenue – Mountain View, CA 94043, Feb 1993.
- [LFGS99] David L. Levine, Sergio Flores-Gaitan, and Douglas 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.
- [Lie93] J. Liedtke. Improving IPC by Kernel Design. In *14th SOSP*, pages 175–188, Ashville (NC), 1993. ACM.
- [LLM88] 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.
- [LTC96] W. S. Liao, S. Tan, and R. H. Campbell. Fine-grained, Dynamic User Customization of Operating Systems. In *Proc. 5th Int. Workshop on Object-Oriented in Operating Systems*, pages 62–66, Seattle, October 1996.
- [M⁺94] J. Mitchell et al. An Overview of the Spring System. In *Proceedings of Compcon 'Spring 1994*, February 1994.
- [MTK97] Jeff Magee, Andrew Tseng, and Jeff Kramer. Composing Distributed Objects in CORBA. In *Proceeding of the 3rd International Symposium on Autonomous Decentralized Systems (ISADS'97)*, Berlin, April 1997.
- [OM92] Douglas B. Orr and Robert Mecklenburg. OMOS - An Object Server for Program Execution. In *International Workshop on Object Oriented Operating Systems*, Paris, 1992. IEEE.
- [OMG98a] OMG. *CORBA v2.2 Specification*. Object Management Group, Framingham, MA, February 1998. OMG Document 98-07-01.
- [OMG98b] OMG. *CORBAservices: Common Object Services Specification*. Object Management Group, Framingham, MA, 1998. OMG Document 98-12-09.
- [OMG99] OMG. *CORBA Components*. Object Management Group, Framingham, MA, 1999. OMG Document orbos/99-07-01.
- [PCS98] Jim Purtilo, Robert Cole, and Rick Schlichting, editors. *Fourth International Conference on Configurable Distributed Systems*. IEEE, May 1998.
- [POS⁺99] Irfan Pyarali, Carlos O’Ryan, Douglas Schmidt, Nanbor Wang, and Vishal Kachroo. Applying Optimization Principle Patterns to Design Real-Time ORBs. In *Proceedings of the 5th USENIX COOTS*, San Diego, CA, May 1999.
- [Qia99] Tin Qian. *Dynamic Authorization Support in Large Distributed Systems*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, November 1999.

- [Rav96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.
- [RKC99] Manuel Román, Fabio Kon, and Roy H. Campbell. Design and Implementation of Runtime Reflection in Communication Middleware: the *dynamicTAO* Case. In *Proceedings of the ICDCS'99 Workshop on Middleware*, pages 122–127, Austin, TX, June 1999.
- [SC99] Douglas C. Schmidt and Chris Cleeland. Applying Patterns to Develop Extensible ORB Middleware. *IEEE Communications Magazine Special Issue on Design Patterns*, 1999.
- [SG96] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
- [SGH⁺89] Marc Shapiro, Yvon Gourhant, Sabine Habert, Laurence Mosseri, Michel Ruffin, and Céline Valot. SOS: An Object-Oriented Operating System — Assessment and Perspectives. *Computing Systems*, 2(4):287–338, December 1989.
- [SS94a] Ravi S. Sandu and Pierangela Samarati. Access Control: Principles and Practice. *IEEE Communications Magazine*, 32(9):40–48, September 1994.
- [SS94b] Christopher Small and Margo Seltezer. Vino: An integrated platform for operating system and database research. Technical report, Computer Science Laboratory, Harvard University, Cambridge, MA 02138, 1994.
- [Sun97] Sun Microsystems. *Linker and Libraries*, 1997. On-line document available at <http://docs.sun.com>.
- [TCC⁺96] S. Tan, R. Campbell, Z. Chen, W. Liao, D. K. Raila, F. Kon, and M. Valdez. Adaptation and Synchronization in Low-Bandwidth Internet Video. In *World Wide Web Consortium Workshop on Real Time Multimedia and the Web (RTMW '96)*, INRIA Sophia Antipolis, France, October 1996.
- [TSS⁺97] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.
- [VAD⁺98] Amin Vahdat, Thomas Anderson, Michael Dahlin, David Culler, Eshwar Belani, Paul Eastham, and Chad Yoshikawa. WebOS: Operating System Services For Wide Area Applications. In *Proceedings of the Seventh Symposium on High Performance Distributed Computing*, July 1998.
- [vSHT99] Maarten van Steen, Philip Homburg, and Andrew S. Tanenbaum. Globe: A Wide-Area Distributed System. *IEEE Concurrency*, 7(1):70–78, January 1999.
- [XWN00] Dongyan Xu, Duangdao Wichadakul, and Klara Nahrstedt. Multimedia Service Configuration and Reservation in Heterogeneous Environments. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS'2000)*, Taipei, Taiwan, April 2000.
- [ZF97] Stanley Zdonik and Michael Franklin. A Framework for Scalable Dissemination Based Systems. In *Proceedings of the 12th OOPSLA*, Sigplan Notices:32(10), pages 94–105, October 1997.