The Eden Project: A Final Report

Andrew P. Black, Edward D. Lazowska,
Jerre D. Noe and Jan Sanislo

Department of Computer Science
University of Washington
Seattle, WA 98195

Technical Report 86-11-01

The Eden Project: A Final Report

Andrew P. Black, Edward D. Lazowska,
Jerre D. Noe and Jan Sanislo

Department of Computer Science
University of Washington
Seattle, WA 98195

Technical Report 86-11-01

The Eden Project: A Final Report

Andrew P. Black, Edward D. Lazowska,
Jerre D. Noe and Jan Sanislo

Department of Computer Science
University of Washington
Seattle, WA 98195

Abstract

The University of Washington's Eden Project was funded in September 1980 by the first award in the National Science Foundation's Coordinated Experimental Research Program. This is the final report of that project.

Eden, like most CER projects, had two objectives. The first objective was technical: to carry out a specific experimental research project concerned with designing, building and using an "integrated distributed" computing system. (Eden was considerably more focused in this regard than most of its successors.) The second objective was environmental: to strengthen Washington's Computer Science department, and to transform it into one where experimental techniques are routinely employed.

The structure of this report reflects this duality. After an introduction, we provide an overview of the system: its architecture, its implementation, its programming environment, its applications, and a technical assessment. Then in two brief sections we outline the project's technical impact and its influence on the environment.

This work was supported in part by the National Science Foundation under Grants MCS-8004111 and DCR-8420945. Computing equipment was provided in part under a cooperative research agreement with Digital Equipment Corporation.

1

Introduction

The University of Washington's Eden Project was funded in September 1980 as the first award in the National Science Foundation's Coordinated Experimental Research Program.

Eden, first and foremost, was a five year experiment in designing, building and using an "integrated distributed" computing system [7, 16, 19, 36]. Eden attempted to combine the benefits of integration with those of distribution by supporting an object-based style of programming across a local area network. The system was integrated because operations could be performed on any object anywhere in the network, without any need to know the location of the object. The hypothesis of Eden was that this environment would be conducive to building distributed applications.

To test this hypothesis, a prototype Eden system (actually, a series of three different prototypes) and a variety of distributed applications were constructed. Roughly 100 different Eden object types existed at the official close of the project, comprising some 300,000 lines of Eden Programming Language code; more object types have been written subsequently. In addition to this work on the Eden system itself, the project proved to be the catalyst for a great deal of other experimental research in distributed systems. Some of this work, such as research on load sharing in distributed systems [9,22,23,35] and on replication, transactions and concurrency control [32,45-48,51,52], used the Eden system as a laboratory and could not have taken place without its presence. Other work, such as the Emerald language [18,20,29,34] and a variety of theoretical studies on distributed systems [24-26,41], was motivated by the experience of designing, building and using Eden, and would not have happened without that context.

Although Eden differed from most subsequent CER grants in that its emphasis was on supporting a focused research effort rather than on providing general departmental infrastructure, to a large extent it succeeded in doing both. Eden represented a collective decision by certain members of the department to pursue a new, more experimental research direction, and an important effect of the project has been to substantially increase the experimental focus and capabilities of the department.

The structure of this report reflects the multiple facets of the project. Section 2 is a technical overview of the Eden system and some of its applications. Section 3 discusses the technical impact of the project; substantial contributions to the field have been made by the Eden system, the Eden Programming Language and various Eden applications, as well as by various research projects that used Eden as an experimental laboratory. Section 4 highlights the effect of the Eden project on the general research environment at the University of Washington.

2

System Overview

The Eden system attempts to combine the benefits of integration and distribution by supporting an object-based style of programming on a number of "node machines" interconnected by a local network. We believed that this environment would simplify the task of building distributed applications. At the scientific level, the goal of the project was to test this belief experimentally; to do so involved building a prototype Eden system and implementing a variety of applications on it. This section of the report presents a technical overview of the Eden system and glimpses of some of the applications. It also describes some of our experiences as builders of a large distributed system. A more complete but less current introduction to the structure of Eden will be found in reference 7.

2.1. The Eden Distributed System

Eden represents a merging of three distinct threads in operating system design. First, Eden is a state-of-the-art object-oriented system. Viewed in this way, Eden is a descendant of Hydra [70]. Second, Eden is a complete distributed operating system. In this sense it is rather like the Apollo DOMAIN [63] system or UCLA's LOCUS system [69]. Third, Eden includes a full-scale implementation of Remote Procedure Call (RPC). In this sense it is rather like the RPC system pioneered by Xerox [61]. Eden thus has something in common with a number of the advanced operating system projects of the last few years. However, in combining the advances of these systems, Eden provides a unique set of facilities. It is distinguished from systems such as LOCUS in being based on a contemporary object-oriented model. It is distinguished from the contemporary implementation of the Apollo DOMAIN system by having a notion of object that is definable and extensible by the user. It is distinguished from the Xerox system by the fact that objects are mobile and that the binding of a client to a server is performed upon every invocation rather than just once. And of course it is a significant advance over the Hydra system in that distribution is an integral part of Eden.

It is important to observe that Eden is not a collection of facilities provided on top of an existing operating system in an attempt to add distribution to a conventional style of computing. This is true despite the fact that the current prototype implementation of Eden is built using the facilities of UNIX.® UNIX is merely an implementation vehicle; Eden itself provides the user with a complete, advanced environment for the development and execution of distributed applications.

Eden objects, the basic building blocks of distributed applications in Eden, exhibit the following characteristics:

- *Invocation* is the means whereby one object obtains service from another. It may be thought of as a request message followed some time later by a response message.
- Invocation is *location independent*; one object does not need to know the location of another object in order to invoke it.
- Objects are addressed by *capabilities*. Capabilities are not addresses. Rather, each object has an unique identifier. A capability consists of that unique identifier and a set of rights. The problem of locating an object given only its capability is handled by the system itself. Capabilities are protected from forgery by the system.
- Objects are *mobile*.
- Objects are *autonomous*. Each object has one or more processes within it. This stands in contrast to Smalltalk objects, where threads of control enter an object when a request is made but leave it when the response is completed. Eden objects can perform activities on their own behalf, as well as in response to invocations.

This section is a revision of an invited paper presented at the European UNIX system Users Group Conference, Manchester, England [19].

® UNIX is a registered trademark of AT&T.

- Each object has a *concrete Edentype*, which may be regarded as a description of the state machine that defines the behavior of the object, i.e., those invocations that it will accept and what their effect will be. In implementation terms the concrete Edentype is a piece of code in the Eden Programming Language.
- Each object has a *data part*, which includes long-term state representing the data encapsulated by the object and short-term state consisting of the local data of invocations currently in progress, hints, caches, and so on.
- An object may *checkpoint*. This is an atomic way of writing its state to stable storage. The data is written under the control of the concrete Edentype of the object. Typically all of the long-term state will be written, and as much of the short-term state as is necessary to achieve the reliability specification of the object.
- Finally, objects are *activated automatically* when they are invoked, if this proves to be necessary. Conceptually we wish to regard objects as active at all times. Should they fail or be *passivated* to conserve resources, activation will take place from the checkpointed representation.

These characteristics of Eden objects lend the following unique characteristics to the Eden system as a whole:

- First, it is an integrated system with a single uniform system-wide namespace. Its space of objects is managed by the system, in the sense that the system takes care of obtaining resources for the creation of a new object and for garbage collecting objects when they are no longer accessible.
- Eden supports the notion of abstract Edentypes. Several different pieces of concrete code can implement services that at some level of abstraction may be considered as identical. For example, there may be two concrete directory types which support the same set of operations but which exhibit different reliability and performance characteristics. When invoking a directory the invoker need not be concerned with which concrete type is actually used. This simple idea permits us to support multiple inheritance hierarchies in the sense of Smalltalk and the asymmetric stream concept, and is discussed further in section 1.5.
- The third consequence is that data encapsulation (information hiding) is supported and enforced by the system. The only code that accesses the representation of an object is the code that makes up that object's concrete Edentype. If the object's data structure is found to violate its invariants then only the object's own code need be examined to find out why. Similarly, if a change in use requires that a data structure be modified, all the code that needs to be modified is within the object itself.
- Fourthly, objects are secure. The system ensures that only through possession of a capability can an object be accessed. The system also ensures that capabilities cannot be forged. The access rights in a capability enable us to provide fine-grained restrictions on access.
- Another consequence of our design is that Eden does not offer automatic insulation from crashes. A general-purpose atomic action system is not one of the primitives that Eden provides. We *do* provide the atomic checkpoint primitive, whereby programmers who so desire can build robust and secure applications. This is a different approach from that taken by, for example, the Argus system [64, 65] and the Clouds system [59], where atomic actions are among the basic building blocks provided at the system level. In Eden, it is possible to experiment with different approaches to providing transactions [51].

2.2. Implementation

The Eden system has been operating on a collection of VAX systems since April 1983 and on a collection of Sun workstations since September 1984. The Eden implementation is in two parts: the Eden Programming Language, and the Eden kernel. Eden coexists with UNIX, in the sense that an individual can make simultaneous use of UNIX and Eden services. This coexistence was crucial in minimizing the software effort required to make Eden usable, and was the main motivation for our choice of prototyping environment.

The set of UNIX facilities that Eden uses is small: processes and address spaces, a minimal flat file system, and the ability to load code into those address spaces from a file. We attempted to minimize the changes made to the UNIX kernel. Starting with 4.1 bsd UNIX, we added an Ethernet driver and an inter-process communication mechanism, and decreased the granularity of the timer; other changes were limited to system parameters.

Each Eden object is implemented as a UNIX process. The Eden kernel operates as an additional UNIX process on each node; this process is called an Eden host. Both the host and the object processes operate in "user-mode", and do not require any special privileges. The rôle of the host is to create object processes, to maintain part of an object's state, to maintain caches of object locations, passive representations and code, and of course to implement the set of system calls that characterize Eden, including the calls that send an invocation, create an object, and checkpoint a passive representation.

In addition to the host, every node with a disk runs a second kernel process called a POD, for *Permanent Object Database*. The POD's function is to manage the passive representations of those objects that have checkpointed onto its disk. When a checkpoint occurs, the POD arranges that the old passive representation is replaced atomically by the new data; this may involve communicating with other PODs if the checksite has moved. The POD also manages the executable code that makes up an Edentype: this code is simply the passive representation of another object of type *Typestore*.

When one object wishes to invoke another, it calls the kernel-provided *AsynchInvoke* primitive, passing the capability of the target object and the appropriate data as parameters. If the target object happens to be located on the same node as the invoker, the kernel will discover this by examining its tables, and will deliver the invocation message directly. To reply to the invocation, the target makes the *ReplyMsg* kernel call, and the kernel routes the reply message to the invoker. If the information in the kernel's location tables indicate that the target object is on another node, the kernel will send the invocation message to the kernel process on that node, which will deliver it to the target object. If there is no entry in the kernel's tables for the target object, the kernel engages in a multi-layered *location protocol*, which will eventually return with either the location of the active form of the target object, or with an indication that the object is not active but that its passive form is located on a particular POD. In the latter case, the object is automatically activated on an appropriate host, to which the POD makes the executable code and the passive representation available. Once the target object is actived, invocation proceeds as normal.

2.3. The Eden Programming Language

A major achievement of the Eden project has been the design and implementation of the Eden Programming Language. EPL is based on Concurrent Euclid, a Pascal extension providing processes, modules and monitors; it provides direct support for the fundamental abstractions of Eden, that is, capabilities and invocation. Capabilities are first-class citizens, even to the extent of having source-language denotations. Syntax exists both for sending and receiving invocations, making invocations as easy to use as conventional procedure calls. We feel that this has been a key factor contributing to the ease of use of Eden.

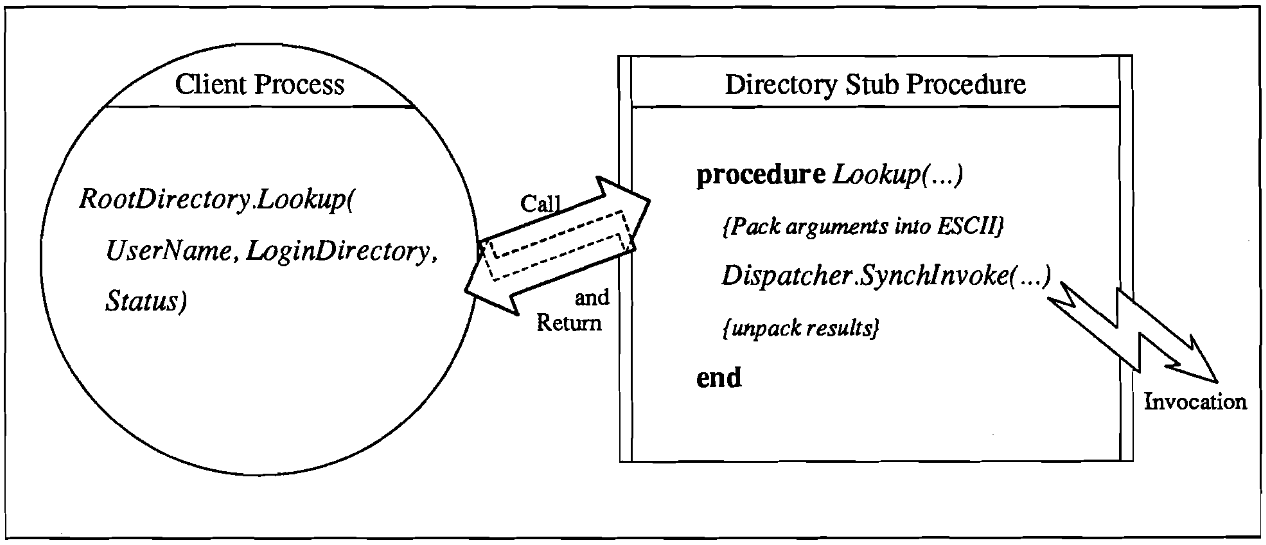


Figure 2.1: Sending a *Lookup* Invocation

The significance of EPL is not that it extends the state of the art of language design, but that there is a careful match between the concepts of Eden and the structures of EPL. The Eden invocation is simple in concept; the challenge was to make its realization in EPL equally simple. On the invoking side, the programmer sees an ordinary procedure call with an additional *status* parameter (see Figure 2.1). On the invoked side, the programmer writes a procedure body, again quite ordinary except that it is designated an *invocation procedure* and certain parameters must be present. In addition, some process on the invoked side must receive the invocation and call the invocation

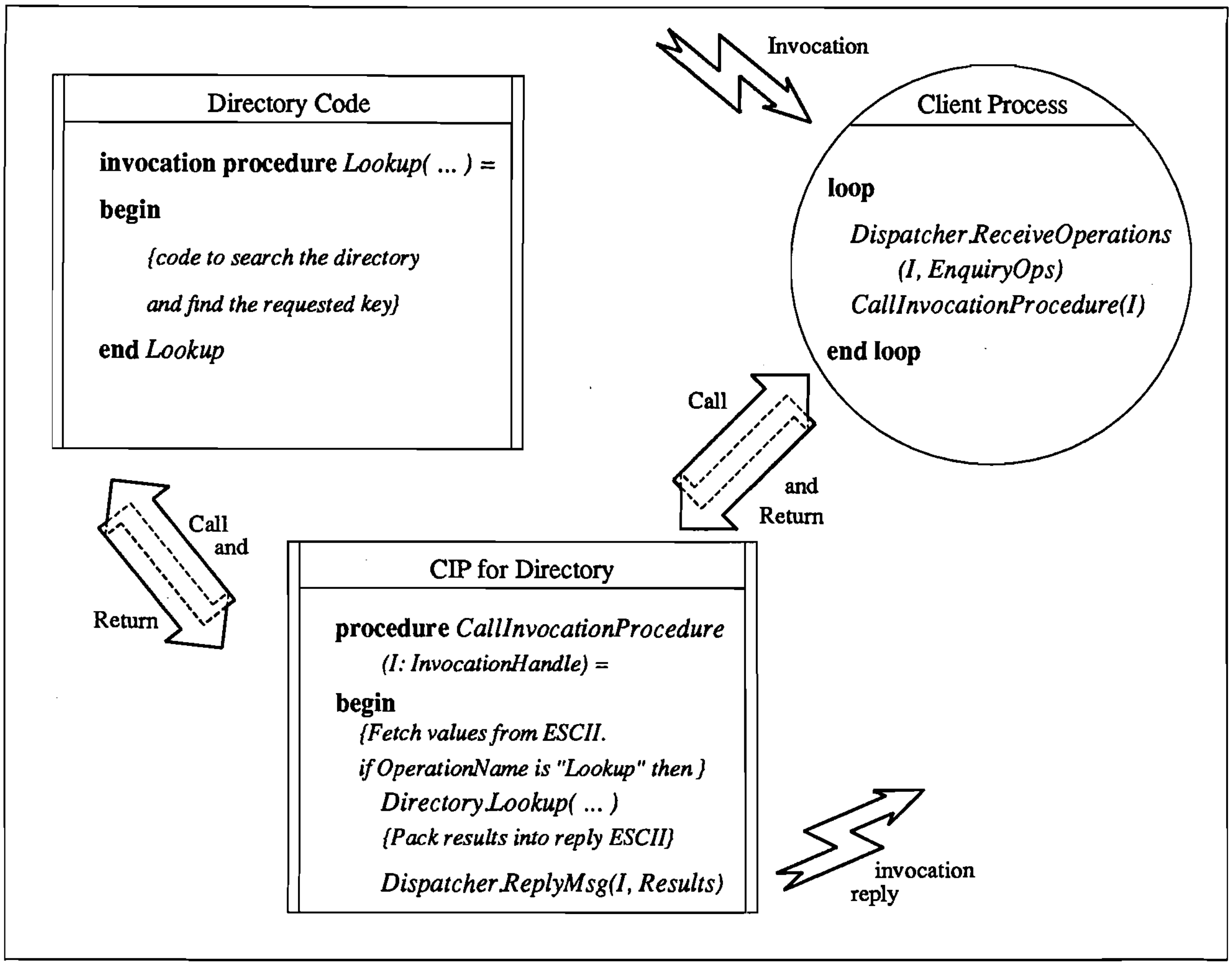

Invocation
Directory Code
invocation procedure Lookup(...) =
begin
{code to search the directory
and find the requested key}
end Lookup
Client Process
loop
Dispatcher.ReceiveOperations
(I, EnquiryOps)
CallInvocationProcedure(I)
end loop
Call
and
Return
Call
and
Return
CIP for Directory
procedure CallInvocationProcedure
(I: InvocationHandle) =
begin
{Fetch values from ESCII.
if OperationName is "Lookup" then}
Directory.Lookup(...)
{Pack results into reply ESCII}
Dispatcher.ReplyMsg(I, Results)
invocation
reply

Figure 2.2: Receiving a *Lookup* Invocation

procedure. The declaration of the invocation procedure is a way of stating that the object is willing to handle a particular invocation; it also defines the parameter list for that invocation, and thus provides the information needed to perform type-checking.

The figures illustrate how invocation support is implemented. The invoking routine actually calls a stub procedure (in the rectangular box, Figure 2.1), which has been generated by a program from a description of the invocation interface. The receipt of an invocation is shown in Figure 2.2. The invocation is received by a user-written process (in the circle in Figure 2.2), but typically the only action of this process is to call the automatically generated *CallInvocationProcedure* which unpacks the arguments, calls the appropriate procedure in the target object, and packages up and sends the results.

The other main contribution of EPL is the provision of intra-object concurrency. A run-time kernel provides multiple light-weight processes and monitors within the UNIX process that supports the object. Thus, when a client process makes a remote invocation, it is possible to suspend just that process pending the receipt of a response; other client processes are free to continue. It is therefore possible to write a library module that implements the abstraction of synchronous communication, using the *AsynchInvoke* kernel call and EPL processes and monitors. One such module, the *Dispatcher*, is the standard way of sending and receiving invocations in Eden; programmers prefer not to use the asynchronous primitives, even though they are available. The matter of synchronous *vs.* asynchronous communication and the availability of concurrency is discussed in greater depth in reference 16.

2.4. Some Eden Applications

Each application that has been built in Eden has three objectives, although the relative emphasis on these objectives differs among the applications:

- *to evaluate the system*

  The value of the Eden architecture, i.e., the hospitability of the system for distributed applications, can only be assessed through use.

- *to make Eden a "complete" system rather than a "kernel"*

  In their evaluation of Cal-TSS, Lampson and Sturgis note that a kernel constitutes perhaps ten per cent of an operating system [62]. A key advantage of the object-based approach used by Eden is that many traditional operating system components can be built as applications.

- *to conduct research into the applications themselves*

  The Eden system provides a laboratory for exploring the design and structure of distributed applications.

To consider a few examples: the Eden Mail System (Edmas) [6], our first application, was written largely with the first objective in mind – testing the support that the Eden system provides for programming distributed applications. It was not the intention to interconnect Edmas with other mail systems to provide a general mail utility, nor was it the intention to study mail systems in general (although considerably more has been learned about this than was anticipated). The same is true of the basic file system (a subset of the system described in reference 32).

The reader may be surprised to see us refer to the file system as an application, but conventional sequential files are not part of the Eden kernel. A sequential file is simply an object that allows its contents to be read and written using the Eden transput (input/output) system. The *Checkpoint* facility allows such an object to maintain its state on the disk, and thus to exist permanently. However, sequential files are not the only form of long-term storage in Eden. All objects, once checkpointed, are equally permanent. A means is required for keeping capabilities for them; the Edentype *Directory* does just this. A Capability for any object can be associated with a mnemonic string and stored in a directory, and can later be retrieved by performing a *Lookup* operation on the directory with the string as argument. Since Directories can be entered into other directories, an arbitrary directed graph of Directories can be built up from a single root.

The Eden Calendar System [27] was written largely with the first and third objectives in mind: to further evaluate the Eden system's hospitability, and to explore the use of transactions in multi-user calendar systems. The Eden Terminal Handler and the Eden Command Language Interpreter were built primarily because they were necessary for the use of the system – the second objective. Finally, moving the translator for the Eden Programming Language into Eden contributed to all three objectives: the translator exercises virtually every aspect of the system, while exploiting the Eden architecture to solve specific practical problems.

Eden has been and continues to be used as a basis for research in distribution, replication, and concurrency control. An experimental version of Eden provides replicated passive representations, and used a voting scheme to keep them consistent in the face of failures [45,48]. Crash-resistant resources have been implemented on top of the normal Eden system by using multiple objects to represent a single logical resource; to update the resource despite some of the copies being unavailable, we ***regenerate*** the inaccessible copies elsewhere in the network [52]. We have also implemented a general purpose nested transaction mechanism out of Eden objects; each transaction is characterized by its own transaction manager object that is responsible for the concurrency control and crash recovery of its sub-transactions [51].

To give the flavor of the way applications are constructed in Eden, we will briefly review the design of the Eden Calendar System. More details can be found in reference 27. The Calendar system is designed to serve both as a personal appointment book, in which one can schedule, list and cancel appointments, and as a shared calendar system that assists in scheduling events with one or more other users, while guaranteeing consistency of the calendars and detecting conflicts. We will first discuss some design considerations, and then the implementation in terms of objects. The way that the objects interact to achieve atomicity will then be described; although this is atypical of Eden applications (most do not have very stringent atomicity requirements), it is interesting to see how transactions can be built that take advantage of the semantics of the particular operations available on the objects concerned.

A fundamental feature of such a system is that information about appointments needs to be shared by all of the participants. There are two obvious ways of doing this: centralising the information in one place, and letting each

Representation

Start Time:	Monday, 22nd Sept 1986 at 13:30
End Time:	Monday, 22nd Sept 1986 at 15:00
Description:	EUUG Conference Session
Event Status:	Definitely Scheduled
Participants:	Black <capability> ; Donner <capability> ; Harper<capability>

Operations

AppendParticipant()
CancelEvent()
ConfirmEvent()
GetNextParticipant()
LookupParticipant()
RejectEvent()
ScheduleEvent()
SetDescription()
Show()

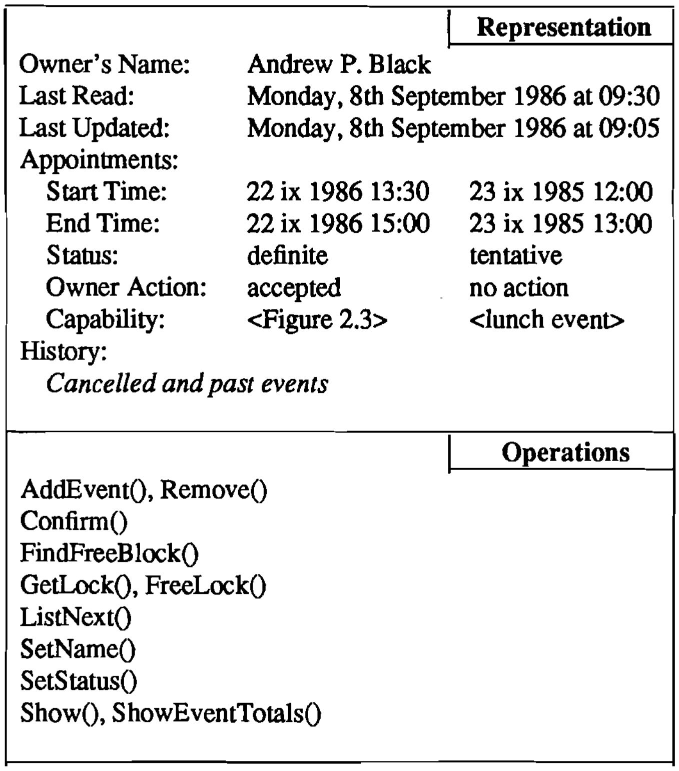
Figure 2.3: An Event Object

participant refer to it, or replicating the data so that each participant has a copy. Centralisation has the advantage that consistency is assured, but performance and availability may suffer. Replication of the data means that one has to take special care to ensure consistency; this also impacts performance, and may also adversely affect availability, depending on the techniques used. The compromise reached in the calendar system is partial replication. The system contains two basic types of Eden objects: *Events* and *Calendars*. Events are the repositories of the true information about an engagement; Calendars contain hints and caches to improve performance, and use a transaction mechanism to ensure consistency where necessary.

Figures 2.3 and 2.4 illustrate typical Event and Calendar objects. They may be thought of as abstract machines that encapsulate the relevant data representation and provides access to it through a set of operations. The representation of the event is a record structure containing the relevant data; note that the list of participants is represented as a list of capabilities for their calendars. The given operations can be applied to the event to add participants, interrogate the data, and so on. The *ScheduleEvent* operation provides automatic scheduling; it takes as arguments an interval of time and a duration, and attempts to find a free slot on all of the participants' calendars of the required duration within the interval. The calendar object (Figure 2.4) has a similar structure; the appointments field is represented as a list of events. As well as Capabilities for event objects, some of the information about each event is replicated in each participant's calendar.

Scheduling an engagement is a human-time operation that may take days or weeks: it cannot be completed until the last participant has examined his calendar and agreed to the tentatively scheduled meeting – and that person may be on holiday. Because most transaction techniques work well only when locks are held for brief time-spans, scheduling is broken into two parts. First the event is tentatively scheduled, which is done at machine speed; confirmations are then obtained from each participant.

The event object itself serves as the commit record for the scheduling transaction. The Event object first locks the calendars of all the participants, finds an appropriate time slot, and then enters itself on all of the calendars in that time slot. A two-phase commit is used to ensure that the event is entered on all of the calendars, or is aborted; the locks can then be released. Each calendar replicates the data concerning the time span covered by the event; this is to enable searches for free times to proceed without the need to invoke every Event object. Since the timing information does not change, its consistency is not an issue. The calendar also keeps information about the status of the event. Initially, the event is tentative; when the calendar's owner agrees to the engagement, the status becomes confirmed. Again, a two-phase commit is used to ensure that the Event object and the confirming calendar agree. When the last participant agrees to the engagement, the Event object attempts to notify all of the participants that the



Representation

Owner's Name:	Andrew P. Black	
Last Read:	Monday, 8th September 1986 at 09:30	
Last Updated:	Monday, 8th September 1986 at 09:05	
Appointments:		
Start Time:	22 ix 1986 13:30	23 ix 1985 12:00
End Time:	22 ix 1986 15:00	23 ix 1985 13:00
Status:	definite	tentative
Owner Action:	accepted	no action
Capability:	<Figure 2.3>	<lunch event>
History:		
Cancelled and past events		

Operations

AddEvent(), Remove()
Confirm()
FindFreeBlock()
GetLock(), FreeLock()
ListNext()
SetName()
SetStatus()
Show(), ShowEventTotals()

Figure 2.4: A Calendar Object

appointment is now definitely scheduled. However, this information is relayed to the calendars on a "best effort" basis; transaction techniques are not used. This is done so that the last user is not prevented from confirming an engagement just because another participant's calendar is unavailable (perhaps because a machine is down). The "confirmed" status in a calendar is thus a hint that needs to be checked against the truth held in the Event object; it is possible that the event is in fact scheduled, but that the calendar object has not been informed. The process of scheduling an event is shown in Figure 2.5.

2.5. A Technical Assessment

The Eden system can be assessed in two frames of reference. First, considered as an architecture for supporting distributed applications, one can take specific features of Eden and see how they contribute to that goal. A recent SOSP paper [16] attempts this task; in this forum we will merely mention some of the more significant findings. Secondly, one can assess the current implementation of Eden as an artifact in its own right: how good a job did we do in building it, and in what ways did our substrate system (UNIX) help us or hinder us.

Considered as an architecture, Eden provides good support for applications. This is of course the finding we hoped for, but it is certainly not just experimental bias. The fact that two students could undertake the building of a distributed mail system as a six week project for the graduate operating systems course – and complete the task on a kernel system that was still being actively debugged – was surprising even to us. The credit must go not only to the system design itself, but also to the programming language support that was provided, almost as an afterthought. EPL's provision of syntactic support for invocation receipt and dispatch, and the combination of synchronous invocation and lightweight processes, seem to be tools that are accessible to the ordinary programmer. Those of us deeply involved in the mystique of the innards of Eden were at first alarmed when programmers who had constructed substantial applications asked questions that displayed what was to us an amazing ignorance of the way the system worked. But of course, that is exactly as it should be: the programming language itself should present a coherent model of computation, and programmers should not need to delve below that level.

One concept that has proved to be very important is the *Abstract Edentype*. While a concrete Edentype is a particular piece of code that defines the interface and behavior of a real object, an Abstract Edentype is an abstraction of this: the specification of an interface and a behavior. This specification may be satisfied by many

Status in Event	Status in Harper's Calendar	Status in Black's Calendar
Tentatively Scheduled by Harper:		
aborted		
	pre-tentative	
		pre-tentative
tentative		
	tentative	
		tentative
Confirmed by Harper:		
	pre-confirmed	
confirmed (Harper)		
	confirmed	
Confirmed by Black:		
		pre-confirmed
confirmed (Black)		
		confirmed
When all participants have confirmed ...		
scheduled		
	scheduled	
		scheduled

Figure 2.5: Stages in Scheduling an Event

concrete Edentypes or by none. A given concrete Edentype may implement several abstractions. The most obvious application of this idea is in device-independent transput. The abstraction of a readable *Stream* is implemented by several concrete Edentypes, in particular by sequential files and by the terminal handler. The implementation of the *transfer* and *close* operations is obviously very different in each case, but an invoking object need not be concerned with this, provided that data is available when *transfer* is called. Once recognized, Abstract Edentypes crop up in all sorts of applications; for example, the extended version of Edmas [5] uses an internal abstraction *MailSink*, as well as sharing an abstract Edentype with the file system. Three different concrete types in the transaction-manager tree [51] implement a lock manager interface; in this case, they also share the same implementation module.

Eden allows the use of Abstract Edentypes, but does not offer any explicit support for them. Because capabilities are not typed, a client that claims to be reading from a *stream* object will have no problem reading from a sequential file, provided that the file supports the right interface. Explicit support for abstract types is one of the design goals of a new distributed object-oriented language currently being implemented [18,20].

One of the less successful features of Eden is *Checkpoint*. In its favor is simplicity of concept and universality: any desired updating of the passive representation can be achieved by using multiple checkpoints. However, in practice this is of little use, because the cost of checkpointing is too high to make it a useful primitive. It may take as long as a second to perform a checkpoint operation – if the amount of data is small (less than a few kilobytes), this time is more or less independent of the size of the data. The reason is that most of the time is occupied by UNIX overhead in updating the disk atomically. This is not something for which the UNIX file system is particularly well adapted. Indeed, in our initial implementation under Berkeley 4.1 UNIX, it was impossible. Berkeley 4.2 UNIX provides an atomic *rename* system call and an operation that flushes the disk cache for a particular file; these enable us to achieve atomicity, but it remains very expensive. Over seventy five per cent of the CPU time used by an object in checkpointing a kilobyte is consumed by *link*, *unlink*, *open*, *creat*, and *access*; *write* uses seven per cent of the CPU time. Similarly, over seventy per cent of the time spent by the kernel process at the checksite is consumed by *link*, *unlink*, and *open*.

One way of avoiding the inefficiencies of the UNIX file system would have been to use the raw disk interface instead. The problem with this is that we would no longer be able to use *exec* to load the code of an object into a new address space. Instead we would have to write our own loader, and execute out of data space. This in turn would have prevented us from sharing code between objects of the same Edentype that happen to be on the same machine. Since object code is large (usually over 150 kbytes, much of it in the form of common libraries) this would have significantly increased swapping and paging.

Another way of looking at the deficiencies of our checkpoint operation is to say that it hides the power of the disk. Object programmers know that disks are capable of random access, and they resent being forced to treat the disk as if it were a magnetic tape. If the file is organized as a list of pages, then a small atomic change can be made simply by creating replacements for a few pages in the file and changing some of the page references in the index. In other words, the disk is capable of atomically changing a small part of a large file, but we do not take advantage of it. In fact, the paging hardware is equipped with dirty bits that could do an efficient job of recording exactly which pages have been changed since the last checkpoint – but of course the UNIX abstraction of address space does not allow one to access that level of the implementation.

Another place where UNIX hides power is discussed at length elsewhere [16]: the Berkeley 4.2 inter-process communication primitives, which omit to report if an IPC message has been dropped, even in the local case. But it is easy to criticize UNIX, and to forget its benefits. Apart from the availability of source code and the relative ease with which it could be modified, the chief reason for our choice of UNIX as a prototyping environment was that it provided a path whereby users could be migrated gradually onto Eden. It also provided an environment in which it was possible to use UNIX tools to accomplish Eden tasks. As an example, at a time when there were no Eden facilities for input from and output to the terminal, it was possible to demonstrate the Eden mail system by using the Emacs editor and a filter process to compose a mail message [6]. Similarly, we have an interface that allows one to use Emacs to edit Eden files as easily as UNIX files.

Another advantage of building Eden on top of UNIX is that the two systems can coexist. Eden is currently running on sixteen Sun workstations, including the one on which we are composing and formatting this report. When Eden activity is low, Eden does not intrude on the UNIX user, yet the constant availability of Eden makes it a more suitable laboratory than if the workstation cluster had to be rebooted with the Eden system for each experimental use.

In the final analysis, we think that we made an appropriate choice in picking UNIX as a prototyping environment. The main cost is performance, and Eden has been criticized on these grounds. In fact, performance of invocation has improved substantially since our first messages were exchanged, and now approaches the limit of what one can expect from a system that requires four cross-address space calls for each invocation and reply. It is adequate for a wide range of experiments. Checkpointing and activation are limited by the speed and structure of the file system, and are more of a bottleneck in some applications. We believe that further significant performance gains can come only from a major reimplementation, in which the use of UNIX processes is severely restricted. As an illustration, the current prototype of the Emerald distributed programming system implements all the objects on a given node inside one UNIX address space. Local invocations can therefore be made without incurring the cost of UNIX context switches; as a result, the most general form of local invocation is only fifty per cent. slower than an ordinary procedure call. Only when accessing the network is it necessary to call the UNIX kernel. Nevertheless, the Emerald workstations can still run UNIX, which is an aid to debugging and provides a suitable environment for the Emerald compiler.

2.6. Summary

Eden is an implementation of an advanced object-oriented distributed programming environment. It is supported by its own programming language, which provided an early implementation of what has come to be known as Remote Procedure Call,[†] including full stub generation for both the invoker and invokee. Over three hundred thousand lines of EPL application code have been written, comprising about a hundred different Edentypes. Significant experimental research projects in transactions, concurrency control, replication, and loadsharing, as well as many more minor studies, have been carried out on top of Eden; none of this would have been possible without the

† In the context of Eden the name is unfortunate: it is fundamental to the system that the invoker does not need to know whether the target is remote, and that the called entity is an object rather than a procedure.

existence of the Eden system.

Having reviewed these successes, it is only fair to point out that not all of the original technical objectives of Eden were fully achieved. Partly this is due to some naive objectives; for example, the 1979 proposal anticipated the Eden system becoming the general department computing utility. Partly this is due to some naive choices in the early years of the project; for example, an initial decision to base the implementation on the Intel iAPX-432 processor. On balance, though, we are pleased with the accomplishments of the project.

3

Technical Impact

In this brief section we attempt to be specific concerning the technical contributions of the Eden project. As noted at the conclusion of the previous section, not all of the original objectives of the project were fully realized. Nonetheless, the accomplishments of Eden are significant, both when viewed absolutely and when viewed relative to other research projects of comparable scope.

Technical impact, of course, is quite hard to quantify. One can count refereed publications (twenty one to date). One can count theses directly related to the Eden research (six completed Doctoral theses, with another six nearing completion; fifteen completed Masters theses). One can observe the influence of Eden on current research in distributed systems (for example, the ISIS project [60] the Clouds Project [59] and the Amoeba system [67]) and even on commercial systems (the Apollo Domain System has adopted may of Eden's features, although the implementation is radically different). Eden was recently chosen for detailed description in Tanenbaum's "Distributed Operating Systems" paper in *ACM Computing Surveys* [68]. A 1985 mailing describing Eden technical reports attracted several hundred requests for copies. Whatever metric one chooses, it is clear that the impact of the Eden project on the technical community has been significant.

Eden's accomplishments arise from the design, implementation, and use of a series of prototype implementations, from the collection of applications that have been implemented, from the research that has been conducted on top of Eden, and from its substantial impact on systems research in the department. Here we briefly list some of the more significant achievements:

- Three versions of Eden were implemented: a single-node prototype built on a VAX running VMS, a distributed version based on VAX/UNIX, and a distributed version built on Sun/UNIX. This last version has been in use since 1984.
- Eden is the first complete implementation of a distributed, object-based system with location-independent invocation of capability-addressed objects. Eden objects are *active* and *mobile*. Eden's invocation mechanism was one of the first implementations of a complete remote procedure call facility.
- A language for programming distributed applications, the Eden Programming Language (EPL) [17], was designed, implemented, and used to build a variety of applications. Based on Concurrent Euclid, EPL provides direct support for two of the fundamental abstractions of Eden: capabilities and invocation.
- While built on top of UNIX, Eden itself provides the user with a complete, advanced environment for the development and execution of distributed applications.
- Eden papers have appeared at each of the last four ACM Symposia on Operating Systems Principles – the major conference in the field [1,15,16,36]. (The 1979 paper was a result of work undertaken during preparation of the grant proposal.) Another major paper appeared in *IEEE Trans. on Software Engineering* [7].
- An adaptation of an Eden M.S. thesis has become a highly regarded monograph [40].
- Within 7 years after the start of the Eden project, we expect that Eden will have spawned roughly a dozen Ph.D.s and fifteen Master's degrees. The Master's graduates are shown in Table 3.1; Ph.D. graduates are noted in Table 4.4.
- A significant number of distributed applications and services were constructed on Eden, including:
  - the Eden File System [32]
  - the Eden Mail System [5,6]
  - the Eden Shared Calendar System [27]
  - the Eden terminal handler

Name	Date	Thesis Title	Advisor
Ellen M. Bierman	12/81	*A Comparative Study of Network-Based Object-Oriented File Systems*	E. Lazowska
Henry M. Levy	12/81	*A Comparative Study of Capability-Based Computer Architectures*	E. Lazowska
Susan J. St. John	12/82	*The Eden Display Subsystem*	E. Lazowska
Jimmy D. Nilson	3/83	*The Eden Node Machine Virtual Terminal*	E. Lazowska
Susan J. Cady	6/83	*DOOM: A Distributed Object-Oriented Message Module*	E. Lazowska
Paul Jensen	6/83	*The Eden Command Language Virtual Machine*	A. Black
Tod K. Johnson	6/83	*Distributed Operating Systems: A Comparison of Two Systems Emphasizing Problems of Distribution*	G. Almes
Wendy Rowley	6/83	*A Graphics Interface for Eden*	G. Almes
Barry C. McCord	1/84	*The Eden Programming Language Translator*	A. Black
Felix S. Hsu	3/85	*Re-Implementing Remote Procedure Calls*	A. Black
Karen Beall	12/85	*Charts for Displaying Quantitative Information in the Eden System*	K. Sloan
Andrew Proudfoot	12/85	*Replects: Data Replication in the Eden System*	J. Noe
Tom Yap	12/85	*Concurrent Euclid Message Module*	A. Black
Agnes Andreassian	6/86	*Effectiveness of Replication in Distributed Computer Systems*	J. Noe
Richard Korry	6/86	*Load Sharing in a Workstation Environment*	J. Zahorjan

Table 3.1: Eden-Related M.S. Degrees
(See Table 4.4 for Eden-Related Ph.D. Degrees)

  - the Eden command language interpreter
  - the Eden Window System (on which the work described in reference 13 is based)
  - the distributed Eden Programming Language translator
  - an application-level service for replicating Eden objects [45], which led to a new technique, *regeneration*, for automatically regenerating lost replicas [52]
  - a kernel-level service for replicating Eden objects [48]
  - an implementation of nested transactions [50, 51]

- Because of characteristics such as location independence, mobility, and remote procedure call, Eden has also been used as a research tool in a number of projects, including:
  - studies of load sharing [9, 22, 23, 35]
  - studies of graphical interfaces [10, 13]
  - a study of replication policies and a comparison of replication methods [47]
  - a study of the performance of time interval concurrency control techniques [46]
- Issues arising directly from Eden have spawned a number of research efforts, including:
  - the design and implementation of a visual, object-oriented, editing-based user interface [55]
  - an analysis of object finding algorithms using forwarding addresses [24, 25]
  - The design of an algorithm for on-the-fly backup of databases [49]
  - a study of garbage collection in distributed systems [57]

- a project in language support for distributed programming [18,20,29,34]
- studies of diskless workstation performance [37] and of dynamic file usage patterns [66]
- a study to design and implement advanced architectural support for object-based distributed systems [11]
- the design and use of a set of notational tools for the specification and implementation of concurrent programs [43]
- a study of the way in which transaction mechanisms can take advantage of object semantics to increase concurrency [30]

Over Eden's lifetime, roughly 40 graduate students, 6 faculty members, and 8 staff members have participated directly in the project; many others have benefited from its existence. A 1985 NSF site visitor stated:

Viewed as a research project rather than as product development, I would rate Eden as a great success; something to be very proud of.

4

Influence on the Environment

Eden, like most CER projects, had two objectives. The first objective was technical: to carry out a specific experimental research project. (Eden was considerably more focused in this regard than most of its successors in the CER program.) The second objective was environmental: to strengthen Washington's department, and to transform it into one where experimental techniques are routinely employed. It is the environmental aspect that is addressed in this section.

It is difficult to assess environmental changes quantitatively, but the Eden Project has played a substantial role in a number of key areas:

- While Washington's department still is moderate in size, we have seen substantial growth, particularly in experimental areas. We are able to recruit excellent new faculty.
- Our orientation continues to be research and Ph.D. education; we have a small, highly-selective undergraduate major program, and a minimal service obligation.
- Grant and contract revenue has increased substantially.
- State support also has increased substantially; particularly important is a large commitment of laboratory space for both research and instruction.
- The composition of our graduate program has improved, and our production of experimentally-oriented Ph.D. graduates has increased markedly.
- Many new industrial relationships have been formed.

These points are illustrated in tabular form in the remainder of this section.

Table 4.1 provides a statistical snapshot of the department in September 1986, as well as 1980 benchmarks for certain key indices. These indices point not only to healthy growth, but also to an increasing experimental orientation.

23 faculty *(up from 14 in 1980)*	Annual budget: $4,000,000+
11 technical support staff *(up from 1)*	55% external research
14 administrative support staff	17 NSF grants (including 1 CER)
90 Ph.D. students 40 M.S. students	4 DoD contracts
15% of applicants are offered admission	9 industrial grants
45% of offers are accepted	35% instruction
150 undergraduates (Juniors and Seniors only)	1/3 undergraduate instruction
very low service teaching load	1/3 graduate instruction
Space per faculty member: 900 square feet *(up from 630)*	1/3 graduate advising
Capitalization per faculty member: $110,000 *(up from $42,000)*	5% internal research
Ranked in the top 10 in the 1982 reputational survey.	5% miscellaneous
	Non-CER Federal revenue up from $400,000 to $2,000,000
	State budget up from $500,000 to $1,500,000

Table 4.1: Statistical Snapshot

Table 4.2 lists the faculty. Half of the current faculty have joined the department since 1980. Of the recent hires, Levy, Snyder, Henry, Notkin, Sloan, DeRose, Ebeling, and Schlag all exhibit a significant experimental dimension in their research. These people have chosen to come to Washington over competing offers from the best academic departments (e.g., Stanford) and the best industrial research laboratories (e.g., the DEC Systems Research Center).

1959	Hellmut Golde	(Stanford)
1961	Theodore H. Kehl	(Wisconsin)
1968	Jerre D. Noe	(Stanford/SRI)
1969	Jean-Loup Baer	(UCLA)
1971	Richard E. Ladner	(Berkeley)
	Alan C. Shaw	(Stanford/Cornell)
1977	Edward D. Lazowska	(Toronto)
	Walter L. Ruzzo	(Berkeley)
	Steven L. Tanimoto	(Princeton/Connecticut)
1980	Alan Borning	(Stanford/Edinburgh)
	John Zahorjan	(Toronto)
1981	Andrew P. Black	(Oxford)
1983	Henry M. Levy	(Washington/DEC)
	Lawrence Snyder	(CMU/Yale/Purdue)
	Paul Young	(MIT/Stanford/Purdue)
1984	Robert Henry	(Berkeley)
	David Notkin	(CMU)
	Richard Pattis	(Stanford)
	Kenneth R. Sloan	(Pennsylvania/Rochester/MIT)
1985	Tony DeRose	(Berkeley)
1986	Richard Anderson	(Stanford/MSRI)
	Carl Ebeling	(CMU)
	Martine Schlag	(UCLA)
1987	Paul Beame	(Toronto/MIT)

Table 4.2: The Faculty

As indicated in Table 4.3, the department has six areas in which its reputation is particularly strong. Many faculty span multiple areas; this is a major strength of our research programs.

Computer Architecture & VLSI	**Programming Environments & User Interfaces**
Jean-Loup Baer	Alan Borning
Carl Ebeling	Richard Ladner
Ted Kehl	David Notkin
Hank Levy	Larry Snyder
Martine Schlag	Steve Tanimoto
Larry Snyder	
Computer System Performance Analysis	**Image Analysis and Graphics**
Jean-Loup Baer	Tony DeRose
Ed Lazowska	Ken Sloan
Jerre Noe	Steve Tanimoto
John Zahorjan	
Computer Systems	**Theory of Computation**
Andrew Black	Richard Anderson
Robert Henry	Paul Beame
Ed Lazowska	Richard Ladner
Hank Levy	Larry Ruzzo
Jerre Noe	Martine Schlag
David Notkin	Larry Snyder
Alan Shaw	Paul Young
John Zahorjan	

Table 4.3: Major Areas of Research Strength

At the graduate level, the major change has not been in the number of enrolled students (which has remained relatively constant), but in their character. Full time students have increased from 56% of those students who entered in the 1979–1981 biennium to 91% currently. Students whose bachelors degree was from the University of Washington made up 22% of our intake in the 1979–1981 biennium; they now account for only 8%. These figures reflect the transformation of the department from one filling regional needs into one with a national reputation.

Finally, the number of Ph.D. graduates has increased from 7 in the 1979–1981 biennium to a realistic projection of 23 for the 1985–1987 biennium, as shown in Table 4.4.

	Name	Date	Thesis Title	Employer	Advisor
E	Jeffrey Scofield	10/85	*Editing as a Paradigm for User Interaction*	Cedar River Soft. Renton, WA	A. Borning
E	Robert J. Fowler	10/85	*Decentralized Object Finding Using Forwarding Addresses*	Univ. of Rochester Rochester, N.Y.	R. Ladner
	Sai Choi Kwan	10/85	*External Sorting: I/O Analysis and Parallel Processing Techniques*	Bell Laboratories Murray Hill, NJ	J.-L. Baer
E	Leif Nielsen	6/86	*Separation of Data Manipulation and Control*	Danish Center for Comp. Sci.	A. Black
	Robert Duisberg	6/86	*Constraint-Based Animation: The Importance of Temporal Constraints in Animus*	Tektronix Beaverton, OR	A. Borning
	Carlen B. Bennett	7/86	*Use of Temporal Coherence in Computer Animation*	Consultant Seattle, WA	S. Tanimoto
E	Calton Pu	7/86	*Replication in Nested Transactions in the Eden Distributed System*	Columbia Univ. New York, N.Y.	J. Noe
	Joseph Pfeiffer	8/86	*Integrating Low-Level and High-Level Computer Vision*	NM State Univ. Las Cruces, NM	S. Tanimoto
	Richard Furuta	8/86	*An Integrated, but not Exact-Representation, Editor/Formatter*	Univ. of Maryland College Park, MD	A. Shaw
	H. Venkateswaran	8/86	*Characterizations of Parallel Complexity Classes*	Georgia Inst. Tech. Atlanta, GA	M. Tompa
	James K. Archibald	12/86	*The Cache Coherence Problem in Multiprocessors*		J.-L. Baer
E	Stephen Vestal	1/87	*Garbage Collection: An Exercise in Distributed Fault-Tolerant Programming*	Honeywell Res. Ctr. Minneapolis, MN	E. Lazowska
E	Norman Hutchinson	1/87	*Emerald: An Object-Oriented Language for Distributed Programming*	Univ. of Arizona Tucson, AZ	H. Levy
	Ronald Blanford	6/87 (*exp.*)	*Pyramid Algorithms for Computer Vision*		S. Tanimoto
E	David Jacobson	6/87 (*exp.*)	*Transactions on Abstract Data Types*		L. Snyder
E	Eric Jul	6/87 (*exp.*)	*Object Location and Mobility in Distributed Systems*	Univ. Copenhagen Denmark	H. Levy
	Philip Nelson	6/87 (*exp.*)	*Paradigms for the Development of Parallel Algorithms*		L. Snyder
E	Sayed Banawan	8/87 (*exp.*)	*Load Sharing in Distributed Computer Systems*		J. Zahorjan
E	John K. Bennett	8/87 (*exp.*)	*Distributed Smalltalk*		E. Lazowska
E	Carl Binding	8/87 (*exp.*)	*The Specification and Implementation of a User Interface Toolkit*		A. Shaw
	Anne Condon	8/87 (*exp.*)	*Probabilistic Game Automata*		R. Ladner
E	Michael Schwartz	8/87 (*exp.*)	*Naming in Large Heterogeneous Systems*		J. Zahorjan
	Akhilesh Tyagi	8/87 (*exp.*)	*The Role of Energy in VLSI*		L. Snyder
	Chyan Yang	8/87 (*exp.*)	*The Multigauge Architecture*		L. Snyder

Table 4.4: 1985–1987 (Academic Year) Ph.D. Degrees (E denotes Eden-Related Ph.D. Degrees)

A number of important industrial relationships have formed at least partially in response to the department's involvement in the CER program. Table 4.5 displays some of these.

Digital Equipment Corporation
- VAX Pascal compiler project (1978)
- Presence of Henry Levy (1981–82; 1983–)
- External Research Program support of CER projects (1981–)
  - more than $650,000 in allowances in the past 12 months
- Sabbatical leaves
  - Lazowska to Systems Research Center, 1984–85
  - Kehl to DECwest Engineering, 1985–86
  - Black to DEC-Littleton, 1986–87
- David Cutler (DECwest Engineering), Affiliate Professor, 1985–
- Faculty Development Award to DeRose

IBM
- Research support for Ladner (DBNet), Baer (architectures for sorting)
- Faculty Development Award to Notkin
- Martin Tompa, Affiliate Associate Professor, 1985–
- $1,200,000 in instructional laboratory equipment for our department
- Two "regular" graduate fellowships

Tektronix
- Equipment support for research and instruction
- Cooperative NSF grant with Borning

Boeing
- Research support for Baer (parallel architectures), Shaw (real–time systems),
  and the graphics lab
- Janusz Kowalik (Boeing AI Center), Affiliate Professor, 1985–

Northwest Laboratory for Integrated Systems (formerly UW/NW VLSI Consortium)
- 8 corporate members: Boeing, Eldec, John Fluke, Honeywell, Microtel,
  Seattle Silicon, Silicart, Tektronix

Xerox
- $500,000 in equipment support for CER and other projects

Department Industrial Affiliates Program
- 10 corporate members

Table 4.5: Industrial Relationships

At the start of the Eden project in 1980, the University of Washington's department was regional rather than national in character. A 1985 NSF site visitor commented:

> *By every conventional measure, this department has made great progress towards becoming a top-quality national resource in computer science.*

5

Participants

The following individuals participated significantly in the Eden Project during its five-year lifetime:

Faculty:

- Guy T. Almes
- Andrew P. Black
- Michael J. Fischer
- Hellmut Golde
- Edward D. Lazowska
- Jerre D. Noe

Staff:

- Mark Baratta
- Cher Gunby
- Gary Mager
- Margie Ramsdell
- Jim Rees
- Blair Rice
- Jan Sanislo
- Voradesh Yenbut

External Review Committee:

- James C. Browne
- Peter Hibbard
- Jim Morris
- Jerry Saltzer

Graduate Students:

- Agnes Andreassian
- Sayed Banawan
- Karen Beall
- John K. Bennett
- Ellen Bierman
- Carl Binding
- Jordan Brower
- Carl Bunje
- Susan J. Cady
- Ann Condon
- Isabel Domenech
- Robert J. Fowler
- Cara Holman
- Felix Hsu
- Norman Hutchinson
- David M. Jacobson
- Paul Jensen
- Warren H. Jessop
- Tod K. Johnson
- Eric Jul
- Gary Kimura
- Thomas Knight
- Richard Korry
- Henry M. Levy
- Peter Ma
- Barry C. McCord
- Eli Messinger
- Leif S. Nielsen
- Jimmy D. Nilson
- Andrew Proudfoot
- Calton Pu
- Rajendra Raj
- Wendy Rowley
- Susan St. John
- Michael Schwartz
- Jeffrey Scofield
- H. Venkateswaren
- Steven Vestal
- Douglas Wiebe
- Thomas Yap

Bibliography

Eden-Related Literature

[1] Almes, G.T. and Lazowska, E.D. The Behavior of Ethernet-like Computer Communication Networks. *Proceedings 7th ACM Symposium on Operating Systems Principles*, December 1979, pp. 66-81.

[2] Almes, G.T. and Lazowska, E.D. Eden: Research in Integrated Distributed Computing. *Proceedings Workshop on Fundamental Issues in Distributed Computing*, December 1980.

[3] Almes, G.T. Objects in the Eden System. *Proceedings Electro/82, 32: Object Oriented Systems and Languages*, May 1982.

[4] Almes, G.T. Integration and Distribution in the Eden System. *Proceedings IEEE Workshop on Computer Systems Organization*, July 1982.

[5] Almes, G.T. and Holman, C. Edmas: An Object-Oriented, Locally Distributed Mail System. Technical Report 84-08-03, University of Washington, Department of Computer Science, August 1984.

[6] Almes, G.T., Black, A.P., Bunje, C. and Wiebe, D. Edmas: A Locally Distributed Mail System. *Proceedings 7th International Conference on Software Engineering*, March 1984, pp. 56-66.

[7] Almes, G.T., Black, A.P., Lazowska, E.D. and Noe, J.D. The Eden System: A Technical Review. *IEEE Transactions on Software Engineering SE-11*, 1 (January 1985), pp. 43-59.

[8] Andreassian, A. *Effectiveness of Replication in Distributed Computer Systems.* M.S. Thesis, University of Washington, Department of Computer Science, June 1986.

[9] Banawan, S. *An Evaluation of Load Sharing in Distributed Systems.* Ph.D. Thesis, University of Washington, Department of Computer Science, Spring 1987. (In preparation).

[10] Beall, K. *Graphical Charts for Displaying Quantitative Information in the Eden System.* M.S. Thesis, University of Washington, Department of Computer Science, December 1985.

[11] Bennett, J.K. *Distributed Smalltalk.* Ph.D. Thesis, University of Washington, Department of Computer Science, 1987. (In preparation).

[12] Bierman, E. *A Comparative Study of Network-Based Object-Oriented File Systems.* M.S. Thesis, University of Washington, Department of Computer Science, December 1981.

[13] Binding, C. The Architecture of a Window Package and its Usage in Building Interface Components. Technical Report 85-08-07, University of Washington, Department of Computer Science, August 1985.

[14] Binding, C. *The Specification and Implementation of a User Interface Tookkit.* Ph.D. Thesis, University of Washington, Department of Computer Science, Summer 1987. (In preparation).

[15] Black, A.P. An Asymmetric Stream Communication System. *Proceedings 9th ACM Symposium on Operating Systems Principles*, October 1983, pp. 4-10.

[16] Black, A.P. Supporting Distributed Applications: Experience with Eden. *Proceedings 10th ACM Symposium on Operating Systems Principles*, December 1985, pp. 181-193.

[17] Black, A.P. The Eden Programming Language. Technical Report 85-09-01, University of Washington, Department of Computer Science, September 1985.

[18] Black, A.P., Hutchinson, N., Jul, E. and Levy, H.M. Object Structure in the Emerald System. *Proceedings First Conference on Object-Oriented Programming Systems, Languages and Applications*, October 1986.

[19] Black, A.P. The Eden Project: Overview and Experiences. *Proceedings European UNIX systems User Group Autumn '86 Conference*, Manchester, UK, September 1986, pp. 177-189.

[20] Black, A.P., Hutchinson, N., Jul, E., Levy, H.M. and Carter, L. Distribution and Abstract Types in Emerald. *IEEE Transactions on Software Engineering SE-13*, 1 (January 1987).

[21] Cady, S.J. *DOOM: A Distributed Object-Oriented Message Module.* M.S. Thesis, University of Washington, Department of Computer Science, June 1983.

[22] Eager, D.L., Lazowska, E.D. and Zahorjan, J. Adaptive Load Sharing in Homogeneous Distributed Systems. *IEEE Transactions on Software Engineering SE-12*, 5 (May 1986), pp. 662-675.

[23] Eager, D.L., Lazowska, E.D. and Zahorjan, J. A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing. *Performance Evaluation 6* (April 1986), pp. 53-68.

[24] Fowler, R.J. *Decentralized Object Finding Using Forwarding Addresses.* Ph.D. Thesis, University of Washington, Department of Computer Science, December 1985.

[25] Fowler, R.J. The Complexity of Using Forwarding Addresses for Decentralized Object Finding. *Proceedings 5th Annual ACM Symposium Principles of Distributed Computing*, August 1986.

[26] Greenberg, A.G. *Efficient Algorithms for Multiple Access Channels.* Ph.D. Thesis, University of Washington, Department of Computer Science, August 1983.

[27] Holman, C. and Almes, G.T. The Eden Shared Calendar System. Technical Report 85-05-02, University of Washington, Department of Computer Science, May 1985.

[28] Hsu, F.S. *Re-Implementing Remote Procedure Call.* M.S. Thesis, University of Washington, Department of Computer Science, March 1985.

[29] Hutchinson, N. *Emerald: An Object-Oriented Language for Distributed Programming.* Ph.D. Thesis, University of Washington, Department of Computer Science, January 1987.

[30] Jacobson, D.M. *Transactions on Abstract Data Types.* Ph.D. Thesis, University of Washington, Department of Computer Science, Winter 1987. (In preparation).

[31] Jensen, P. *The Eden Command Language Virtual Machine.* M.S. Thesis, University of Washington, Department of Computer Science, June 1983.

[32] Jessop, W.H., Jacobson, D.M., Noe, J.D., Baer, J. and Pu, C. The Eden Transaction Based File System. *Proceedings 2nd Symposium on Reliability in Distributed Software and Database Systems*, July 1982, pp. 163-169.

[33] Johnson, T.K. *Distributed Operating Systems: A Comparison of Two Systems Emphasizing Problems of Distribution.* M.S. Thesis, University of Washington, Department of Computer Science, June 1983.

[34] Jul, E. *Object Location and Mobility in Distributed Systems.* Ph.D. Thesis, University of Washington, Department of Computer Science, Winter 1987. (In preparation).

[35] Korry, R. *Load Sharing in a Workstation Environment.* M.S. Thesis, University of Washington, Department of Computer Science, June 1986.

[36] Lazowska, E., Levy, H., Almes, G., Fischer, M., Fowler, R. and Vestal, S. The Architecture of the Eden System. *Proceedings 8th ACM Symposium on Operating Systems Principles*, December 1981, pp. 148-159.

[37] Lazowska, E.D., Zahorjan, J., Cheriton, D.R. and Zwaenepoel, W. File Access Performance of Diskless Workstations. *ACM Transactions on Computer Systems 4*, 3 (August 1986), pp. 238-268.

[38] Levy, H.M. *A Comparative Study of Capability-Based Computer Architectures.* M.S. Thesis, University of Washington, Department of Computer Science, December 1981.

[39] Levy, H.M. A History of Capability-Based Computer Systems. *Proceedings Electro/82, 32: Object Oriented Systems and Languages*, May 1982.

[40] Levy, H.M. *Capability-Based Computer Systems.* Digital Press, Bedford, MA, 1984.

[41] Manber, U. *Concurrency Control for Dynamic Data Structures and Fault Tolerance.* Ph.D. Thesis, University of Washington, Department of Computer Science, August 1982.

[42] McCord, B.C. *The Eden Programming Language Translator.* M.S. Thesis, University of Washington, Department of Computer Science, January 1984.

[43] Nielsen, L.S. *Separation of Data Manipulation and Control.* Ph.D. Thesis, University of Washington, Department of Computer Science, July 1986.

[44] Nilson, J.D. *The Eden Node Machine Virtual Terminal.* M.S. Project, University of Washington, Department of Computer Science, March 1983.

[45] Noe, J., Proudfoot, A. and Pu, C. Replication in Distributed Systems: The Eden Experience. *Proceedings Fall Joint Computer Conference*, November 1986, pp. 1197-1209.

[46] Noe, J.D. and Wagner, D.B. Measured Performance of Time Interval Concurrency Control Techniques. Technical Report 86-08-05, University of Washington, Department of Computer Science, August 1986.

[47] Noe, J.D. and Andreassian, A. Effectiveness of Replication in Distributed Computer Networks. Technical Report 86-06-05, University of Washington, Department of Computer Science, June 1986.

[48] Proudfoot, A.B. *Replects: Data Replication in the Eden System.* M.S. Thesis, University of Washington, Department of Computer Science, October 1985.

[49] Pu, C. On-the-Fly, Incremental, Consistent Reading of Entire Databases. *Proceedings Eleventh International Conference on Very Large Data Bases,* Stockholm, Sweden, August 1985.

[50] Pu, C. and Noe, J.D. Nested Transactions for General Objects: The Eden Implementation. Technical Report 85-12-03, University of Washington, Department of Computer Science, December 1985.

[51] Pu, C. *Replication and Nested Transactions in the Eden Distributed System.* Ph.D. Thesis, University of Washington, Department of Computer Science, July 1986.

[52] Pu, C., Noe, J.D. and Proudfoot, A. Regeneration of Replicated Objects: A Technique for Increased Availability. *Proceedings Second International Conference on Data Engineering,* Los Angeles, CA, February 1986, pp. 175-187.

[53] Rowley, W. *A Graphics Interface for Eden.* M.S. Thesis, University of Washington, Department of Computer Science, June 1983.

[54] Schwartz, M. *Naming in Large Heterogeneous Systems.* Ph.D. Thesis, University of Washington, Department of Computer Science, Summer 1987. (In preparation).

[55] Scofield, J. *Editing as a Paradigm for User Interaction.* Ph.D. Thesis, University of Washington, Department of Computer Science, August 1985.

[56] St.John, S.J. *The Eden Display Subsystem.* M.S. Project, University of Washington, Department of Computer Science, December 1982.

[57] Vestal, S. *Garbage Collection: An Exercise in Distributed Fault-tolerant Programming.* Ph.D. Thesis, University of Washington, Department of Computer Science, January 1987.

[58] Yap, T. *Concurrent Euclid Message Module.* M.S. Thesis, University of Washington, Department of Computer Science, December 1985.

Other Cited Literature

[59] Allchin, J.E. and McKendry, M.S. Synchronization and Recovery of Actions. *Proceedings 2nd Annual ACM Symposium Principles of Distributed Computing,* August 1983, pp. 31-44.

[60] Birman, K.P., Abbadi, A.E., Dietrich, W., Joseph, T. and Raeuchle, T. An Overview of the Isis Project. Technical Report 84-642, Department of Computer Science, Cornell University, October 1984.

[61] Birrell, A.D. and Nelson, B.J. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems 2,* 1 (February 1984), pp. 39-59.

[62] Lampson, B.W. and Sturgis, H.E. Reflections on an Operating System Design. *Comm. ACM 19,* 5 (May 1976), pp. 251-265.

[63] Leach, P.J., Levine, P.H., Douros, B.P., Hamilton, J.A., Nelson, D.L. and Stumpf, B.L. The Architecture of an Integrated Local Network. *IEEE Journal on Selected Areas in Communications SAC-1,* 5 (November 1983), pp. 842-857.

[64] Liskov, B. and Scheiffer, R. Guardians and Actions: Linguistic Support for Robust, Distributed Programs. *Conference Record of the Ninth ACM Symposium on Principles of Programming Languages,* Albuquerque, NM, January 1982.

[65] Liskov, B. Overview of the Argus Language and System. Programming Methodology Group Memo 40, M.I.T., Laboratory for Computer Science, February 1984.

[66] Maloney, J.H. and Black, A.P. File Sessions: A Technique and its Application to the UNIX File System. *Proceedings 3rd International Conference on Data Engineering,* February 1987.

[67] Mullender, S.J. and Tanenbaum, A. The Design of a Capability-Based Operating System. *The Computer Journal 29,* 4, pp. 289-306.

[68] Tanenbaum, A.S. and van Renesse, R. Distributed Operating Systems. *ACM Computing Surveys 17,* 4 (December 1985), pp. 419-470.

[69] Walker, B., Popek, G., English, R., Kline, C. and Thiel, G. The LOCUS Distributed Operating System. *Proceedings 9th ACM Symposium on Operating Systems Principles,* October 1983, pp. 49-70.

[70] Wulf, W.A., Levin, R. and Harbison, S.P. *HYDRA/C.mmp: An Experimental Computer System.* McGraw-Hill, 1981.