Report on the

Second European SIGOPS Workshop "Making Distributed Systems Work"

September 8 – 10, 1986
Amsterdam, Netherlands

Introduction

The organization of the Second European SIGOPS Workshop in Amsterdam established a new tradition: a bi-annual workshop on operating systems in Europe. The initiative to this new and important tradition was taken by Liba Svobodova. She proposed and organized the first workshop in Zurich, Switzerland and expressed the hope that it would become a tradition.

The first workshop was a success; nearly a hundred people expressed their interest, 58 were invited and 56 from 11 countries attended. Much of the current research in computer networks and distributed systems was represented, and the discussions were spirited and of a very high level.

Organizing a second workshop with an equally high level and an equally good turn-out seemed a difficult task. Fortunately, the Zurich workshop had established a reputation that the Amsterdam workshop could build on. Although the call for participation went out rather late, the number of position papers received exceeded our expectations. We received 85 position papers, almost without exception of a very high level.

We did not want the workshop to turn into a small conference, we wanted as much interaction through discussions as possible. The programme committee, therefore, decided to accept no more than 50 participants. A larger number would make a reasonable discussion impossible. In order to allow as many projects as possible to be represented, the programme committee also decided to accept 50 position papers, and allow one author per accepted position paper to attend the workshop.

The workshop itself consisted of 10 sessions, chosen such that the subjects covered most of the material submitted in the accepted position papers. Each session began by very short introductions (ten minutes and not a second more) by a few invited speakers, chosen to represent various angles to the subject matter of the session. The rest of the session was then devoted to discussions.

The reports of the sessions were prepared by the session chairpeople. The one on the session 'Applications' is missing, unfortunately. The notes for these reports were made by Bram Janssen, Robbert van Renesse and Jennifer Steiner. They did a wonderful job and made the work of the session chairpeople much lighter.

I would like to thank the Programme Committee and the Session Chairpeople for the work they put into the workshop, the Centre for Mathematics and Computer Science in Amsterdam for hosting it, and the Ministry of Science and Education and the Mayor and Aldermen of the City of Amsterdam for presenting the participants with a reception in the Amsterdam Historical Museum. I am not in a position to thank Diaboli In Musica for the concert they gave.

A special word of thanks has to go to Marja Hegt, the secretary of the Department of Algorithmics and Architecture of CWI, for handling the organizational work for the workshop almost singlehandedly. We wouldn't have had a successful workshop without her.

Sape Mullender
Programme Chairman

Sessions

Established Projects — Disillusionment

Session chair	Sape Mullender
Speakers	David Cheriton
	Mark Gealy
	Paul Levine
	Jerry Saltzer
	Andy Tanenbaum

New Projects — Hope

Session chair	Andrew Herbert
Speakers	David Anderson
	Benjamin Bacarisse
	Bharat Bhargava
	Sacha Krakowiak
	Radu Popescu-Zeletin
	Marc Shapiro

Lessons Learned — Paradise Lost

Session chair	Liba Svobodova
Speakers	Andrew Birrell
	Dave Clark
	Paul Leach
	Lindsay Marshall
	Sape Mullender
	Roger Needham

Applications — Succeeding in spite of it

Session chair	Najah Naffah
Speakers	Orna Berry
	Bojan Grosilj
	Lee Hollaar
	Douglas Terry

Tools — Building Up

Session chair	Andy Tanenbaum
Speakers	Gregory Andrews
	Mark Day
	Alan Dempster
	Roger Gimson
	Ralf Guido Herrtwich

Debugging — Knocking Down

Session chair	Dave Clark
Speakers	Jean Bacon
	Dieter Haban
	Steve Vinter
	Larry Wittie

Mechanisms — Clockwork

Session chair	Jerry Saltzer
Speakers	Jean Paul Banatre
	David Cheriton
	Hermann Kopetz
	Roger Needham

Security — Crime & Punishment

Session chair	Klaus Peter Lohr
Speakers	Andrew Birrell
	Dan Nesset
	Andy Tanenbaum
	Michael West

Upcoming Approaches — New Wave

Session chair	Walter Tichy
Speakers	M. Auslander
	David Gelernter
	Toby Bloom
	Michael Evangelist

Making Distributed Systems Work (Paradise Regained)

Session chair	Sape Mullender

Established Projects

Chaired by Sape Mullender

The *established projects* session was shorter than the other sessions, so there was very little room for discussion. Five well-known distributed systems were presented to start off the workshop by giving five different viewpoints on distributed systems research.

Experience with Clearinghouse in Large Internetworks

Mark Gealy, Xerox Corp.

Clearinghouse is a nameserver for internetworks. It maps *names* into data, called *property*. Property may be binary data (e.g., a network address) or a sequence of names (e.g., a distribution list). The name space is hierarchical and has three levels, for instance, <Mark A. Gealy : Palo Alto : Xerox>, and is parsed as *<local name : domain name : organization name>*.

The Clearinghouse database is a distributed and replicated one: No single server has all information, therefore it is distributed; portions of the database are replicated in several servers, therefore the database is replicated. The unit of replication is the domain.

Updates to the database are propagated using electronic mail. The Clearinghouse that accepts a database change sends a notice to the other Clearinghouses serving that domain. Concurrent updates at different servers may cause inconsistencies. A background *"antientropy" checker in each Clearinghouse compares its copy of each domain with the other copies and fixes inconsistencies.*

Currently, some 200 Ethernets, distributed across Japan, North America and Western Europe are connected by internetwork routers running at speeds varying between 1200 baud and a megabaud. In this network, about 350 Clearinghouse services are in operation. The root domain, which is universally replicated, is thus replicated some 350 times. The "Xerox" organization is replicated about 200 times.

The system works: data may be located and read worldwide, updates do propagate, and query performance and update propagation are reasonable in small internets. In large internetworks, there are problems, however. Network congestion is a problem. The cost of an update has complexity of $O(n^3)$ which is too high. Performance is also a function of internetwork diameter, speed of the internetwork links and the continuous availability of routes.

Problems that occur in the Xerox internetwork are: Updates may take days or weeks to propagate; updates to heavily replicated domains may not fully propagate within the lifespan of the deletion updates (of 30 days); Clearinghouse-induced traffic is a major component of all internetwork traffic; Clearinghouse processors frequently use 85% of the cpu and drive their disks at capacity; queries are slow (0.5 sec. at best, 10 sec. not uncommon). These problems are not observed in small networks. The threshold seems to be around 20 to 50

nodes.

The causes for these problems are (unordered; some have been corrected):

- Workstations locating an organization retrieve data proportional to the number of organizational servers;
- RPC transport has poorly tuned timeouts and retransmission intervals;
- Heavily replicated domains, containing large objects, are in a relatively high rate of flux;
- The frequency of ending updates for inconsistencies detected by antientropy is too high, forwarding the updates is costly and the algorithms for choosing partners are poor;
- obtaining a thousand elements requires a thousand RPCs, rather than a single RPC containing a thousand elements;
- The system has grown unbridledly.

Lessons to be learned from all this are

- Supporting heavy replication is hard to do right. It should be avoided, or better methods for doing it have to be found.
- Watch out for bugs. They are hard to find in small test cases.
- Expect far greater demand than initially suspected.
- Make sure that the algorithms scale.
- The basic architecture of Clearinghouse is serviceable.
- A large company can successfully adapt its operations to use network products.

The V Distributed System or The Protocol as the Product, not Software

David Cheriton, Stanford University

Distributed systems research is, above all, *protocol research.* A distributed system that works is a set of interconnected nodes *adequately* implementing and using a common *adequate* set of protocols. A successful distributed system is "open" if the protocols are the only prerequisite for participation.

Three levels of protocol can be distinguished: the *transport* protocol, the *naming* protocol and the *I/O* protocol.

V is an experimental system for distributed systems protocol research. The project was established four years ago, V runs on SUNs and MicroVaxes, it is in operation as the operating system for daily use. The protocols currently in use are second-generation protocols.

The V Kernel provides a transport-level communication base for all other services. The challenge in designing the V protocols was to make V's general-purpose transport protocols support file access that is at least as fast as the problem-oriented protocols of, for instance, Locus. The V system proves that this is possible using a request/response protocol with a careful kernel implementation.

The protocol, VMTP (Versatile Message Transaction Protocol, also known as V Multicast Transport Protocol), uses network-address-independent names, allowing migration, mobile hosts and multihomed hosts. Names are *stable*, that is, the relation between valid meanings and bindings does not change. Names can refer to individual entities or to groups of entities.

The protocol has been optimized for RPC. It uses selective retransmission and rate-based flow control. Requests and responses can be multiple packets. There is an *idempotent*

response option: a server can indicate to the transport service that it need not keep a response for possible retransmission; the server can regenerate it if needed.

Extended functions provided by the protocol are the well-known V-multicast facility, datagram requests (requests without response), secure communication using encryption and streaming

The V naming protocol is a protocol using names that are character strings. Again, the name space is unified by a common protocol, not a common mechanism. A client-server protocol exists for name definition, mapping and query. An additional server-server protocol exists for management purposes.

The basic name mapping protocol uses multicast to locate an appropriate server and provides for caching to minimize the number of multicasts. The preformance is near optimal for valid names and is optimally resilient if the network provides robust multicast.

The V I/O protocol defines a basic interface with a *UIO object* which is essentially a readable or writable sequence of blocks. Optionally, these objects may also be random access, editable, lockable, recoverable and replicated. It supports conventional I/O through a "thick" run-time I/O library.

Discussion

Auslander: If your cache does not give you an answer, when do you try again?

Cheriton: The hit ratio is 99.7%, so it's not important.

Casey: How's the performance of your system?

Cheriton: Very good, because we avoid maintaining protocol state.

The Domain System

Paul Levine, Apollo Computers

The first implementation of the Domain system was completed in March 1981. Now, over 20,000 workstations are running with Domain. The largest Domain network has more than 1800 nodes in nine buildings in two states, more than 50 gigabytes of storage in a single name space.

Apollo is moving from homogeneity to heterogeneity. The protocols are not the problem; reconciling semantics is. Heterogeneity, by definition, is living with mismatched semantics on different machines.

The approach in Domain is the "Open System's Toolkit," a layer between the applications programs and the various operating systems, which provides a homogeneous interface to the applications in spite of the diversity between the underlying operating systems.

Homogenizing things like *open*, *close*, *read* and *write* is relatively easy. The problems arise when trying to deal with structured files: how to pass and interpret information; binary data files: where to do the translation; consistent names: forming network-wide system-independent names; naming syntax: provide the user with a consistent set of rules and conventions.

The Apollo approach to living with these problems is to make applications use a minimum set of standard operations while making the gateways between the standard interfaces and the

host operating systems implement a maximum set of operations.

Heterogeneity problems, however, are often beyond solving. Many sad examples can be named to illustrate this. Some observations on heterogeneity are:

- The "more" an application "does," the less likely it is to run across multiple gateways.
- Generic file system application program interfaces [sic.] are (almost) impossible to write.
- If the file system gateway operations don't match *your* semantics, they are very difficult or expensive (or both) to support.
- Some application programs rightly refuse to use the "minimum" set of operations that will work to every file system.

Discussion

Tanenbaum:	Operating systems have too many bells and whistles anyway. Why try to support them all?
Levine:	The minimum set is too small.
Cheriton:	First protocol, then try matching (to find semantics that are implementable).

AΘE

Jerry Saltzer, MIT, Project Athena

The biggest difference between Project Athena and other distributed systems projects is that the goal of Athena is *educational* impact, not advances in computer systems. Athena is built of off-the-shelf systems (IBM PC/RT, MicroVAX-II).

This talk consists of eight [sic.] observations:

Firewalls in gateways are necessary.

A firewall prevents an error from spreading out over the whole network. The effects of misbehaving nodes, misbehaving software and misbehaving users should be confined to a small subset of the network.

Broadcast is bad.

Broadcast is an evil that causes many problems. Broadcast packets may flood the network, and bad broadcast packets can crash machines all over the network.

Hardware quality is awful.

Network interfaces are usually terrible. It takes a long time to get them to work. Interfaces from different vendors won't talks to each other. Interfaces from the same vendor won't talk to each other.

Porting is easy.

Applications can be easily ported between different hardware environments. An application, running on BSD 4.2, X-windows, written in C will run happily on an IBM PC/RT, on a VAX station and on a SUN, provided these machines offer BSD 4.2 Unix, X-windows and C.

Porting is hard.

Porting application between different software environments hard. Applications running on a VAX station with BSD 4.2 Unix are very difficult to get to run on a VAX station with VMS.

There is a system-human interface.

Interesting systems are usually very fragile and require experts to keep them running.

Updating is not difficult.

Software libraries, for example, need to be up-to-date, but too much so. It does not usually matter if the updates to a library shared by all users reaches every user simultaneously.

Discussion

Emer: On lazy updates.

Saltzer: Use it when it works.

Amoeba

Andy Tanenbaum, Vrije Universiteit, Amsterdam

The Amoeba system architecture recognizes four kinds of components, connected together by a fast local network: personal workstations, a processor pool, specialized servers and gateways to other Amoeba systems. The goal of the Amoeba project is to build a coherent framework for building distributed systems.

Amoeba processes have multiple threads of control in a single address space. An process (or address space) is a *cluster*, the threads of control are *tasks*. Intracluster communication between tasks can be through shared memory. Communication between tasks in different clusters is through message transactions. Amoeba has no system calls the way most operating systems have them: communication with the Amoeba Kernel is also via message transactions.

The Amoeba Kernel implements the task and cluster abstractions and provides very fast (secure) message transactions (bulk transfer rates of more than half a megabyte per second between address spaces over an Ethernet).

The Amoeba file system is a hierarchy of services. An application can choose the services best suited to its needs. File naming is part of a separate service, called *directory service*, which actually provides a general naming mechanism for any kind of object. The bottom of the hierarchy is formed by different kinds of storage servers, for example, disk block servers, stable storage servers, RAM-disk servers.

For accounting and maintaining quota on scarce resources, there is a *bank server*. Each user who wants to make some service available and establish quota (or just get paid for allowing clients to use it) can establish a *server account*. Each client of such a service must establish a *client account*. To obtain service, a client transfers "money" from its client account to the server account and then makes a request on the server. Server accounts automatically keep track of the source of a deposit. For different resources or different accounting methods, different *currencies* can be created.

The boot server is a service where other services can register if they "want to live forever." A request tells the boot server how to tell the living servers from the dead, and how to revive the dead. It does not do the crash recovery for arbitrary services, of course, but it is a useful tool for crash detection and starting up the recovery process. Boot service itself consists of multiple server processes keeping internal checks on each other.

Several bridges exist to close the gap between Amoeba and Unix. The Unix operating system has been equipped with extra system calls for doing message transactions, and the Amoeba System has a Unix-emulation library that converts Unix system calls in message transactions with Unix-supporting services: a Unix file system and a *fork-exec* server.

As one test to see if the mechanisms of Amoeba are sufficient to support parallel processing, a parallel travelling salesman algorithm was implemented on Amoeba (using branch and bound). Each processor traverses the tree to some depth and passes the remaining subtrees to other cpus.

Discussion

Naffah: How do you use ISO and X.25?

Tanenbaum: As a dumb wire.

New Projects

Chaired by Andrew Herbert

This session consisted of six presentations about new distributed system projects. The presenters were asked to explain what significant new steps they were taking in their project.

DASH

David Anderson

DASH, at the University of Berkeley, is in the design phase. The target is support for communication and graphics applications using a high bandwidth, long delay, global packet network connecting shared memory multi-processors and using encryption hardware.

At the heart of DASH is the authenticated database protocol (ADP). ADP is based on public key encryption. It provides a single key secure channel between a pair of hosts.

Above ADP there is a session-oriented resource access structure akin to Unix streams. Streams are half duplex message channels. Filter processes in the stream provide buffering; protocol processes provide sequencing, elimination of duplicates and reliability. "Bundles" of streams can be passed between clients to enable resource migration.

DASH is being implemented in C++, an object oriented programming language.

ADMIRAL

Ben Bacarisse

Ben Becarisse presented ADMIRAL, a UK Alvey-funded project involving GEC, British Telecommunications and University College, London.

The focus of ADMIRAL is to provide distributed computing tools for use in high speed WANs based on multiple IP switches driving 256 kilobit lines.

A high performance sequenced exchange protocol (SEP) is used to transport remote procedure calls. SEP provides a two layer transaction protocol to permit mutual recursion between services. SEP provides reliable uni-directional messages. The SEP service interface is designed for RPC stubs. Small transactions typically consume a single packet exchange and bulk transfer is as efficient a typical stream protocol (30 ms/call, 30 Kbytes/sec for SUN 2 Unix on 10M Ethernet).

Conclusions from using SEP have been that software engineering is a good thing and RPC brings it closer to distributed computing. However mixing language support and operating system support is dangerous; the alternative - integrated OS and language - is less "open" and makes heterogeneity difficult. Simple (but not minimal) semantics are usually sufficient.

The next ambition of ADMIRAL is to build an object-based system level structure and to distribute resource location functions between object managers with a well-defined presentation layer so that objects can be accessed across the system regardless of language, machine or operating system choices.

RAID

Bharat Bhargava

The aim of RAID is to provide an adaptable approach to reliability permitting algorithm replacement and support for replication.

There are three critical components to RAID: site recovery, atomicity control and concurrency control.

User transactions are compiled by a parser and feed to an Action Driver which accesses local databases and passes completed transactions onto the Atomicity Controller.

The Atomicity Controller ensures transaction atomicity across sites. It informs the site recovery module about transactions so that fail locks can be kept. The Concurrency Controller interacts with the Atomicity Controller to ensure serializability. This modularity permits a flexible memory organization.

The claimed feature for the system is that the flexibility to switch dynamically from eg time stamps to locking enables good performance to be obtained under differing operating conditions.

GUIDE

Sacha Krakowiak

Sacha Krakowiak of the University of Grenoble presented the GUIDE system.

The goals of GUIDE are to investigate

- object oriented system structuring
- distributed "virtual machines" to enable heterogeneity and load sharing
- version support in an object storage server
- user interfaces

GUIDE is based on RPC and permits a range of binding times (compilation, link, runtime).

Active objects are called "managers" and consist of a virtual address space containing possible multiple, nested threads of control. Static objects are called structures, and are abstract data types. (The data types are organized into a hierarchy). Objects are passed by reference,

via typed capabilities. Names, types and locations are controlled by a distributed Resource Manager, which is partially replicated. Managers (active objects) can migrate between nodes. Object invocation is by RPC.

Y

Radu Popescu-Zeletin

Radu Popescu-Zeletin of the Hahn Meitner Institute of Berlin presented the Y System.

The Y System provides support for cooperation between heterogeneous, autonomous components. The system has five layers: application, agent, server, service, unit.

An object in layer i is either a sequential or parallel composition of objects at layer $i-1$. Objects cooperate with the environment via defined gates.

Operations in Y are based on the transaction concept. Each object must preserve data consistency at the object level. Global operations are constrained to leave the system in a consistent by compensation for actions that must be undone.

SOS

Marc Shapiro

Marc Shapiro of INRIA presented the SOMIW operating system (SOS).

SOS uses an object-oriented approach. Object synchronization, access control and locking is separated from object function. Objects may be distributed and protocols are encapsulated within objects.

Clients have one interface to a resource via a local object; a proxy, which provides a handle on a remote object, or collection of remote objects.

The structure of a distributed object is visible to the proxy, but not the client and the protocol is internalized.

A Name Service is pre-installed in every context to permit new proxies to be imported as additional services are contacted.

Lessons Learned

Chaired by Liba Svobodova

After the presentations of various established and starting projects in the first two sessions, the third session focused more specifically on the lessons learned from the design, implementation, and operation of different kinds of distributed systems in different environments.

Andrew Birrell (DEC Systems Research Center, USA) assessed his experience from several distributed systems projects as basically positive, adding: "We do not build systems that don't work." But, there are a number of problems to watch for, and issues that are often neglected.

The first problem raised by Birrell was *unexpected effects of scale.* This problem had been already pointed out in the session on established projects. Birrell further qualified these effects as congestion, administration, and entropy, the last requiring end-to-end checking. Birrell's second point concerned transparent replication of resources and services. In principle, such replication is very useful, but: it makes it harder to find bugs, it can represent a substantial overhead, and it aggravates the system administrators when the replication suddenly shows through. Birrell criticized the insufficient concern for security in distributed system designs: adding security mechanisms later is difficult, and can have significant performance impact. Similar arguments can be made about concurrency. As Birrell put it, "concurrency is hard to fake and hard to achieve"; it must be built in early in the design. Birrell also brought up the issue of the lack of truly distributed applications, referring to the remark Roger Needham (University of Cambridge, U.K.) had made in the previous SIGOPS Workshop held in January 1985: "Perhaps mail is the only distributed application." We have not seem to have made much progress in this area since that first workshop.

Birrell concluded with an encouraging statement that "simple systems work surprisingly well." This was also the main message of Roger Needham (University of Cambridge, U.K.): simple things nearly always work, and simple things are extensible. Needham also pointed out some problems, though: faulty or 'mean' clients can cause a needless overload of servers; some control mechanisms (capability-based) are needed in the clients. Finally, Needham remarked that a cold start of a distributed system can be rather difficult.

Paul Leach (Apollo Computer Inc., USA) started his presentation with an observation that system design and development projects cycle between two phases: an exploratory/experimental phase, and consolidation/formalization phase. In the exploratory phase, problem-oriented (application-oriented) solutions are often investigated. The consolidation phase can start when the problems can be classified and the solutions to them are sufficiently well understood to that they can be embedded in the "system." Leach then demonstrated this two-phase concept on two design issues: extensibility and RPC as a communication paradigm. Extensibility is provided via abstract types with extensible interfaces. In the exploratory phase, it is desirable not only to be able to define new object types but also new interfaces. In the consolidation phase, a successful sort of interface is to be chosen

which can be implemented for many object types. In Leach's view, the RPC design represents an *integration point* The exploratory phase must determine the suitable invocation and binding paradigms, and consequently the RPC interface. Since the RPC interface and client interface implement quite similar abstraction, the respective implementations should be able to share (part of) the interface specification. However, most software engineering methods and tools do not support this sort of integration.

David Clark (MIT Laboratory for Computer Science, USA) focused mostly on his experience with implementing TCP/IP, where TCP/IP is to be used as a general-purpose communication facility in heterogeneous environments. Starting with the question "What is wrong with distributed systems," Clark gave the response: *performance*. He then showed how data copying, marshalling of arguments, and the operating system overhead make it impossible to match the speed of the present and future networks. Clark concluded with two observations: "To increase bandwidth, you must spend money. To decrease latency, you must outwit reality."

Lindsay Marshall (University of Newcastle, U.K.) reported on his experience with the Newcastle Connection, a distributed UNIX system. He brought up again the problem that there seem to be very few *real* distributed applications. In fact, he suggested that it is time to reconsider the "link" to mainframes. Some of his negative experience stems from the fact that users will always try things the designers do not expect. Finally, he concluded with a "call" for new ideas.

Sape Mullender (CWI, Netherlands) discussed the lessons learned from the Amoeba project. The asynchronous request/reply communication paradigm used in the original Amoeba turned out to be too complex, and was replaced by a synchronous call with at-most-once semantics, supported by light-weight tasks. The at-most-once semantics proved very useful in designing fault-tolerant services. The ports and capabilities of Amoeba appear to be a sufficient protection and authentication mechanism; however, ports form a flat name space, which does not scale up well. In the conclusion, Mullender remarked that *good* distributed systems work requires a large team.

General discussion

The discussion focused on three issues: how to achieve concurrency, what are the real distributed applications, and the performance problems. The question why it is difficult to achieve concurrency triggered a lively discussion, which, however, did not bring any new insights. On the issue of distributed applications, Hermann Kopetz (TU Vienna, Austria) remarked that the discussions at the workshop had so far ignored real-time systems, where the distribution is dictated by the application, and which exhibit natural concurrency. The performance question was discussed in the context of generality vs simplicity argument. David Clark maintained that simple protocols do not perform well in complex networks. David Cheriton (Stanford University, USA) reinforced Clark's observation that low latency is hard to achieve, even with simple protocols. He raised a point that although low latency is very important in distributed systems, people working in the area are constantly focusing on improving throughput. Roger Needham added that latency increases with buffering; simple protocols that do not require buffering are thus the potential solution. Greg Andrews (University of Arizona, USA) objected that too much emphasis on performance hinders usability. Finally, Andrew Herbert (ANSA Project, U.K.) responded to the closing remark of Sape Mullender: "it is hard to manage a large research team!"

Tools

Chaired by Andy Tanenbaum

Distributed Systems Research at Arizona

Greg Andrews, University of Arizona

The first part of the talk described three related projects: Saguaro, RS, and MLP. Saguaro is examining a variety of mechanisms for distributed operating systems including a type system. SR is a distributed programming language and is being used to implement Saguaro. MLP is a system for developing mixed-language programs containing procedures written in different languages and possibly executed on different machines, MLP uses the Saguaro type system. The second half of the talk examined two key aspects of the latest version of SR: resources and communication primitives. Resources are parameterized, dynamically created modules that export operations and contain one or more lightweight processes. Operations are involved by synchronous call or asynchronous send; they are serviced by creating a new process or rendezvous with an existing process. Concurrent invocation is also supported.

It was claimed that all these mechanisms have proved to be useful. The talk concluded by identifying current research activities, which include language mechanisms for fault-tolerant programming and for dynamically linking new programs into an executing program.

Discussion

Cheriton:	How does an asynchronous sender get a reply?
Andrews:	By passing a capability to the server. This is also how upcalls work (by passing capability to the kernel).
Cheriton:	Do you have group communication?
Andrews:	Similar to V, except embedded into the language.

Building an Application in Argus

Mark Day, M.I.T.

Argus is an integrated programming language and system for constructing distributed programs. Its key concepts are guardians that encapsulate resources, and atomic actions that help mask the effects of concurrency and failures. A distributed implementation of a mail

repository has been built to gain experience with a sizable distributed program. The implementation of the mail repository also required the use of Argus for replicated data and dynamic reconfiguration.

The basic elements of Argus work well; guardians, actions, and communication by remote procedure call are all effective for system structuring and implementation. Furthermore, the performance is quite reasonable. However, there are also some significant problems. Deadlocks arise often enough to be aggravating, and Argus does not provide any form of deadlock detection. Retrying an action is a useful concept, but is difficult to express. User-defined atomic types provide useful functionality, but are difficult to build and debug. Both replication and reconfiguration are difficult to express cleanly. Reconfiguration requires explicit manipulation of guardian interfaces, avoiding the default use of guardian interfaces. It is difficult to encapsulate replication algorithms in a way that allows simple replacement of the replication algorithm. The replication algorithm is encapsulated within the implementation of a replicated object's type, but is spread through the operations of the type.

Discussion

Naffah:	What about performance? Is it that bad compared to centralized systems?
Day:	No, comparable.
Naffah:	Does replication slow down your system?
Day:	Yes, but unknown how much, since current implementation is running without replication.
Svobodova:	Is replacement of guardians a safe operation?
Day:	Yes, it's done with recoverable transactions.
Krakowiak:	Thinks guardians are too big.
Day:	Guardians could share parts, but that makes it harder to debug. Could have bigger machines.

The CONIC toolkit for building flexible distributed systems

Alan Dempster, Imperial College

Conic is a language based toolkit for building distributed systems for embedded applications such as factory automation, telecommunications, process monitoring and control.

The Conic programming language is used for individual software modules. It provides extensions to ISO Pascal giving lightweight processes and request/reply messages. Messages refer only to local exit and entry ports giving configuration independence and allowing reuse of modules. The Conic configuration language is used to specify a system by creating instances of modules and linking exit and entry points. Run-time supports for systems built from Conic modules is provided by implementing a simple virtual machine on any combination of hardware with or without a native operating system. The virtual machine implements process management and communications.

The main advantage of this approach is flexibility. Currently systems are configured in advance of execution. However, facilities are being developed to allow dynamic reconfiguration within a running system. This will have minimal impact on those modules

not directly affected by the change. Changes will either be generated by interactive operator intervention or in response to programme events such as detectable failures. The Conic approach also allows potential use of other module programming languages perhaps more suited to specific tasks. Hence a knowledge based approach might be used to detect system failures and action a change.

Discussion

Bhargava:	Is your system implemented?
Dempster:	Yes.
Casey:	Can messages be lost during reconfiguration?
Dempster:	Yes, but not often. Configuration changes made in a non-normal environment.
Marshall:	What we want to say instead of distributed application is distributable application.
Bhargava:	What reverse operations do you have?
Dempster:	Only the inverse of linking.
Herbert:	Is reconfiguration restricted? Is it possible to start a new module with new parameters?
Dempster:	No. Yes.
Svobodova:	On line expert would be useful in this area.

Formal Specification as a Tool for System Design and Documentation

Roger Gimson, Oxford University, Programming Research Group

We use 'Z', a formal specification language, as a tool to design and document distributed system servers. Producing a high-level specification of a server makes it easier to discuss alternative designs and to concentrate on user requirements while avoiding unnecessary implementation detail during design.

Clear presentation of specifications, including the formal 'Z' description to complement the more conventional informal text, in the user manuals of our servers provides an unambiguous, but readable, 'contract' between user and implementor.

Avoidance of implementation bias has encouraged us to design servers having some novel features. In our low-level block server, a user must provide an expiration time on block creation (after which the block will cease to exist) and is charged accordingly. A refund may be obtained if the block is destroyed before its expiration time. This has been successfully implemented and shown to be acceptable to users.

Discussion

Nessett:	Can you specify non-functioning servers?
Gimson:	Yes, if this is part of the user's view.
Kopetz:	Can you specify time-outs?
Gimson:	Not done yet, but there are several ways to do it. It must be part of the application.
Lohr:	Can you specify parallelism, blocking transactions, etc.?
Gimson:	Not done for the sake of simplicity. CSP is probably more appropriate.
Day:	How do you specify mutating objects (e.g., when overwriting a file)?
Gimson:	No problem, since most operations are atomic.

The DIADEM System – A Tool for Distributed Application Interdependent Maintenance

Ralf Guido Herrtwich, Technical University of Berlin

Today's software development and maintenance tools concentrate on supporting the implementation and administration of software systems. They fail to address the problem of software installation. The DIADEM System (DIADEM = DIstributed Application interdependent Maintenance) developed at the Technical University of Berlin is intended to fill this gap. It allows failure-atomic remote maintenance of applications in a distributed system. Particular support is given for applications which run on a large number of hosts and for applications which are themselves distributed so that the maintenance of different hosts has to be coordinated. The DIADEM maintenance approach is inherently centralistic: one entity on the development site (the maintainer) controls several entities on the production sites (the maintainees). An overload of the maintainer is prevented by delegating the control of independent maintenance processes or by constructing a maintenance hierarchy which - unlike nested transactions - delegates communication activities rather than transaction control.

Discussion

Levine:	It is a distributed application, but can it also maintain centralized applications? too.
Herrtwich:	Indeed.
Levine:	Doesn't believe that you can update all hosts atomically. Was this the goal?
Herrtwich:	Yes, an entire update should be completed after several hours.
Zeletin:	Does it scale up to a 1000 IBM PCs?
Herrtwich:	Yes, because of maintenance hierarchy.

Svobodova: But that doesn't decrease communication.

Herrtwich: Communication isn't the bottleneck.

Svobodova: Doesn't the overhead of atomic transactions become too large for system consisting of a 1000 IBM PCs (instead of the current 30)?

Herrtwich: Future experiments will show, but he has faith.

Svobodova: What was clumsy about using ISO?

Herrtwich: It specifies things we didn't need.

Naffah: Did you have your own RTS?

Herrtwich: X.25 and Teletex implementations.

Nessett: How do maintainees authenticate themselves?

Herrtwich: Not done yet, but we need it.

General Discussion

Gealy: Looking for tools that help in overview of complexity. Tools for social machines? Cooperating machines are complex.

Herrtwich: No one has one.

Tanenbaum: Do monitoring tools exist? (No one had any.)

Gimson: Do tools for hiding complexity exist?

Saltzer: Looking for an anti-entropy tool to help in decaying software (when system grows, some things stop working).

Droms: Happens when the OS is no longer controllable.

Birrell: Is bothered by it too. Entropy didn't exist in centralized systems because they were managed by central operator.

Debugging

Chaired by David Clark

This session was to focus on the issue of how to debug a distributed system. Debugging of distributed systems has presented serious practical problems, because of the remote, asynchronous and uncontrolled nature of these systems. There were four speakers in the session, with discussion following as usual.

Jean Bacon, from the Computer Laboratory at Cambridge University, described PILGRIM, a debugger for the Mayflower system, a debugger designed and implemented by Robert Cooper. The Mayflower system is a distributed programming environment; the language is CLU, and the remote invocation mechanism is RPC. The debugger for this environment, PILGRIM, provides a source level view of the distributed computation. The goal of the debugger was to work in the normal execution environment, rather than a special debug mode or simulation. For example, the debugger has control of the clocks, so that if one component of the system reaches a breakpoint, all the modules see time stop in a consistent manner. As a result, time does not go faster or distort during debugging. Because the debugger is a part of the normal environment, it is always available, even in production code. A client program may debug a call of a production server without disruption of the server. The only effect on the server is that the clock for that client slip with respect to real time. Mayflower provided substantial support to realize this facility.

Dieter Haban, from the INCAS Project at the University of Kaiserslautern, West Germany, described the Distributed Test Methodology of the INCAS project. The INCAS project is an environment for distributed programming, using LADY, a Language for Distributed Systems. A number of processes are gathered together in a LADY-team, which has an input port and an output port. The LADY support system provides a logical bus which connects these ports together. To monitor and debug this environment, each physical node is provided with a specialized Test and Measurement Processor (TMP), which attaches to the bus of the node. The TMP is a general processing element, with 68020, memory, I/O and communications, which is used to monitor the bus. Since messages between LADY teams are specially defined on the physical bus, the TMP can track these messages. The result is that monitoring is a permanent activity of the operation phase, with a monitoring overhead of about 0.1%. The INCAS environment supports a particular Distributed Test Methodology with four phases, which tests in turn one team, all teams on one node, the complete system, and then monitors the operational phase.

Steve Vinter, from BBN Laboratories in Cambridge, Mass, described the Cronus system, which is an object oriented distributed system of clients, servers and objects. Cronus is a small IPC-based kernel which provides a basic set of system services including interprocess communication, authentication, symbolic naming, and a file system. The goal of Cronus is to exploit the distributed system traits of availability, reliability and scalability to support large, complex distributed applications that can evolve and scale. Cronus attacks debugging with a

number of techniques. It partitions the problem into those which are local to a node, which are isolated to the native debugger for that machine, and those which relate to the distibution. Cronus provides a debugging machine which can access several nodes at once. For distributed problems, Cronus supports dynamic logging, tracing, and structure snapshots. It attempts to avoid (rather than diagnose) distributed problems by containing distributed communication within interface modules which are generated automatically, so that the system can insure semantic correctness. The Cronus debugging interface provides a structured way to review message delivery and instances of objects. It does not provide the illusion of a global clock controlled by the debugger. If a native debugger halts one component of the system, the other parts see time continue.

Larry Wittie, from SUNY, Stony Brook, New York, described the BUGNET distributed Debugging System. BUGNET provides an environment in which distributed processes communicate via messages, with interprocess calls like Thoth. The debugging environment captures interprocess interactions, along with their local time, and has a checkpoint algorithm which permits the state of any process to be rolled back to the state at the time of a message. This permits the local behavior of any process to be recreated by replaying the messages to it. The user can interact with this via a convenient graphical interface, which permits him to monitor local states and global activity, and to replay, correct and retest local code. This scheme does not depend on a global clock which can be controlled, but is based on message arrival logged with the local clock of each node.

The talks and questions after each of these talks, as well as some of the general discussion, suggest that the major design issue in a distributed debugger is the approach to control of time. The four talks reveal an interesting taxonomy. One approach (Mayflower) is to control the global time and stop the global clock when any one node stops. This requires a high degree of homogeneity, but makes system behavior easy to understand. Another approach is to let the clocks run, and only do passive monitoring (INCAS). If sufficient information is gathered, this may permit the recreation of the environment of a isolated part of the system (BUGNET), which permits breakpoints of a node without a global clock. A final approach (Cronus) is to let the clocks run free. This can cause problems; the Cronus system reported problems with protocol timeout occuring during a debugging session. In the discussion, the point was made that the real behavior of time is an important aspect of triggering or reproducing bugs, and any system that does not reproduce or simulate actual time is going to create headaches.

Other points in the discussion were:

- A tool is needed to damage the messages and trigger the recovery mechanisms.
- Reliability measures hide bugs. They should log their activity, and be disabled during selected test.
- Many bugs in distributed systems are performance bugs. By the previous point, recovery mechanisms turn functional bugs into performance bugs. Since protocols are robust, even serious functional bugs can be masked, with awful performance implications. Without a monitor tool that can display real time, these are hard to detect or isolate.

Mechanisms

Chaired by Jerome Saltzer

Gothic

Jean-Pierre Banatre

Jean-Pierre Banatre presented an overview of an object-oriented system named Gothic, a follow-on to the Enchere distributed system. The main item described was a new, hardware-based device to provide a one megabyte high-speed stable storage, in support of higher-level atomic transactions. The talk assumed familiarity with the predecessor system, Enchere; the questions from the audience suggested that many present were not familiar enough with that system to appreciate the differences in the new approach.

Discussion

Lohr:	Can you give examples for the use of multi-functions? And more details on the implementation.
Banatre:	Example of multi-function use is updating replicated objects. RPC in Gothic and process management. Designed implementation of multi-functions on multi-processor.
Zeletin:	Do you mean multi-function or multi-functions?
Banatre:	Multi-function.
Andrews:	Isn't it the same as replicated procedure call?
Banatre:	No! (Never became clear why.)
Svobodova:	How large is your stable storage?
Banatre:	1 megabyte, but this wasn't a problem yet.
Geihs:	Are there restrictions on "n" and "p"?
Banatre:	No.

Hints

Roger Needham, Cambridge University

Roger Needham explained what he called a half-baked idea, involving a systematic approach to dealing with hints that turn out to be wrong. This systematic approach is to always choose one of three courses:

1. Quietly recompute the value and use it, ignoring the hint.
2. Complain to the supplier of the hint and refuse to do anything.
3. Recompute the value, use it, and let the supplier know.

The third course seems most useful, and calls for standardization of a response message.

Discussion

Marshall: Possibly it'll do something you don't want? Don't like the idea (reminds me of error correcting shells).

Needham: No! Something is just out-of-date, but it's usually good information. Results checked in outer loop.

Real-time Systems

Hermann Kopetz

Hermann Kopetz expanded the usual range of topics of operating systems specialists by discussing the special characteristics of real time systems, with examples such as a steel rolling mill. The basic message is that the people doing operating systems generally don't know the requirements in the real-time arena; those requirements are quite different from non-real-time systems. For example, a correct value that is computed too late is as bad as a wrong value. Further, if a processor doesn't have enough cycles to keep up, installing a faster processor doesn't necessarily solve the problem; doing things in the right order to meet deadlines is equally important. Another interesting aspect of design of real time systems is that they usually start from the beginning by precisely identifying the scope of possible failures.

Discussion

Nessett: Do you really get high reliability in distributed systems? More nodes would seem to imply less reliability.

Kopetz: having redundancy.

Clark: Reliability depends to a great extent on partitioning.

Kopetz: Agreed.

Clark: Do you really want a distributed rolling mill?

Kopetz:	This is a research topic.
Zeletin:	Please say more about emergency, hard, and soft categories; implies prioritization.
Kopetz:	Priority only solves part of the problem. Time rigid scheduling will solve the problem. Statically planned ahead.
Leach:	Sounds like there are two phases, one where there is a central decision maker and then other drivers.
Kopetz:	Yes.
Leach:	What is high load?
Kopetz:	High load is derived from minimum time between transactions and maximum transaction time, so it is not CPU load. The system should still work under high load.
Shapiro:	What about an emergency stop?
Kopetz:	Not a good idea (plane transparency).
Marshall:	How do you know what the high load limit is?
Kopetz:	That depends on specs not the system design.
Svobodova:	Did you study how useful existing distributed systems might be to you?
Kopetz:	Yes, and the answer is "not very". Not much available for real time processing. Also, they need static, not dynamic processes.
Levine:	How do you do debugging?
Kopetz:	By monitoring, slowing down internal clock in cluster.

Multicast

David Cheriton, Stanford University

David Cheriton delivered a short essay on why Multicast is a useful system design tool. His argument started from the position that ability to send a single message to a list of recipients is a (perhaps the) fundamental operation of distributed applications. He went on to point out that the usual objections to multicast arise from defining both the implementation and the semantics of multicast wrong; using broadcasts (and having uninterested parties throw unwanted packets away) gives multicast a bad name. Perfect request/response reliability isn't required, only reliable delivery of the initial request. The talk ended with an impassioned plea for multicast implementations in every network, at every level. A lively discussion ensued.

Discussion

Clark:	What is the difference between multicast and sending a list of messages?
Cheriton:	Multicasts are addressed to a group.
Clark:	Maybe we should look for alternatives to multicast?
Bacarisse:	Why doesn't it need to be reliable?
Cheriton:	Sometimes reliability is appropriate, not always.
Casey:	This is analogous to logical addressing.
Cheriton:	No, logical addressing is just an application.
van	Sometimes you want to have processes in a group do different things with a received multicast (e.g., distributed chess). How do you do that?
Cheriton:	Either use a long message with parts intended for particular processes, or use separate messages instead of multicast. It is an interesting question to see which is more optimal.
Nessett:	Why multicast as a network vs. application service? Is unicast more expensive?
Cheriton:	No. Physical broadcast is available at low level. Needs to be taken advantage of there.
Vinter:	What about the layering problem?
Cheriton:	Given this as building block, application can add reliability if necessary.
Clark:	The problem with multicast is that it can damage a lot of machines. How do you prevent that?
Cheriton:	Use transport level gateways as fire walls.
Levine:	Not a good idea to scream to a roomful of people "What time is it?"
Cheriton:	Right, multicast is a fallback position.
Mullender:	I just send a message to one and that's good enough.
Cheriton:	Like in South Africa. Too oppressive.
Marshall:	The Post Office doesn't offer multi-cast. Junk mail is multicast. Is that desirable?
Cheriton:	The alternative is worse.
Leach:	Group may get to be too large–do you have size control?
Cheriton:	Yes, if there's a firewall.
Levine:	You want to prevent groups to be joined by "bad persons."

Security

Chaired by Klaus-Peter Lohr

Reasoning about authentication

Andrew Birrell, DEC Systems Research Center

At DEC SRC, a method for formal reasoning about authentication protocols is being developed. The method should help in constructing and understanding authentication services for large distributed systems. Andrew Birrell gave an introduction, using a variant of the Needham/Schroeder protocol as an example. Secure channels are composed by repeated application of a forwarding rule for authentication messages. As a consequence, precise statements can be made about who knows what and who trusts whom.
Birrell suggested using relative names for identifying principals in a naming hierarchy. This stirred up a debate on benefits and problems of hierarchical naming in general (Cheriton: "I don't believe in world-wide hierarchical naming") and relative naming in particular - without resolving the issue.

Authentication in heterogeneous networks

Dan Nessett, Lawrence Livermore National Laboratory

Dan Nessett dealt with authentication in heterogeneous networks of separately administered systems. At LLNL, they are looking for a solution that wouldn't burden the user with lots of different log-on formats, passwords, etc. while keeping changes to the existing system minimal.
Nessett views the network as a collection of authentication domains, each with its own log-on procedure. If a user wants access to a system in a different authentication domain, her or his log-on request first causes an authentication request to be sent to an authenticator (which may reside in a different domain). The authenticator checks the user's identity and sends an authorization request to the target host (possibly again in a different domain). Finally, the host sends a reply (ok or rejected) to the user.
The discussion centered around security threats. The need for message encryption is obvious. Also the terminals have to be tamper-proof.

Sparse capabilities in the Amoeba system

Andy Tanenbaum, Vrije Universiteit Amsterdam

Using capabilities in the Amoeba system was the subject of Andy Tanenbaum's talk. Amoeba is a message-based system; operations on objects are initiated by sending messages to servers. Objects are identified using capabilities. Capabilities are cheap in Amoeba: their integrity does not depend on a heavy-weight mechanism–they are just sparse.

In addition to object id and access rights, a capability carries a 48-bit port name for the server and a 48-bit random number that is also stored with the object. If the access rights are to be protected as well, they are encrypted (together with a constant) using the object's random number as key (There is also a variant that allows rights restriction without intervention of the server; one-way functions are used). Revocation is easy with this scheme (the random number stored with the object is changed).

Liam Casey pointed out that comparing capabilities with different access rights is impossible. Roger Needham mentioned the similarity between the Amoeba approach and his 1979 proposal of using encrypted capabilities for file protection.

Upcoming Approaches

Chaired by Walter Tichy

This session included four presentations on unusual approaches to building distributed systems. Marc Auslander described a virtual, distributed memory with mapped files. David Gelernter presented Linda, which uses a large, logically-shared, associative memory for communication. Toby Bloom discussed the Common System, an attempt to provide program composability across widely differing systems via common type definitions. Finally, James Peterson introduced RADDLE, a language for describing interprocess communication. Although these four approaches are quite different, they can be viewed as experiments trying to identify the common ground that makes communication possible. The first two approaches use some form of shared memory for interaction, the other two propose a common language. Further experimentation will show haw these novel communication paradigms compare to the more established RPC-paradigm.

Distributed Data Base Memory

Marc Auslander

Marc Auslander (IBM T.J. Watson Research Center, U.S.A.) reported on building a distributed, shared memory for implementing distributed data bases. The basic idea is to map files as segments into virtual memory, and to provide kernel facilities for locking and atomic update of those segments. By sharing the virtual memory among several CPU's (without shared physical memory), distributed application programs become simple, because distribution is essentially "free."

The shared memory with mapped files is operational on a central IBM RT/PC. Presently, the kernel is being extended to manage shared memory across a collection of CPU's. Distributed and efficient versions of the page handler, the concurrency controller, and the log and restart mechanisms must be designed. These components must also be resilient in the face of failures.

Discussion

A number of questions were raised regarding logging. Auslander briefly discussed the relative merits of central logging, logging per machine and transaction, and logging per machine and segment. He claimed that his choice, a central, replicated log, was adequate because of its reliability and simplicity. Log traffic is reduced by using large granularity memory maps (128 bytes). Auslander also explained that the system is not designed to handle network

failures, long running transactions, or long term failures. Replication will be supported by volume.

Linda

David Gelernter

Linda, discussed by David Gelernter (Yale, USA), is a programming environment for building parallel applications for both distributed systems and multiprocessors. The basis of Linda is a logically-shared tuple memory through which processes communicate. Processes never exchange messages directly. Instead, a process with data to communicate adds it as a tuple to the shared memory, and a process expecting data seeks it, likewise, in that memory. There are only three operations defined for the shared tuple memory: OUT adds a tuple, IN removes one, and READ examines one without removing it. OUT never blocks. Both In and READ provide a template matching facility for finding a suitable tuple; they block if none is available.

Linda's facilities are most suitable for applications where the producers of data do not care which processes receive it and at what time. The classical applications are those in which replicated worker processes draw tasks out of, and drop new tasks into, a pool of tasks. However, using the template matching facilities, all other communication primitives, including RPC and streams, can be simulated easily. Linda's tuple memory is coarse-grained enough to be supported efficiently without physically shared memory, i.e., on distributed systems.

Discussion

The ensuing discussion centered on the differences between RPC and Linda. RPC has built-in flow control to prevent buffer overflow, whereas tuple memory overflow in Linda must be handled by the application. Prioritization of access to Linda tuples is also handled by the application, for instance with a special process that watched tuple memory, or by building priorities into the tuples themselves. If partitioning of tuple memory is desired, then the application has to provide that also. Existing applications of Linda include Matrix multiplication, LU decomposition, and VLSI simulation.

The Common System

Toby Bloom

Toby Bloom (MIT, USA) presented the Common System project. The goal of the project is to provide program composability across multiple languages (List, C, Argus) and systems (VAX-Unix, Symbolics List Machines, TI Explorers). The following forms of communication will be supported: (1) blocking RPC, (2) message sending with reliable delivery and sequencing, but without response, and (3) fast, unreliable, asynchronous message sending.

Arguments are passed by encoding/decoding them into/from Common System-defined types by senders and receivers. The Common types are registered in the Common System and consist only of an external representation and a set of encode/decode operations, plus some documentation.

Hence, the Common System approaches the heterogeneity problems by providing global types and requiring all arguments to be passed as values of these types. The global typed descriptions do not include type-specific operations other than encode/decode, because different languages and systems may provide different operations for the decoded types.

Discussion

The problem of accommodating heterogeneity in computer networks was generally recognized as an important one, but the audience was clearly divided into two camps. The skeptics were of the opinion that the languages, systems, and applications were too disparate to permit a satisfactory solution. The optimists (including Bloom) were of the opinion that useful program composability and communication can be achieved by careful definition of the global types and interfaces.

The RADDLE project

James Peterson

James Peterson (MCC, USA) presented RADDLE, a design language for specifying communication among potentially distributed processes. The central concept in RADDLE is the N-party interaction among processes (called "roles") within a process group (called a "team"). N-party interaction is a generalization of the (2-party) rendez-vous with conditional acceptance of the entry calls. Conceptually, a N-party interaction takes place all at once. Petri Nets are used to model control flow. A distributed RADDLE execution environment is planned.

Discussion

In response to questions, Peterson explained that N-party interaction is implemented as multiple, 2-party interactions, but that N-party interaction is conceptually cleaner and easier to understand. N-party interactions do not nest. The value of Petri-net modelling was questioned on the grounds that Petri nets provide no abstraction, and that graphical representations of complex systems are unintelligible. Peterson hopes to modularize the nets in some appropriate way and pointed out that machine processing and checking of the representations in one of the major advantages of RADDLE.

Making Distributed Systems Work

Chaired by Sape Mullender

The final session of the workshop was a very lively discussion session. The programme committee had prepared a few discussion items, which are indicated as if spoken by the chairman (which they were, actually).

Chairman:	Have we made any progress during the last two years?
Nessett:	It is harder to get things into production than we expected, even when the concepts are simple. It just takes a lot of time.
Gealy:	The support problem is large, that takes time.
Mullender:	We are just trying to get something to work in the first place.
Cheriton:	We tend to ignore an important aspect of progress: we do not make giant steps, but we are getting details right. Since the last workshop we have a better idea of those details.
Bhargava:	An analogy can be made with automobile industry: the automobile is complex, but the user interface is not. We must try to get our interfaces simple, but whatever complexity is needed to work well is done.
Kopetz:	Gives the example of an airplane pilot who is flying along but does not know where he is heading; what is the goal of distributed computing, what is the application?
Chairman:	Do we need distributed applications to justify distributed systems?
Tanenbaum:	The justification of our work is that we can make systems more reliable and cheaper than without distribution.
Clark:	That is not sufficient, it can be different next year.
Casey:	We should not forget that in some cases a Cray II can do our applications better.
Clark:	We must distinguish between parallel and distributed applications.
Shapiro:	Many people seem to think that no 'real' distributed applications exist. But we have seen one in this workshop (referring to the talk by Kopetz). I think there are many more, but we do not know the people that use them.
Levine:	So, we should try to solve real problems, and a way to do this is using distributed systems. Distributed systems started out being a solution to a problem. After that is solved, we can look into further implications.

Tichy: We have to try applications in order to get things right. A message sending facility is not sufficient.

Marshall: Distributed systems started because we had these machines and this wire, and we wanted cooperation.

Bhargava: There are lots of applications, for example travel reservations.

Svobodova: We need operating systems interconnection. It is not an operating systems problem, but a connecting problem.

Gealy: We will have distributed applications but we do not need to recognize them before we make distributed systems. The architecture for connecting components is what we need.

Cheriton: All human systems are distributed.

Wittie: 3 temptations:
a - they will be connected anyway
b - they will be faster (parallelism)
c - the driving force is the hardware - use it

Levine: There will always be differentiation: the large computer manufacturers are benefiting from it. What we in research are trying to do is to blur the boundaries. Therefore different connection methods are appropriate for us than those between two banks for example.

Svobodova: The reality is separation.

Vinter: So you will agree to connect to the outside using OSI.

Leach: When we build a new system we have two choices: use OSI and connect to the outside world, or write our own and communicate among ourselves

Svobodova: Concept of OSI is important, especially the lower layers.

Nessett: We are making progress, although not dramatically.

Needham: There is an analogy with early time sharing systems: however, they did not ask for applications of those. So we should not worry too much about applications. Future will show two important things: (a) faster LAN's, and (b) multimedia communication, different sorts of technology

Tanenbaum: OSI does not provide RPC, how do we get around it? RPC is useful, but we cannot use it, since it is not in OSI.

Herbert: ISO *has* recognized that RPC is useful. We have to get our technology into the real world.

Tanenbaum: PTT's refused to implement them, so they were thrown out of the standards.

Herbert: It may take a while to send datagrams over the wire, but it will happen.

Cheriton: ISO (the OPEC of protocols) can only stop us for some time, but if military forces are going to use different system, ISO will be no standard. If competition leads to a better alternative that is going to be used.

Marshall: Analogy with audio world, some very bad standards are still used.

Cheriton: But in that case we are not talking about orders of magnitude difference. In the long term situations like these cannot stay.

Casey: We should get our work into ISO.

Gimson: But every standard has to have a timeout.

Cheriton: 10 years from now, the challenge will not be facilitating communication, but restricting it.

Marshall: Some of us are arguing from a privileged position. Do not produce too high a standard.

Clark: Do distinguish between research and standard setting. Standard makers should be somewhere in between researchers and industry.

Birrell: Standards are not for the universities, they will be used by the rich industry.

Kopetz: Do not standardize things that are not understood.

Chairman: Do we need a third workshop and what should be its title and its topics?

Andrews: Yes, especially for people who did not attend this one.

Cheriton: Workshop is very useful for finding out how others are progressing.

Herbert: Not another talk about RPC. What about removing 'distributed' from the title. Lots of people who are doing communication who have nothing to do with communication are not here. Maybe about multimedia applications, maybe a wider scope.

Svobodova: This workshop was incorrectly called: second workshop on Distributed Systems. Hopefully, the third will not be another on distributed systems. But what should it be called?

Bhargava: Very useful to meet people, but we need more people to attend, especially others than just Europeans [less than 50% of the people attending were European, Ed.].

Clark: Focus on innovative processes. Also try to make it more useful for Europeans. Suggestions: multimedia, autonomy, scaling.

Cheriton: SOSP is defined by what people are doing, not restricted. We should choose a wishy washy topic and see what interesting papers turn up.

Tanenbaum: OS is alive, still important to focus on it. The only other option is SOSP. Multimedia is too limited. Let us just say: Operating Systems.

Bacarisse: Scaling is not so important right now, because most of us cannot scale since we do not have the resources.

Herbert: Observation: the title is important. We have to try to get other people involved. Maybe it is helpful to defer the title until after the papers are in.

Needham: Really anybody. Not a narrow area.

Clark: Bring in 'outsiders'.

Mullender:	There is a tradeoff between getting 'good' projects to the workshop, and getting people that can learn something here.
Marshall:	It is gross to have 'insiders' and 'outsiders'.
Andrews:	More parallelism, more multiprocessors. Van Renesse: More controversial ideas, maybe a workshop for rejected papers.
Marshall:	What about the title: support for distributed applications.
Tichy:	We should have a wishy washy title, with tentative topics, and note that others may also be accepted.
Herbert:	Let's stop talking about the next workshop now, and talk about distributed systems.
Svobodova:	Outsiders can pick up ideas from us, but also the other way around. For example, we can learn a lot from people in industry.
Cheriton:	We have to get people in that made effort on their own.
Clark:	Everyone has to bring an outsider along next time.
Herbert:	Let conference attenders express requirements.
Bhargava:	Early September is a good time, before universities are in session.
Gealy:	Maybe chairman is responsible for getting more people involved.
Droms:	Maybe in position papers why research is innovative.
Mockapetris:	Let's have a workshop on matching distributed systems with applications.
Marshall:	Ideas are converging, maybe there are no new ideas.
Cheriton:	Maybe we need very detailed lectures, they can be very interesting.

List of participants

Workshop Making Distributed Systems Work
Amsterdam, September 8-10, 1986

David P. Anderson	*University of California at Berkeley*
Gregory R. Andrews	*University of Arizona*
Marc Auslander	*IBM, T.J. Watson Research Center*
Ben Bacarisse	*University College London*
Jean M. Bacon	*Cambridge University*
J.P. Banatre	*IRISA/INRIA*
Yolande Berbers	*Katholieke Universiteit Leuven*
Orna Berry	*System Development Corporation*
Bharat Bhargava	*Purdue University*
Andrew D. Birrell	*DEC Systems Research Center*
Toby Bloom	*MIT*
Liam Casey	*Bell-Northern Research Ltd.*
David Cheriton	*Stanford University*
David D. Clark	*MIT*
Mark S. Day	*MIT*
A.J. Dempster	*Imperial College London*
Ralph E. Droms	*IBM, T.J. Watson Research Center*
Joel Emer	*Digital Equipment Corporation*
Michael Evangelist	*Microelectronics and Computer Technology Corp.*
Mark Gealy	*Xerox Corporation*
Kurt Geihs	*IBM, European Networking Center*
David Gelernter	*Yale University*
Roger Gimson	*Oxford University*
Terence E. Gray	*UCLA*
Peter den Haan	*Philips Research Laboratories*
Dieter Haban	*Universitat Kaiserslautern*
Andrew Herbert	*ANSA Project*
Ralf Guido Herrtwich	*Technische Universitat Berlin*
Lee A. Hollaar	*University of Utah*
Hermann Kopetz	*Technische Universitat Wien*
Sacha Krakowiak	*Université de Grenoble*
Hanne Larsen	*Norsk Regnesentral*
Paul J. Leach	*Apollo Computer Inc.*
Paul H. Levine	*Apollo Computer Inc.*
Klaus-Peter Lohr	*Freie Universitat Berlin*
L.F. Marshall	*University of Newcastle upon Tyne*

Paul Mockapetris	*USC*
Sape J. Mullender	*Centrum voor Wiskunde en Informatica*
N. Naffah	*BULL TRANSAC Corp.*
Roger M. Needham	*Cambridge University*
Dan Nessett	*Lawrence Livermore National Laboratory*
R. Popescu-Zeletin	*Hahn-Meitner Institut*
Jerome H. Saltzer	*MIT*
Marc Shapiro	*INRIA*
Liba Svobodova	*IBM Zurich Research Laboratory*
Andrew S. Tanenbaum	*Vrije Universiteit*
Douglas B. Terry	*Xerox PARC*
Walter F. Tichy	*University of Karlsruhe*
Carl Tropper	*McGill University*
Stephen T. Vinter	*BBN Laboratories*
John Wilkes	*HP Labs, 3U*
Larry D. Wittie	*State University of New York at Stony Brook*