

DSM-JAVA: FOUNDATION OF A LEAN DISTRIBUTED OPERATING SYSTEM

Oliver Schirpf, Michael Schöttner, Peter Schulthess, Stefan Traub, Moritz Wende
Ulm University (schulthess@informatik.uni-ulm.de)

Keywords

DSM Java, distributed Web-Applications, native Java, Java Compiler, lean Systems Design, Optimistic Transaction Scheduling,

Abstract

Combining atomic transactions, distributed shared memory (DSM) and a typesafe language (Java) might simplify the construction of intranets, of telecooperation systems, of parallel computations and in general the consistent management of distributed data structures. DSM not only simplifies the communication mechanisms between programs and program modules but also the structure of the underlying run-time environment. Read/write conflicts between concurrent transactions on different network nodes can be automatically resolved by the operating system. DSM-Java is the core facility of our native operating system Plurix which might serve as a prototype for „Next Generation Distributed Operating Systems“. As a companion to the native DSM-Java environment a plug-in for web-based DSM-Java is planned.

1. Research targets

Operating System for Distributed Applications

DSM-Java offers intrinsic support for distributed virtual storage. A document in DSM-storage is accessible via regular load- and store machine instructions. The programming of distributed applications is therefore simplified in comparison to traditional approaches which involve message passing and remote procedure calls at the application level. Notably this applies to telecooperation scenarios: when several participants wish to simultaneously access a shared document the consistency of the document may be guaranteed by the operating system and not by means of specific replication protocols in the application program. DSM-Java may also be built on top of existing operating systems and browsers but in this case it suffers from the weight of the imported run-time system.

Data Storage in the Network

All data objects reside in networked DSM storage and user can access their data without awkward reconciliation mechanisms (Coda¹, the Briefcase in Windows). The most current version of an object is automatically presented. This facility may be of interest to software development teams, for inventory lists in a small business environment, for telephone directories and appointment scheduling plans in office environments and many other areas. If data objects are carefully partitioned substantial computations are feasible in the network of workstations.

Lean System

Operating systems along the lines of DSM-Java can omit many functions and concepts which make contemporary operating systems voluminous and clumsy. In particular we omit disk based file operations and the loading and linking of program modules and classes. Applications which only manipulate private data objects are not penalized by the management of distributed storage. By rewriting a native operating system from scratch an efficient management for storage-, runtime- and transactions was achieved. The Java classes required for the operating system kernel are compact and yield short transaction times and a small working set. At the CeBit-fair 1999 a PC-cluster was demonstrated taking less than 60 kbyte of program memory.

2. Operating System for Distributed Applications

Transparent Resource Distribution

Resources of individual nodes should be integrated to present a single image of the the totality of resources in the network. User perceive the same state of their data objects from any workstation they may logon from. Consistency of documents and data bases is guaranteed even if several users manipulate a document concurrently. Large and network spanning tasks should be performed by making efficient use of pooled resources in the workstation cluster.

Sockets and Pipes

At the transport level the data exchange between stations may use sockets, streams or pipes. Communicating processes transmit messages calling primitives such as send and receive (typically TCP or UDP). Messages mostly contain textual data but may also include complicated data structures. These structures are explicitly serialized before being transmitted.

RPC and RMI

Calling methods or procedures remotely (RMI in Java², RPC³) offers greater programming convenience than sockets and streams. Remote procedure calls are very similar to the invocation of local procedures (Disregarding a few inevitable semantic restrictions). Serialisation of parameters and the mapping into an external data representation (marshalling) is automatic. However, the programmer must be in command of a complicated set of development tools. RPCs and RMI generate substantial overhead and are typically slower than sockets and streams.

Distributed Shared Memory

Objects in distributed shared memory (DSM) can be directly accessed by pointers and machine instructions from individual nodes (Fig. 1) – even if initially the objects are not locally present. Several alternative strategies are suggested in the literature⁴ to guarantee the consistency of the shared storage.

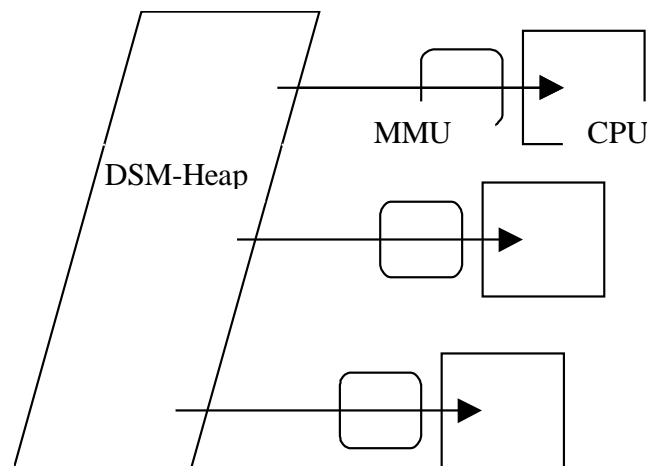


Figure 1 Distributed Shared Memory (HW)

Distributed shared virtual storage may be created either by the hardware alone or by a combination of hardware and software. Examples of DSM systems are ^{5 6 7}. A Java/DSM system with a modified virtual machine is presented in⁸. In our Plurix operating system virtual storage is implemented through the MMU in each node⁹. Typesafe languages such as Java offer the potential to reorganize storage and to protect processes from each other.

Services, especially name services

Name services are provided for example by WINS¹⁰, Corba and the Domain Name System¹¹ in the Internet. Objects in the network may carry a name together with appropriate attributes and may be found by users and clients irrespective of their physical location and address. Typical objects are servers in the net-

work, subdirectories, files, but also single variables exported by programs and classes. ¹² offers an overview of old and new name binding services and concepts. Security services must protect data and the operating system from unauthorized access and manipulation. The distributed nature of the system makes it more vulnerable to attacks and corruption. Ultimately secure distributed systems will emerge from a combination of cryptography, operating systems technology and computer architecture¹³.

Client-Server Scenarios

Individual services are often provided on a separate machine. On a separate machine the consistency of a data base and protection from unauthorized access is more readily achievable than in a fully distributed system. By keeping the partitions of a database on separate machines we arrive a distributed database system with all the regular ACID properties.

3. *Lean Systems and not really lean Systems*

Not really lean Systems

At present the market is dominated by Windows, Unix and the MacOS. Unfortunately these systems are slow, complex and heavy-weight for a number of reasons:

- A plethora of peripherals and driver configurations is supported
- New interfaces are layered on top of venerable and old interfaces
- Devious communication mechanisms are used between entities of the system
- User access rights must be enforced
- Network architecture is not designed into the operating system
- Different operating systems must be integrated into a single system
- Frameworks, compiler, linkers generate voluminous modules
- Modules are replicated in memories, in caches and on disk
- Visually interesting user interfaces absorb processing power.

We do not question the professionalism of software engineers at large. Rather we believe that the forces of the market prevent the discarding of computational traditions and a radically new approach for distributed systems design. The complexity of contemporary operating systems and toolkits makes program development hard. Only a small number of experts is capable of using the abundant spectrum of basically useful functions reliably and efficiently. Education and documentation for programmers and analysts is expensive and time consuming.

Oberon & Java, examples for lean Systems

It is more natural for a university than for an industrial company to attempt a fresh start. The Oberon System¹⁴ developed at the Federal Institute of Technology in Zurich is a good example of a lean system. Building on a few basic types (DisplayFrame, Text, Gadget¹⁵) a very fast operating system is provided, which for many tasks requires up to two magnitudes less resources than leading operating systems. With the exception of its visual appearance the functionality of the Oberon system is very rich.

Java applets and programs require only a limited set of run-time support and carry also the potential of a lean system and of efficient resource utilization if appropriately compiled. The concept of dynamic binding loads only those classes which are actually referenced by the program. In addition Java offers a spectrum of well designed communication facilities. The set of available Java classes is impressive and sometimes overwhelming but their inclusion is optional and the possibility of building light weight systems remains.

For this reason we select Java and a portion of the run time library to build a new lean operating system and to try to provide an alternative method for building distributed systems.

4. *Java Objects in a DSM-Heap*

The basic idea for building a lean distributed operating system is to allocate all objects in a distributed shared memory. On 32 bit PCs a maximum of four gigabytes DSM-storage are managed much like a local

heap. Authorized nodes use regular object references (pointers) to access the DSM-heap. Initially we assume that programs are not capable of forging object references.

Restartable Transactions

Consistency of data structures in the DSM-heap can only be preserved if no thread is given uncontrolled heap access. All accesses must occur in the context of a restartable transaction¹⁶. Write operations to DSM-pages occur to local copies at first. A transaction will either complete all its actions or it may be reset if a conflict with another transaction is detected. A conflict with another transaction arises if either both transactions write to the same page or if a transaction reads a page which is written by another. A losing transaction is automatically restarted. The read/write access pattern is continuously monitored by the MMU and recorded in a read-set and a write-set.

Short transactions are less likely to collide with other transactions because temporal overlap is reduced. We therefore encourage short transactions such as a keyboard entry, a mouse click or the compilation of a module (!). Allocation of local copies for modified pages, short transaction times and small working sets are only feasible in a lean operating environment. This fact among others made it imperative to write our DSM operating system for the native PC hardware platform.

Relocation of Objects and global Garbage Collection

Objects in the DSM-Heap are relocatable to permit memory compaction and to reduce fragmentation of the heap. All global pointer references to a heap-block are chained together and reachable from the heap-block itself (Fig. 2). After moving a heap-block all references can be adjusted. This backchain must be updated for each global pointer assignment and for each pointer variable which is garbage collected.

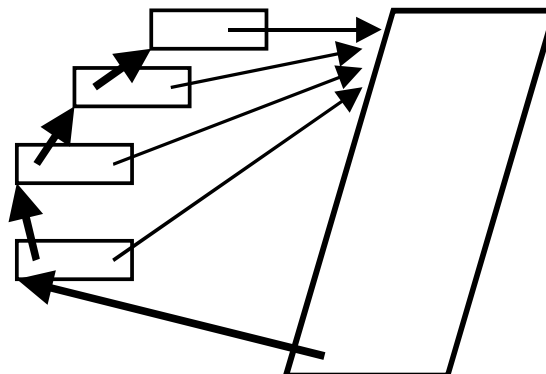


Figure 2: Backchain for references to a heap object

Garbage collection in the DSM-heap is imperative but it is not practical to put all network nodes to sleep before starting a mark and sweep cycle. Rather we attempt to collect one heap-block at a time and allocate a separate transaction for the collection of a small number of heap-blocks. A copying garbage collector attempts to collect groups of objects which exhibit a cyclic reference pattern but which are no longer referenced from live data structures. However, we are still in search of a more elegant mechanism to identify isolated and cyclic garbage structures.

False Sharing

Only minimal performance penalties arise from DSM management if the read/write sets of concurrent transactions do not intersect. However, if competing processes in different nodes access the same object in alternation („Read/Write Sharing“) severe interferences are the result. Pages are repeatedly transmitted and conflicting transactions are frequently restarted.

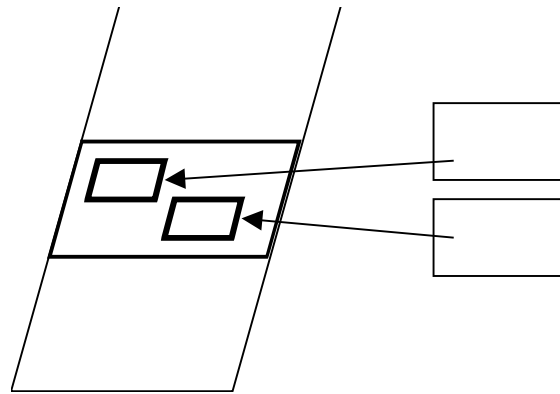


Figure 3: „False Sharing“ Situation

These performance penalties will also arise if the competing nodes access two different objects which unfortunately reside in the same page. This situation is generally known as „False Sharing“ (Fig. 3). If possible one of the two objects should be relocated to another page, thus eliminating the false-sharing situation. To initiate the relocation of false-shared objects appropriate heuristics are required.

5. The Compiler for DSM-Java

Lean Compiler

Commercial Java compilers and compiler generators proved to be unsuitable for our case, because they do not allow for efficient access to the hardware and because of their immodest storage and run-time requirements. Our compiler is itself written in Java and comprises less than 100 kbytes of Java bytecode. Run-time requirements are modest in the form of a „new“-facility and a string class. After being bootstrapped the compiler will run in the target machine and directly compile and link the Java classes into the heap. At the moment we implement a simple codegeneration for Intel 486 machines with a fixed register convention. In a second step we plan to implement a modern codegeneration engine – including dynamic codeoptimization along the lines of ¹⁷.

Proprietary Runtime Data Structures

The DSM-heap holds Java objects and associated run-time structures such as class descriptors, interface descriptors, code segments and constant pools. The heap-blocks include fields to support relocation and efficient memory management. The compiler allocates heap-blocks and initializes references to imported classes and inherited codesegments. Loader and linkage editor are unnecessary because their task is already performed by the compiler.

Generating native Code from Bytecode

A diploma thesis was recently completed¹⁸ describing the generation of native Intel 486 code from Java bytecode. The run-time structures of this compiler back-end are compatible with regular Plurix descriptor formats. As a continuation of this thesis we plan to provide a plug-in module for the netscape browser to execute DSM-Java programs.

6. Operating Systems Kernel

Hardwareplatform and Bootprocedure

DSM-Java directly runs on PCs with Intel 486 CPU or larger. At the end of the BIOS bootprocedure the bootsector from disk is invoked. A small operating system kernel is fetched from diskette, from the serial line or via the network adapter. After loading the kernel additional objects and classes are loaded from DSM storage. The state of the DSM-heap is stored on a disk server, when the distributed operating system is shut down. We plan to complete the bootprocess within 3 seconds but this will mean restricting ourselves to single type of motherboard.

Transactions and Storage Management

Management of the shared virtual storage is done in the kernel. Logical address space is structured as a contiguous sequence of free or allocated heap-blocks. Portions of the logical address space are locally mapped to physical memory pages. For all physical pages appropriate access rights are recorded. A page which resides in the global address space but which is not locally present carries no access rights respectively the access right „extern“ (Fig. 4).

A page which is fetched from the network to be read by a local program will be marked with a „read“ attribute. If a page is modified storage management creates a backup image of the page and marks the page as „write“. The backup image is created during a transaction when a page is first written. When a transaction enters its commit-phase the read- and write-sets are computed and the transaction is now ready to compete with other transactions. Currently we implement the „first-wins“ strategy which means that a transaction which terminates can abort all conflicting unfinished transactions. More sophisticated conflict resolution is recommended however. If a transaction is aborted it is immediately scheduled for a rerun. If the transaction wins („Commit“) the next transaction is prepared and all local access rights are reset to „no access“ („gesperrt“). Figure 4 shows the possible states of a memory page. These states do not depend on whether the page is logically allocated or free.

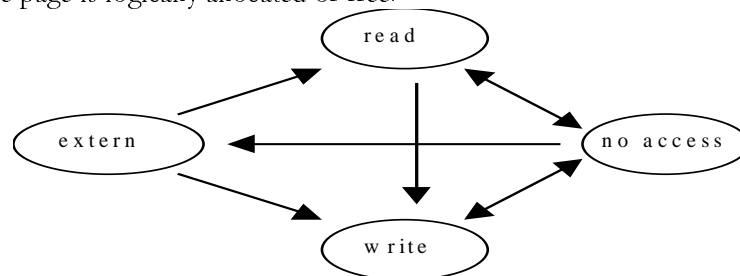


Figure 4: Access rights to a page

Transactions are called from a main operating systems loop and typically last less than a second. For longer transactions the probability of a conflicts increases but the integrity of the system is unhampered. In this time frame the system may accept a mouse-click, a keyboard entry or even compile a module, compress an image and update the screen. Our experience with the Oberon System¹⁹ indicates that very short transactions are achievable. Unfortunately we have not been able to test our system under realistic load conditions yet.

Persistence of Objects

The Plurix operating system does not include a diskfile system. Rather we transfer memory pages between main memory, network and sector-structured disk. Objects are registered in a directory located in the DSM-heap. If an individual station wishes to shut down it can submit its pages to neighbouring stations or to background storage on the disk server. The disk server records all pages transmitted on the network and his view partially reflects the state of the DSM-heap. Background storage is passive and its write-set is always empty. In order not to create a single point of failure and a performance bottleneck the background storage can be partitionned and replicated.

Device Drivers

The kernel includes native drivers for display (S3 Virge ...), for the network adapter card (NE2000), for mouse and keyboard and for hard disk access. The drivers require special precautions to allow for restarting a transaction with the original input and output configuration. Devices which service an interrupt must reside in local storage which is independent from the DSM mechanisms.

Multithreading

Each node hosts a single thread which shares the address space with all other threads in the system. Each node holds a mapping between logical memory and local pages of real memory. The mapping is achieved by the page tables and page directories of virtual memory management. Multithreading within the same node may be implemented by exchanging page tables (including CPU and cache) when switching from one thread to the next. Our current implementation does not support multithreading.

7. Programming the Hardware in Java

Direct Access to the Hardware

A design goal of the Java language was to prevent applets from accessing the hardware. In our case, however, direct and fast access to the hardware was crucial to develop drivers and to implement compiler and memory management. To this end our compiler includes Java extensions to manipulate the hardware. It can generate hexadecimal inline code, it can access memory as a large array of selected container size, it can issue I/O instructions and it may circumvent typecompatibility rules.

Fragmentation of Java Objects

In principle each Java object is allocated as separate heap-block which can reference further blocks for arrays, strings and objects in general. A separate block is created even if the size of the object is known at compile time and the compiler would be in a position to embedd a referenced object directly into the original object. We thus risk to obtain a heap with an unnecessary amount of fragmentation and with many small blocks which are tedious to manage and to collect.

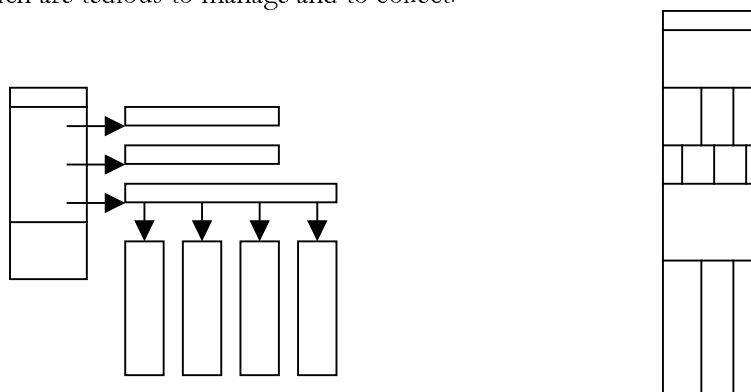


Figure 5: Fragmented vs. linearised Java Object

This object fragmentation in the heap is particularly disturbing because the compiler must set up class descriptors. These contain scalars and arrays which in real Java would lead to complex linked list structures (Fig. 5). The compiler creates a linearized and monolithic class descriptor by bypassing the type rules. We do not include a language feature extension for inline arrays and for embedded objects however.

8. Security Considerations

Globally shared virtual storage makes access to data from other stations simple and straight forward. To enforce access restrictions in such an environment presents an additional challenge and requires a multilevel approach. The implementation of a secure Plurix environment is under study.

Java Typesafety

Java is a typesafe language and does not allow the unauthorized creation of references to foreign data structures without support from an extended compiler. The Plurix class descriptors can not be duplicated by a regular Java program because their structure is not based on Java references. In principle, however, a manipulated compiler can be employed to break the type rules and more comprehensive precautions will be required to protect the system against malicious attacks.

Protected Storage Areas and Authorisation

The typesafety of Java does not offer protection against malicious attacks. Confidential data might therefore be allocated in a protected storage area which is removed from the DSM-Heap. Requests by unauthorised nodes to obtain protected pages are refused. Protected storage objects might be encrypted and sent to a trusted machine at station shut-down.

Objects may be registered in a private directory whose address is only known to the owner or to members of his group. Private directories may reside in protected storage. At login-time cryptographic procedures will send a directory entry point to the incoming user.

Authorized Code

An additional risk is created if an invader infiltrates his code into a remote machine. This code might copy portions of protected storage to DSM-storage and then fetch the stolen data via the regular paging mechanism (Fig. 6). To protect against this type of intrusion it will be necessary to allow access to protected storage to authorized code only. Code might be authorized because it resides in protected storage and carries special hardware privileges (Intel protection ring number).

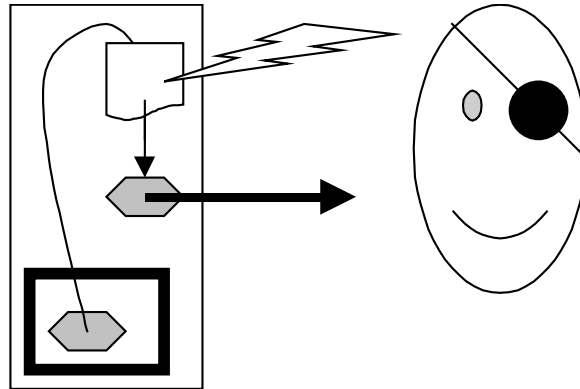


Figure 6: Stealing confidential data from protected storage

9. Work in Progress

The Plurix operating system is a prototypic implementation of DSM-Java and was successfully demonstrated at the CeBit 1999 in Hannover. Three PCs were connected to form a Plurix cluster. A 360 degree panorama image was read from shared virtual storage and displayed on the respective screens. The achieved frame rate was in excess of 100 frames per second. In a next step further components of the system must be completed and the performance of these components must be studied in realistic application scenarios. Our particular interest will be in multimedia telecooperation.

Algorithmic refinements will be required. Currently the conflict resolution between transaction uses the „first wins“ strategy (see above). This strategy is neither fair nor economic in terms of resource usage. The false-sharing syndrome is analyzed further and its resolution shall be attempted using different heuristics. Problems associated with security and recovery must be addressed in depth.

Codegeneration in the compiler should be improved. Faster and denser code will be produced based on an intermediate SSA-representation. Since classes and instances are available in an already linked form further avenues to optimization are open. Processing power for massive code optimization is available from the workstation cluster.

On a longer time-scale we intend to present an alternative to large-scale operating systems. Distribution is envisaged in a similar form as for Linux. We hope to provide additional incentive for commercial IT-companies to create lean and fast distributed applications. This will reduce the effort spent on configuring and operating computing infrastructures. We hope that the computer professionals can eventually direct their attention towards important problems outside of the realm of computer science and open whole new relevant application areas.

10. Literature

¹ Satyanarayanan: Coda: A Highly Available File System for a Distributed Workstation Environment, Second IEEE Workshop on Workstation Operating Systems, 1989

² Java RMI Dokumentation,
<http://java.sun.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html>

³ Bloomer: Power Programming with RPC, O'Reilly & Associates, 1992

-
- 4 A. S. Tannenbaum: Verteilte Betriebssysteme“, Prentice Hall 1995
- 5 P. Keleher: TreadMarks: Distributed Shared Memory on Standard Workstations and
Operating Systems“, USENIX Winter 1994, 1994
- 6 E. Speight & J.K. Bennett: Brazos: A Third Generation DSM System, USENIX-NT
Workshop, 1997
- 7 Traub: Speicherverwaltung und Kollisionsbehandlung in transaktionsbasierten
verteilten Betriebssystemen, Dissertation, Universität Ulm, 1996
- 8 Yu & Cox: Java/DSM: A Platform for Heterogeneous Computing, Workshop on
Java for Science and Engineering Computation, 1997
- 9 Schöttner, Traub & Schulthess: A transactional DSM Operating System in Java,
Parallel and Distributed Processing Techniques and Applications, 1998
- 10 Microsoft: WINS: „The Windows Internet Naming Service – Architecture and
Capacity Planning, Microsoft Corporation, 1996
- 11 Austein & Saperia: DNS Resolver MIB Extensions, RFC 1612, Domain Name
System Working Group of the IETF
- 12 Lupper: Dynamische Verwaltung globaler Namensräume, Dissertation, Universität
Ulm, 1997
- 13 L. Keedy: Making Computers Secure, Vol. 1: a secure computing environment,
republication edition, Universität Ulm, 1997
- 14 Wirth & Gutknecht: Project Oberon, Addison-Wesley, 1992
- 15 J. L. Marais: Design and Implementation of a Component Architecture for Oberon,
PhD thesis, swiss federal institute of technology Zurich, 1996
- 16 Camelot & Avalon: A Distributed Transaction Facility, Morgan Kaufmann
Publishers, Inc., 1991
- 17 Michael Franz: Adaptive Compression of Syntax Trees and Iterative Dynamic Code Optimizati-
on:Two Basic Technologies for Mobile-Object Systems, in Jan Vitek and Christian Tschudin (Eds.), *Mobile
Object Systems: Towards the Programmable Internet*, Springer Lecture Notes in Computer Science, No. 1222,
263-276, Feb. 97
- 18 Andreas Böhm: „Codeerzeugung für VVS-Java aus Byte-Code“, Diplomarbeit, Universität Ulm
- 19 Wirth & Gutknecht: Project Oberon, Addison-Wesley, 1992