# OPC – Making the Fieldbus Interface Transparent

Dipl.-Ing. Thomas Hadlich, Dipl.-Ing. Thorsten Szczepanski

ifak system GmbH, Steinfeldstr. 3, 39179 Barleben

**Abstract.** The performance of fieldbus systems has been studied more or less thoroughly. With the advent of high speed fieldbus systems as PROFIBUS-DP the bottleneck of systems moves from the communication to the application, especially to the interface between both. Two requirements are placed on such interface in general. First it should be based on an open, accepted standard where secondly it should support real-time operation. As the PC becomes more and more important even in the realm of industrial automation the quality of interfaces based on Microsoft Windows technologies gains importance too. The first solution provided with Windows 3.1 was the Dynamic Data Exchange (DDE). Although an open an finally widely accepted standard its communication performance was not satisfactory. The development of the OPC (OLE for Process Control) architecture aimes at solving the question of performance while providing an open interface standard between applications. Although the openness of this solutions is not debatable, the performance issue has to be investigated. This paper contributes to this by exploring the performance of an implementation of the OPC interface standard.

## 1 Introduction

In every area of industry there is a move from proprietary solutions to open, vendor-independent standards. As well as reducing costs it allows the choice of components according to their performance and reduces the dependence on suppliers. An impressive example for this move can be found in industrial communications. Today’s prevailing systems are all based on open, vendor-independent fieldbus standards. Such open standards were needed in the domain of applications too. The specification of the Dynamic Data Exchange (DDE) protocol provided a first solution for the data exchange between MS-Windows based applications. The main drawback of this solution was its low bandwidth. Therefore DDE was not very well suited for real-time systems, a major requirement as far as automation systems are concerned.
The performance of fieldbus systems has been the subject of countless studies. As revealing as such studies may have been, from the user point of view it is the overall performance of the system consisting of application and communication stack that matters most. This paper presents the results of a performance study of an OPC/COM hierarchy. The investigation of OPCs performance capabilities was based on the especially developed suite of applications. It stands in proxy for a probable software chain from a process control system (OPC test client), OPC server (OPC test server) responsible for the data acquisition from possibly one or more data sources. Based on this, an environment which allows to maintain a given real-time criterion was determined. This environment was defined in terms of the workload of the system as

well as a specific netload on the Ethernet connection between the computers involved. The client-/server model as well as the publisher-/subscriber model (both supported by OPC) are taken into the consideration. In the next step the different load scenarios are determined for the case of a concentrated application (one PC running the data server as well as the client application) and a distributed application. The relevant load parameter are changed from best-case to worst case conditions. The presented results are reduced to the most important values system reaction time (client-/server model) and message interval (publisher/subscriber model) respectively. Based on the gained information the definition of a so called working environment for the case of concentrated and distributed application took place. The working environment was set especially with the observation of a given message interval in mind. The workload (resp. netload) which guaranteed a 99% probability of sustaining the interval was chosen. The conclusion discuss the usefulness of OPC in the framework of real-time applications.

## 2 Overview

From a communication point of view a distributed system consists of service provider and service user, which are logically connected. The kind of logical connection can be described by two different models- the client/server and the publisher/subscriber model. The OPC standard distinguishes only between synchronous and asynchronous services. This section defines the use of these terms within the framework of the paper.

### 2.1 Communication in Distributed Systems – Client/Server

The data exchange takes places between a client and a server, where the client addresses (polls) the server. A client issues orders or requests to an accessible server. The communication between client and server is determined by protocols, which main tasks are as follows:

- Reception of request or response service primitives from client or server applications and the coding of these primitives in protocol data units (PDU)
- The decoding of PDUs and their report to the client (confirmation service primitive) or the server (indication service primitive) [9].

The communication between client and servers is acyclic (Fig. 1. ). The action rests with the client which sends a service request to the server. This request is answered by the server with a service response. Services are send only according to the need of the client. Typical applications are the transfer of commands between applications or irregular data transfers.

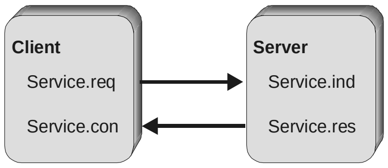

**Fig. 1.** Client-/Server-Model

## 2.2 Communication in Distributed Systems – Publisher Subscriber

The publisher-/subscriber model assumes a cyclic data supply by the publisher. The data transfer from the publisher depends either on an external request or an internal event (e.g. an expiring timer). As shown in the figure, the data send by the publisher can be received by more than one subscriber. Usually there is no explicit addressing of the subscribers.

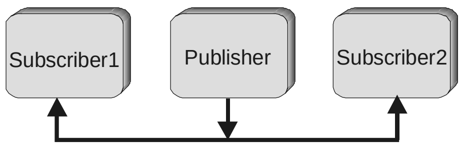

**Fig. 2.** Publisher-/Subscriber-Model

## 2.3 The Component Object Model

With the development of the Component Object Model (COM) a new platform dedicated to openness as well as speed was made available by Microsoft. COM defines mechanisms of interaction between objects. Objects in the view of the COM-model are application objects as well as data objects. Internal data of these objects are stored and accessed locally. The communication between objects is limited to so called “interfaces”, which are collections of functions. COM has to be considered as replacement of all methods of data exchange like proprietary API calls, DDE or direct calls of Windows API functions.

The basis method of communication as supported by COM is based on synchronous (blocking) calls. The calling object waits until it receives the response of the called method. The employed mechanism corresponds to the client-/server model. But in order to carry through a cyclic data exchange it would be preferable to use a mechanism comparable to the publisher-/subscriber model. It would reduce the system load which stems from the cyclic polling of data which would be necessary in the client server model, to wait for an event controlled message from the data source. COM defines “advisory connections between data object and one or more advisory sinks” [4] which can be used to implement asynchronous communication.

## 2.4 OLE for Process Control (OPC)

Based on the COM architecture OPC - Ole for Process Control – was created as a vendor-independent standard aimed at the definition of data exchange between MS-Windows based applications as well applications and vendor-specific hardware in the realm of automation. It is maintained by an independent association – the OPC Foundation. OPC defines logical objects and methods tailored to the special needs of the process industry based on COM technology. Several OPC-Clients are able to access one OPC-Server in parallel. In the area of fieldbus communication OPC is the most often installed interface between a fieldbus controller and PC based applications.

OPC uses the mechanisms provided by COM to implement synchronous services (comparable to client/server) and asynchronous services (comparable to publisher/subscriber). For an in-depth study of OPC the reader is referred to [5].

## 3 Measurement Preliminaries

### 3.1 Used Applications

**OPC test client.** The OPC test client serves as a simulation of the communication interface of a SCADA system. The different communication tasks (read/write, cyclic, acyclic) can be controlled by the client.
**OPC test server/COM test client** The OPC test server/COM testlient provides the middleware. It simulates an application which provides an OPC server interface to the SCADA system, but acquires its  data from a underlying COM server.
**COM test server.** The COM test server represents a low level communication interface to any hardware.
**Workload-Generator.** The workload generation was done by the Microsoft tool “rprobe.exe”, which is part of the Windows NT Resource Kit. The generated load is determined by the variation of the three phases wait, data access and compute of a single thread. A further control over the generated load is given by the variation of the number of parallel threads and processes maintained by the workload generator. For a detailed description see [3].
**Netload-Generator.** The netload generation used the Linux tool “ping”. It is contained within the version 5.1 of the Suse Linux distribution (or any other Linux distribution). The tool uses the ICMP service ECHO_REQUEST. The host called by this request responds with an ECHO_RESPONSE datagram. The frequency and size of the data packets determine the generated netload. The interested reader is referred to [8] for a more detailed description.

### 3.2 Environment Conditions   - Concentrated Application

**Best Case.** The PC of the concentrated application (Fig. 3. ) runs only the OPC/COM-applications. There is no other load on the PC than the load generated by the operating system itself is existent. The additional  workload generated by the workload generator is set to 0 (in fact the generator is not running).
**Worst Case.** The OPC/COM-applications are executed under an additional workload of 100%.
**Workload-Profile.** The generated workload is increased in 10% steps.

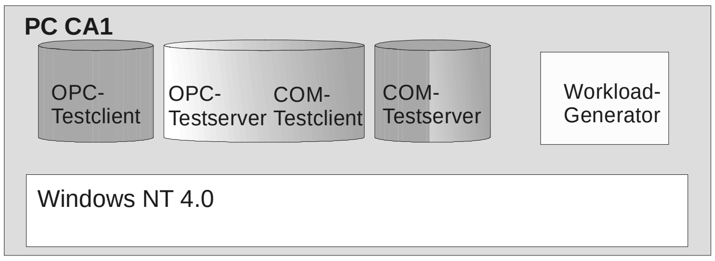

**Fig. 3.** Concentrated Application – Hardware/Software

## 3.3 Environment Conditions - Distributed Application

**Best Case.** The PCs of the distributed application (PC DA1, PC DA2) execute the OPC/COM test applications only. There is no other load on the PC than the load generated by the operating system itself. The additional workload generated by the workload tool is set to 0. The PCs (PC VA1, PC VA2, PC VA3) are connected by a coaxial cable forming a bus topology. The netload consists of the idle communication between these computers as performed by the operating system. There is no additional netload.
**Worst Case.** The execution of the distributed application takes places under an additional workload of 100%. The additional workload is generated on the computer running the OPC test client (PC DA1) as well as on the computer running the OPC server/COM client/COM server applications. The PCs (PC DA1, PC DA2, PC DA3) are connected by a coaxial cable forming a bus topology. The netload consist of the idle communication between these computers as performed by the operating system plus an additional netload of 60%.
**Workload-Profile.** The generated workload is increased in 10% steps.
**Netload-Profil.** The generated netload is increased in 10% steps.

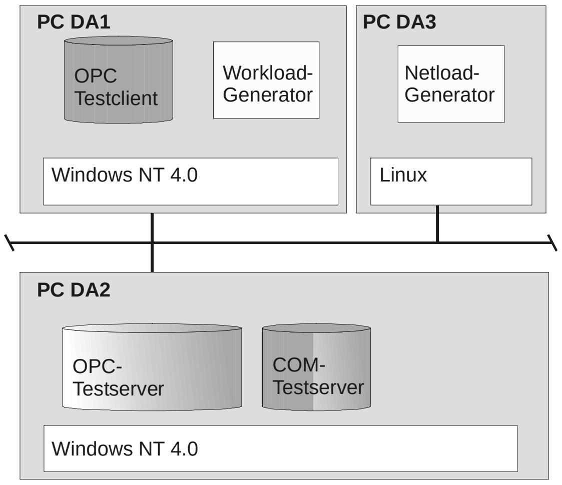

**Fig. 4.** Distributed Application – Hardware/Software

# 4 Measurement Results

## 4.1 General Remarks

The application uses a set of commands which represent a typical working scenario between a client and an OPC server. The next table contains an overview of the different communication structures and their short labels as used in this section.

| Short Label | Meaning |
|---|---|
| KD0 | Function call with return state |
| KD1 | Function call with input data (1 Byte) and return state |
| KDn | Function call without input data but output data (array of four floats) and return state |
| DT | Cyclic data transfer (array of four floats) |

The interpretation of the measurement results in this section focuses on the main parameters needed to assess the system performance from the users point of the view. In the client/server model the time needed to execute a synchronous service in the OPC/COM-Toolchain starting from the request in the test client until the reception of the function result back in the client was chosen. For the publisher-/subscriber model the interval between two message events in the client was considered most important.
**Client/Server.** The evaluation of the results of the client/server measurements is based on the system reaction time.
*System reaction time*. The system reaction time allows the assessment of the performance of the complete OPC/COM tool chain, because it exhibits all delays in the chain while executing a synchronous service. The figures contain the mean reaction time to an OPC request (line chart) as well as the standard deviation (bar chart) over a set of 1000 values. The resulting distribution function contains 80%-90% of all measured values within one standard deviation of the mean value.
**Publisher/Subscriber.** The chosen parameter message interval, the time difference between two consecutive message arrivals at the client, is considered as the most important one, because the observance of this interval matters most to the user.
*Message interval.* The given real time frame of 100ms in mind a 2-times oversampling rate was considered sufficient for this application. The OPC server sends data messages every 50 ms to the client. The resulting distribution equals a standard distribution, meaning that 95% of all values are within two standard deviations of the mean value. The figures contain the mean message interval (line chart) as well as the standard deviation (bar chart) over a set of 1000 values.

## 4.2 Concentrated Application

**Scenario 1.** This scenario investigated the system behaviour while a number of read accesses from the client took place. The processor load was increased in 10% steps. The results as shown in the diagram (Fig. 5. ) exhibit the relative robustness of the system below a workload of 70%.

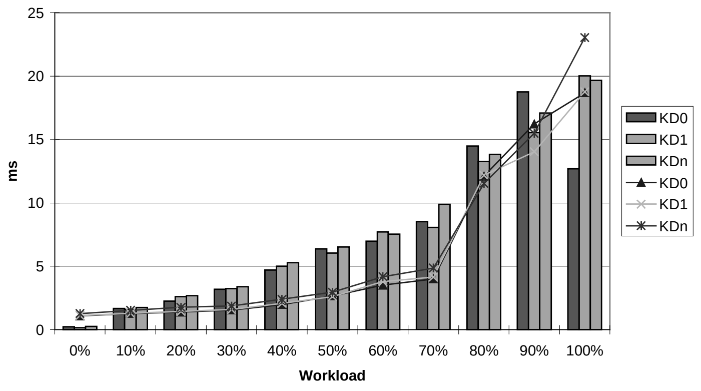

**Fig. 5.** System Reaction Time – Concentrated Application

**Scenario 2.** This scenario investigated the arrival interval of a cyclic telegram transferred from the OPC-server. The relatively high standard deviation (if compared with the previous scenario) results from an error compensation mechanism which was built into the OPC sever. If the server detects a delay in the send process it tries to compensate it by adapting the next interval to the delay. The message interval as shown in Fig. 6. Can be observed with a probability of 99% up to a processor load of 40%.

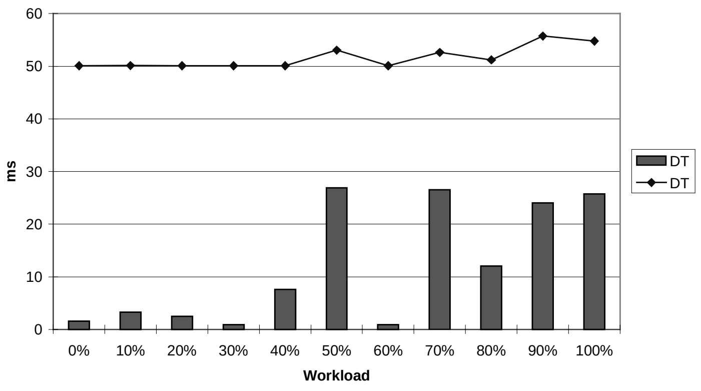

**Fig. 6.** Message Interval – Concentrated Application

## 4.3 Distributed Application

**Scenario 3.** The scenario corresponds to scenario 1 as far as data access is concerned. The application is distributed as described in section 3.3. The workload on the PC running the OPC client was increased in 10% steps. The results show the indifference of the mean value and standard deviation of the reaction times to the client PC workload. This behaviour could be expected, because the main load in executing the OPC request rests with the server. The singularity at a 60% workload points to the fact that, all efforts notwithstanding, Windows–NT should not be mistaken for a real-time operating system in a very rigid sense. A certain amount of non-determinism remains as far as the task execution is concerned.

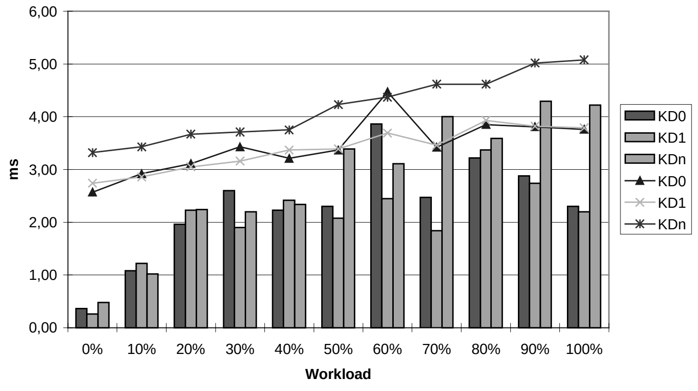

**Fig. 7.** Distributed Application – System reaction time (Client Dependence)

**Scenario 4.** Scenario 4 aimed at the test of the dependence of the system performance on workload changes on the server PC. The observed behaviour shows a direct correspondence between the workload and the mean value and standard deviation of the system reaction time. As explained above, it is a behaviour which could be expected because the main load in the execution of the OPC requests lays on the server side.

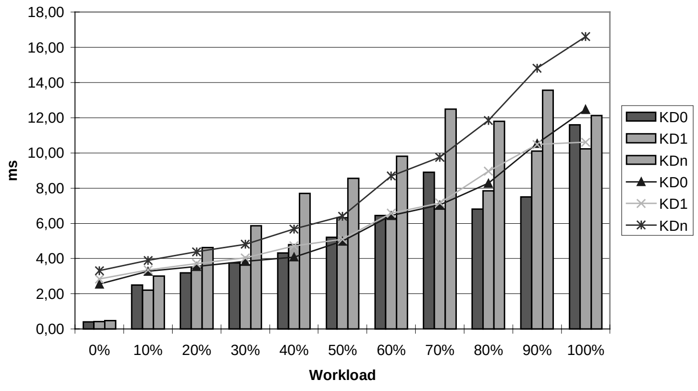

**Fig. 8.** Distributed Application – System reaction time (Server Dependence)

**Scenario 5.** Scenario 5 considered the influence of the netload to the overall performance of the system. Up to a netload of 30% the mean value and the standard deviation of the system reaction time are rather indifferent against the load on the network. A further increase in the netload leads to a rapid deterioration of the system behaviour.

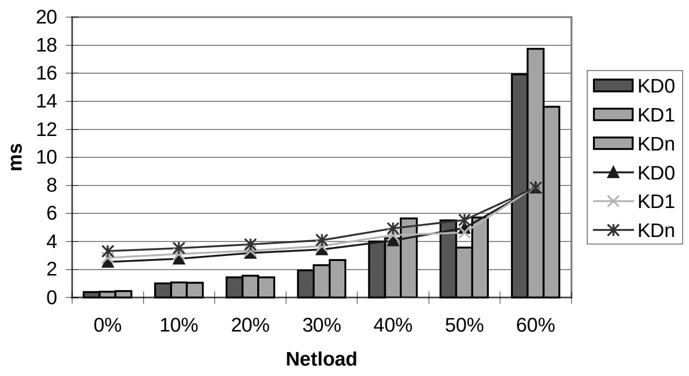

**Fig. 9.** Distributed Application – System reaction time (Network Dependence)

**Scenario 6.** The previous scenarios evaluated the change in system performance dependent on one load parameter under the assumption of the remaining of all other parameters at the best case level. Within this scenario an average load (50%) of the PCs executing the OPC/COM application is simulated. The increase of the netload allows the definition of a working set of load conditions.
The chart shows a slow increase in mean value and standard deviation of the system reaction time up to a netload of 50%. The overall slower system reaction follows from the existent processor load.

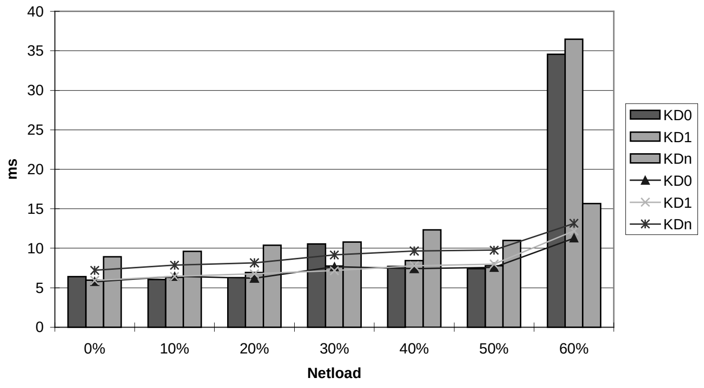

**Fig. 10.** Distributed Application – System reaction time (Network/Processor Dependence)

**Scenario 7.** The behaviour of the message interval corresponds to the behaviour of the system reaction time. As seen in scenario 4 there is a direct correspondence between the server load and the mean value and standard deviation of the parameter in consideration. Above a processor load of 80% only 95% of the measured values comply to the given maximum of 100ms.

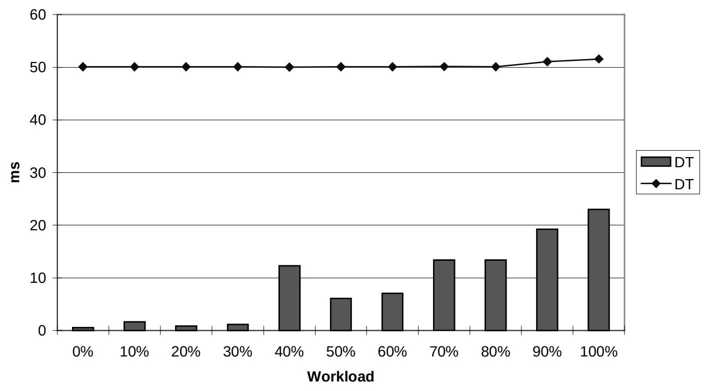

**Fig. 11.** Distributed Application – Message Interval (Network Dependence)

**Scenario 8 .** This scenario is concerned with the observance of the message interval under an average processor load of 50% on the client and server PC. The chart exhibits a relative instable behaviour of the standard deviation. This is caused by the error compensation mechanism described in scenario 2, which exerts now a major influence on the values (caused by the existing workload on the server PC). Up to a net load of 20% the given maximum of 100ms is observed with a probability of 99%.

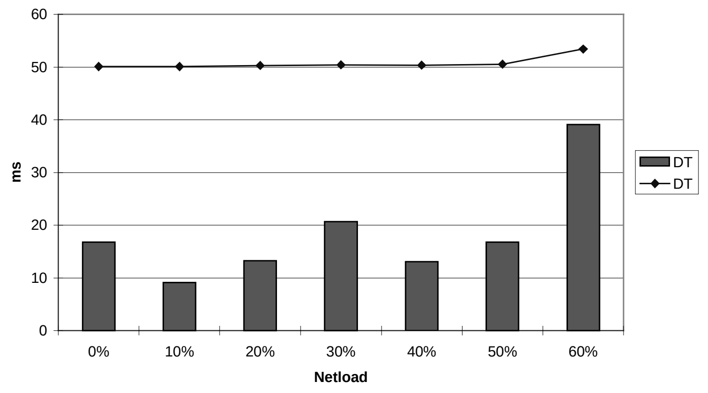

**Fig. 12.** Distributed Application – Message Interval (Network/Processor Dependence)

## 5 Conclusions

This paper researched the impact of system design and system load on the performance of distributed applications. All documented scenarios are based on tests searching not only for a “positive” result but especially for a “negative” result, which exhibited the condition under which the application started to work unreliably.

### 5.1 Concentrated Application

The scenarios of the concentrated application show that the observance of the message interval can be guaranteed (with a certain probability) even while executing all parts of the application on a single PC. In order to guarantee a 99% compliance to the message interval the external workload should not exceed a boundary of 40%.

### 5.2 Distributed Application

The scenarios which were based on distributed execution of the software investigated the behaviour of the system by changing one load parameter while assuming the others resting in the best case state. The achieved results lead to the following rules of thumb:

- The influence of the workload on a PC executing the client software on the overall performance of the system is relatively small
- The workload on a PC executing the server software determines the overall performance of the system
- The netload on the connection between the computers determines the overall performance of the system

The scenarios 6 and 8 led to the definition of a set of load conditions which would guarantee the message interval of 100 ms. It could be proven that if the upper limit was formed by a workload of 50% on the client and server computer and a netload of no more than 20% the message interval was observed with a probability of more than 99%.

Regarding the general application of OPC within the framework of SCADA or control systems we believe that OPC is well suited for transportation of data for visualisation and data storage. The expected progress in PC soft- and hardware will allow the use of OPC even in control applications in the near future.

Still, the deepest impact on the performance of any distributed system (based on OPC or not) is reached by considerate system design.

## References

1. Brockschmidt, K: Inside OLE, Microsoft Press, Redmond, 1995.

2. Burke, Th. J.: The Performance and Throughput of OPC – A Rockwell Software Perspective, White Paper, Rockwell Software Inc., 1998.

3. Microsoft-NT Workstation Resource Kit, Microsoft Press, Redmond, 1997.

4. MSDN Library January 1999 release, Microsoft Corp., 1999.

5. OLE for Process Control – Data Access Standard V1.1, OPC Foundation, 1997.

6. Rogerson, D.: Inside COM, Microsoft Press, Redmond, 1997.

7. Schnittstellen zum Prozeßleitsystem, Draft, GTM, 1998.

8. UNIX System Manager´s Manual.

9. Wedekind, H.: Verteilte Systeme, Grundlagen und zukünftige Entwicklung aus der Sicht des Sonderforschungsbereichs 182 "Multiprozessor- und Netzwerkkonfigurationen"; BI Wissenschaftsverlag, 1994.