View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Utrecht University Rep

Remote Participation Services

Report II

Werkgroep Fysische Informatica, faculty Physics and Astronomy, Utrecht University

E.A. van der Meer,
Armin Gerritsen,
B. Niderost,
A. Taal,
H Blom,
H.M.A. Andree,
W.Lourens

Utrecht, 15-9-2000

Content

Measurement Database Distribution

Introduction

Why distribution?

Expanding on the track of the DYNACORE project[1] the aim is at a centralized database to store all the measurement data. Data that is generated during various plasma physics experiments at the Textor '94 tokamak. As will be shown in the sequel, it seems not feasible to implement such a database on one single computer for a number of reasons.

One reason is the performance when accessing the database to store new measurement data. This performance is very important, as it determines the delay from the end of a tokamak shot up till the researchers have full access to their data.

Another reason is the transition from legacy software, where data is stored in a format that is specific for a certain diagnostic or research institute, to a flexible standard format, where all data is accessible to all participants via one (generic) interface. This transition cannot be made in one giant step. It is possible, however, to make the transition step by step, by constructing adaptors for existing data formats. These adaptors make measurement data that is stored in traditional formats available via the *new*, generic interface. A scientist using *new* (object oriented, C++ or Java) client programs, designed for usage with the standard database, can send a request for specific data to such an adaptor, which will retrieve the data and transform it, where necessary, to fit the new standard format. It simultaneously offers the scientist the opportunity to access the traditionally formatted data with *old* (e.g. FORTRAN, C) (analysis) programs.

Redundancy is a next reason for database distribution. If a scientist or computer program can access the measurement data via a number of paths, the data can still be accessed (for the greater part) if, for any reason, one of the computers fails. Having always two copies of the measurement data (by means of data replication, which is a feature of many modern database systems), stored on different computers, could further increase the reliability. When one computer fails, the data is still available from the second computer.

For the reasons mentioned above, distribution of the database over multiple computers is necessary. In the following sections will be described in detail what distribution means, and how it can be implemented. A possible distributed database scenario, which allows system operators to incorporate existing data sources and replicate data as necessary, will be presented.

What is distribution?

Two-tier versus three-tier

In a two-tier scenario, a database system consists of two entities. One entity is the database, which contains all the data. The second entity is the database client. This is a computer program that needs access to some of the data in the database. Important in a two-tier scenario is that the client must know how the data is stored in the database, since it has to access the data directly from it.

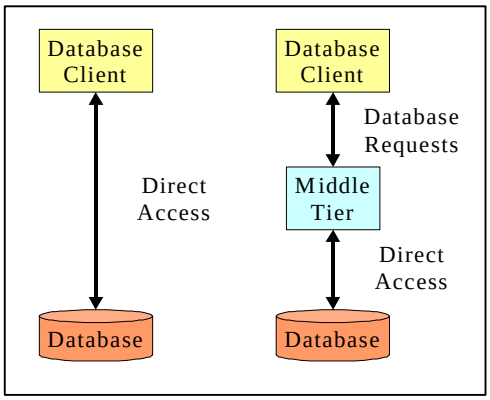
An example of a two-tier scenario is a payroll database running in Microsoft Access. Here, the payroll data is stored in a *payroll.mdb* database file, and the Microsoft Access graphical user

[1] Former EU project in Telematics Application Programme (TAP-RE4005) aimed at remote control of large scientific instruments.

interface (GUI) is the database client, which retrieves data from the *payroll.mdb* file, whenever this is necessary to complete a user request.

In a three-tier scenario, there exists a third entity between the database and the database client. This entity separates the database client from the database, in the sense that it hides all the knowledge about how to store and retrieve data in the database from the client. The client simply knows how to make database requests to this middle layer. The middle layer in its turn knows how to access the database.

Figure 1 illustrates the difference between the two-tier and the three-tier scenario.



- Figure 1 Two-tier (left) versus three-tier (right) database scenario. In the two-tier scenario, a database client accesses the database directly. In the three-tier scenario, a database client issues a request to the middle-tier, which will access the database directly as necessary.

Advantages of a middle-tier

The extra layer of abstraction in the three-tier scenario has a number of advantages.

For one thing, it allows system administrators to plug in any database they want below the middle-tier, without having to change the database client. The database client typically is a widespread application, under the control of end-users. Therefore, it is costly to update. The middle-tier, on the other hand, is situated at the server side, under the control of the system administrators. If, in a three-tier scenario, a new database requires changes to the middle-tier, these changes can be rolled out into the operational system rather quickly.

This "database-plug-in" feature can be considered as very important alternative for solving the issue of the future data acquisition at Textor '94. It makes smooth, gradual migration from legacy systems to the new system possible. A system administrator can write a middle-tier adaptor to an existing measurement database, and make the database available to all existing database clients. If the system administrator decides later on that the measurement data should be moved to another database (for example, a centrally managed standard measurement database), a simple change in the middle-tier suffices to accomplish this.

Another advantage is the possibility to implement extra functionality into the middle-tier. An example of such functionality is a set of special middle-tier routines that allow the client to retrieve pre-processed data. A typical pre-processing function is the sub-sampling of signal data to match the resolution a GUI-client that needs to present a graph to the scientist. If the pre-processing is done at the client, all the raw signal data is needed. But the amount of pre-processed data often is smaller than the corresponding raw data. This means that, when the pre-processing is done in the middle-tier instead, the data transfer from middle-tier to client - often a bottleneck, since it happens over a slow Internet connection - becomes faster. The connection between middle-tier and database should be direct and fast to prevent it from becoming an additional bottleneck. A GUI-client might retrieve multi-megabyte signals over the Internet, only to display eight hundred data points on a computer screen. In such a case, usage of special routines in a middle-tier could achieve a tremendous performance increase.

A further advantage is that a database client does not need to know the location of the database. It only needs to know where to access the middle-tier. The middle-tier knows the location of all

databases. This not only makes the “database-plug-in” feature possible, but it also allows a system administrator to move databases from one computer to another without having to notice the client of the change. This is useful for system maintenance, for example when replacing defective computers.

Finally, the middle-tier layer also makes load balancing over a number of server computers possible. A system administrator can spread the measurement database over any number of server computers. Every computer then only has to handle a part of the total load. If a computer or network connection is getting overloaded, the load can be diverted to other computers, usually dynamically!

Drawbacks of a middle-tier

The largest disadvantage of an extra middle-tier between database and client is the extra overhead. In fact it is the price to pay for utmost flexibility. Several factors contribute to the overhead. One factor is the extra method invocation that is needed with a middle-tier: instead of one invocation, when the client accesses the database, there are two invocations needed, one when the client makes a request to the middle-tier, and one when the middle-tier accesses the database.

Another factor that causes overhead is data conversion. The data in the database can be stored in a format that is not appropriate to transmit from middle-tier to client. The middle-tier might have to convert the raw data that it retrieved from the database, before it can transmit it to the client. The same holds for data that it received from the client to be stored in the database.

Finally, locating the data can cause some overhead. The middle-tier has to consult look-up tables in order to know where data resides or should be stored. Especially when the middle-tier serves a large number of different databases, this overhead can become quite large.

Except for overhead, there is another drawback to having a middle-tier. All requests from database clients have to pass through the middle-tier. When the middle-tier is a single process on one computer, and the middle-tier must do considerable preprocessing, it can become a performance bottleneck.

Load balancing in the database layer does not solve this problem. The only solution is to add another level of load distribution: multiple active middle-tier objects. The middle-tier objects can be implemented as separate threads in a single server process, multiple processes on one computer, multiple processes running on different computers, or a combination of these options. A client first locates a suitable middle-tier object, and then makes its request on that object.

The remaining question is how a client finds a suitable middle-tier object. There are two possibilities: the client can have a reference to a default middle-tier object hard-coded into its program, or he can contact a special management object and ask it for a suitable reference.

Once a client has found a middle-tier object, it is bound to it. This is a problem when a middle-tier object is unable to fulfil its requests, for example because of an increase in the load on the object, or because the object does not have access to the necessary database. In this case, it is possible to implement a hand-over mechanism. Such a mechanism provides a way for one middle-tier object to tell the client to connect to another middle-tier object. The first middle-tier object is responsible for initialising the second one, before it provides the client with the reference of the second object.

A Possible Scenario

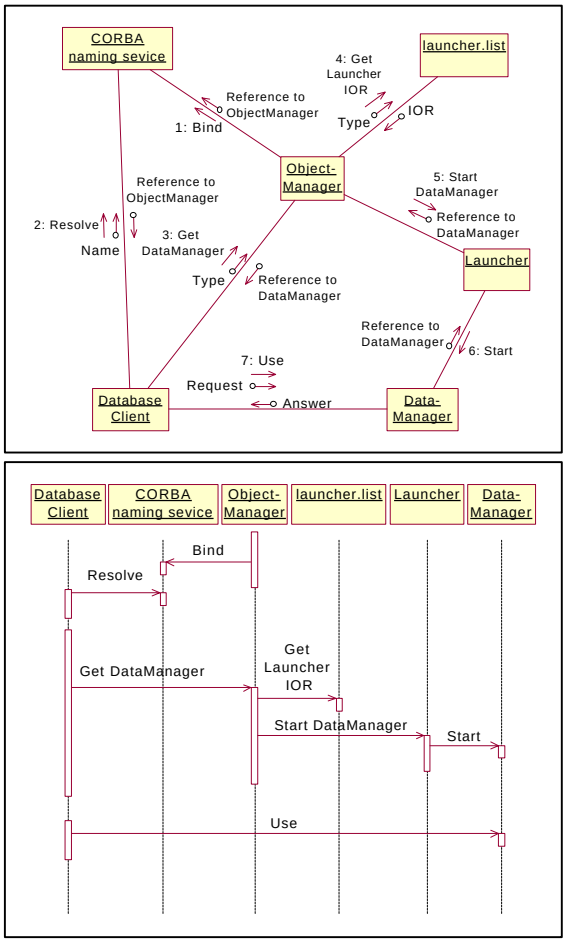
This chapter describes the three-tier scenario that has been built for the Dynacore demonstrator (named DynaDemo), which is considered as a promising basis of a future Textor'94 data storage and retrieval system. The demonstrator can be accessed from the Dynacore website at http://hst3731.phys.uu.nl/dynademo. Documentation about Dynacore and the demonstrator is available from the same website, here one will find also all recent sources of the software[2]. *The DynaDemo has been demonstrated at the SOFT21 conference in Madrid*[3].

In the next sections indications will be given how the existing middle-tier design can be augmented in order to serve a highly demanding environment.

Distribution of the middle-tier

DataManager objects

The DynaDemo middle-tier is based on DataManager objects. There exist different implementations of DataManager objects. Every implementation provides access to a different kind of database. At the moment, DynaDemo uses three different implementations. One implementation supports an object database, as described in the next section. Another implementation provides access to data in existing DOM4 files. DOM4 is the file format that is used traditionally by the FOM Institute in Rijnhuizen. The third implementation links to RT2 data. This is the data that Textor'94 uses to store general measurement data, which must be available to all scientists.



- Figure 2 Collaboration (above) and sequence diagram (below) showing how a database client can contact a DataManager.

CORBA naming service

A database client must have a reference to a DataManager in order to access data in the measurement database. To get such a reference, the client must traverse the path in the collaboration and sequence diagram shown in Figure 2. It must first contact a CORBA naming service. The reference to the naming service, which is a CORBA server object itself, is stored in a file that the database client can read.

The naming service maintains a list of references to active CORBA server objects, mapped onto unique, meaningful names. This list is updated constantly, as new CORBA server objects register themselves at the naming service when they are started. A client asks for the reference to

[2] Please contact Beat Nideroest to obtain download permissions

[3] A software Architecture for remote participation at the TEXTOR '94 Experiment, B.U. Niderost et .al. Soft21, Madrid, Sept. 2000

a special server object, named ObjectManager. Only one such object exists in the whole middle-tier.

The ObjectManager and Launcher objects

The ObjectManager (,see also the appendix for details,) is responsible for managing all active DataManager objects in the middle-tier. A database client can ask the ObjectManager for a DataManager. It includes the name of the desired DataManager implementation in the request. The ObjectManager answers to the request by returning a reference to an available DataManager object of the specified implementation. In case no idle DataManager object is available, a new one can be created.

The ObjectManager delegates the creation and management of DataManager objects to special Launcher objects. These CORBA server objects are specialized in starting and managing one specific kind of DataManager implementation. They give a reference to a DataManager object back to the ObjectManager, which in turn sends the reference to the database client that placed the request in originally.

The delegation mechanism makes two important features possible. It allows the ObjectManager, running on one particular computer, to start and manage DataManager server objects on different computers, provided there is a Launcher running on these (remote) computer. If the ObjectManager would start and manage DataManager objects directly, these could only be running on the computer on which the ObjectManager is running itself, since there are no generic, cross-platform mechanisms to start and manage processes on remote computers.

The second important feature of the delegation mechanism is that knowledge of different DataManager implementations is taken from the ObjectManager implementation. The ObjectManager can delegate the start-up and management of any new DataManager (type) implementation just as it delegates these for already existing implementations. It only needs to have a reference to a Launcher object that accepts the delegation to start and manage an object with the new implementation. How the ObjectManager gets these references is treated in the next section.

Launcher references

In DynaDemo, the ObjectManager retrieves a list of references to available Launcher objects (,see also the appendix for details,) from a file named *launcher.list*. Every launcher is mapped onto a unique name that is also stored in the file. This name is the same name that a database client uses to indicate which DataManager implementation it desires. So a database client actually chooses a desired Launcher, which in turn provides a reference to an object of the desired implementation.

There is currently no mechanism for a Launcher to automatically register itself at the ObjectManager. This means that a system administrator has to add or change an entry in the *launcher.list* file manually whenever a Launcher is (re) started, moved to another computer, etc. The Launcher can not do this for the administrator, as it most likely neither does know the location of the *launcher.list*, nor have access to it.

A solution to this problem would be to have Launcher objects register themselves at a naming service in a special naming context [1]. The ObjectManager could then scan this naming context instead of the *launcher.list* file when it is looking for a reference to a specific Launcher. The naming context could itself contain other naming contexts. If a client refers to such a child

context, the ObjectManager can choose any Launcher that is registered in the child context, thereby distributing the system load over multiple computers. A client can of course also name a single Launcher from the child context, in which case it chooses the desired Launcher itself.

Load balancing

If necessary, the ObjectManager would be the correct place to put load-balancing functionality for the middle-tier. In DynaDemo, however, such functionality is not present, as there is no need for it yet. The performance bottlenecks are located elsewhere.

For the same reason a possible other friction point has not yet been removed: the DOM4 and the object database Launcher objects still start a new DataManager object every time a database client requests one. To increase performance, these Launchers could be changed so they maintain a pool of references to idle DataManager objects. These could be returned to the ObjectManager as soon as requests come in. A pool of idle DataManager objects could decrease the client response time considerably, as the client does not have to wait for the time-consuming start-up of a new DataManager object.

If load balancing becomes necessary in the future, it could be implemented as follows. In addition to the name of the desired DataManager implementation, the ObjectManager could use the client's properties like its username or physical location to select the name of the Launcher to which to delegate the request. It could use child naming contexts as described above to select all Launchers that can provide the desired DataManager implementation on suitably located computers. Then it would ask each of these Launchers what its load is, and select the least occupied.

Reusing DataManager objects

In the object-database and DOM4 implementation of the DataManager, there is a one-to-one relationship between a DataManager object and a database client. The client opens a transaction on the DataManager, after which it can perform any database request it needs. When the client (finally) closes the transaction, the DataManager is left waiting, in case the client might want to open another transaction. After having been idle for a predefined time, the DataManager shuts itself completely, thereby freeing any resources it might have allocated. If the client wants to access the database again, it must go back to the ObjectManager and apply for a new DataManager, as the old reference is no longer valid.

Starting DataManager objects and shutting them down can be very time-consuming. Recycling them could therefore enhance the performance of the whole system. If DataManager pools are implemented in the future, the Launcher that started a DataManager could implement this recycling by returning a DataManager object to its DataManager pool after the DataManager has been idle for a predetermined time. This would permit an additional increase in performance.

Multiple clients per DataManager

In contrast to the object-database and the DOM4 implementation, the RT2 implementation does not cover transactions. Since database clients are not aware of this, they will still try to open and close transactions. They don't notice that nothing actually happens at the server side.

The absence of transactions (- a guarded, critical section in the code, not be re-entered -), in the DataManager object implies that now one object can simultaneously serve multiple clients. To exploit this feature, a special Launcher object has been constructed. It does not start a new

DataManager every time it receives a request from the ObjectManager, but, instead, always returns the same reference to a single DataManager object, which was originally started by the system administrator. This improves performance, as it circumvents the time- and memory-consuming overhead of starting a new process for every single database client. However, performance might improve even more having the Launcher also spawn new DataManager objects into separate processes, when the existing process is extremely busy. This feature could easily be added in the future if necessary.

DataManager implementation listing

In the scenario described above, a client has to select a DataManager implementation before he can access the database. However, a specific implementation might only provide access to a small part of the whole data collection. Which part that is depends on the implementation. A DOM4 DataManager, for example, does not provide access to RT2 data. A client must therefore know the location of the data, he requires. He must explicitly know a certain DataManager implementation by name. To access data subsequently that is in different parts of the database, he must even know the names of several implementations.

This situation is not preferred. Whenever a new DataManager implementation is added or data is moved from one part of the database to another, all clients must be informed, otherwise, they will not be able to locate all the data correctly.

In DynaDemo, the ObjectManager has a special list function, which will return the name of all available DataManager implementations. This partly solves the problem mentioned above. Every DataManager implementation already organizes all its data in a tree-formed database graph. A database client can treat the list that is returned by the ObjectManager as an additional layer on top of this graph. This way he can organize his accesses to the whole database.

There are still problems with this solution. In the first place, the solution is not transparent to the client. The client must be aware of the special list function, and actively use it to compose its own, virtual database graph. Secondly, the solution does not allow a system administrator to organize the database graph logically. The administrator can only choose the name of the DataManager implementation, as it will appear in the top level of the database graph. She cannot “mount” a DataManager implementation in a lower level, under a specific node.

DataManager handover mechanism

A better solution would be to implement a handover mechanism. DynaDemo does not use the handover mechanism yet. However, the necessary provisions are already present in the DataManager interface definitions.

A client that uses the handover mechanism would always request the ObjectManager to provide him with a reference to a special Root DataManager. This Root DataManager is indicated by a unique name, the only one a client has to be aware of. The Root DataManager knows where to find the other DataManager implementations.

A system administrator configures the Root DataManager, so it knows the location of all other DataManager implementations. Furthermore, the system administrator can impose any logical database graph on top of the physical structure she finds suitable. It is even possible to mount a part of the database under multiple nodes in the database graph. This way, a system administrator

can, as an example, make the same data available organized both by shot number and by research institute.

A client might ask the Root DataManager for information about one of the nodes in the database graph that the Root DataManager cannot provide itself. In that case, the Root DataManager will raise a CannotProceed exception. This exception contains a reference to another DataManager implementation. The client can contact that DataManager to investigate further. The combination of exception raising and contacting another DataManager is the actual handover.

It is imaginable that the new DataManager implementation still cannot answer the client. Or it might be too busy to handle the request. In such a case, it can raise another CannotProceed exception, and so on. This makes the handover mechanism a very powerful tool. A system administrator can use it to construct database graphs of arbitrary complexity, and it can even help balancing the system load.

Database wide object names

Data in the measurement database may contain references to other data. Since the data is organized in objects, ordered in the database graph, references only need to specify the absolute path to the object in the graph. This works fine as long as the graph is predefined and fixed, so that all DataManager implementations are aware of it, and can organize their object names accordingly.

However, in DynaDemo, the graph is not predefined, but created dynamically by the clients. A DataManager implementation has therefore no way to resolve the absolute path of its data objects. It will return a relative path instead.

A DataManager can construct references to data that is made available by other DataManager implementations only if the handover mechanism is implemented. In that case, the system administration should construct the database graph in such a way that all DataManager implementations have a reference back to the Root DataManager, mounted as a node in their own sub-graph. A client might then ask a DataManager implementation to resolve a relative path that starts with the path to such a node. The DataManager will answer by raising a CannotProceed exception, containing the reference to the Root DataManager, and the relative path with the path to the node stripped off. The client finally will continue to resolve the path at the Root DataManager, which has access to all data in the database. If a data object wants to reference another data object, elsewhere in the measurement database, it only needs to append the absolute path to that object to the relative path to the Root reference node in the current DataManager implementation, and a database client will be able to resolve the compound path.

Implementing ObjectManager and DataManager objects as naming contexts

There is a large overlap in the listing functionality between the ObjectManager, the DataManager objects and CORBA naming context objects. Once the handover mechanism is implemented, this overlap will even be larger. While the interface definitions for the listing functionality differ considerable between the mentioned naming mechanisms, the handover-mechanism related part of the DataManager definition has been designed to reflect the overlap. In the future, it might be useful to adapt the ObjectManager and DataManager interfaces, to make them implement the naming context interface. This would standardize their interfaces, making them easier to understand (as CORBA developers might already be familiar with naming contexts), and available to existing CORBA tools, for example a naming service browser.

Distribution on database level

This paragraph describes the distribution mechanism of the object database that is used by one of the most elaborate DynaDemo DataManager implementations. The object database could supposedly be the alternative amongst the TEC collaboration to store future measurement data. It is implemented using the Objectivity/DB database product [2].

Problem and solution

As data volumes are increasing, high performance is very important. The user requirement specification in the plasma physics community at TEC-IPP states that the system must be able to store 500 MB of measurement data within one minute. Previous performance measurements have shown that a single DataManager server running on a SUN Ultra-10 computer cannot meet this requirement. Distribution at the middle-tier, using more than one DataManager object, running on different computers, distributes the system load. However, in principle there still is only one database, on a single computer, where all the measurement data is stored. All DataManager objects have to access this database, which is becoming a new bottleneck.

All objects in an Objectivity/DB single database are physically stored in the same file, on computer. However there exists a higher level of aggregation, the so-called *federation*. It provides a unique naming and access scheme for all objects stored in any database that is part of the federation. The database files themselves can be distributed over many different computers, but since Objectivity/DB provides the mechanism to access objects transparently, a database application is not aware of this.

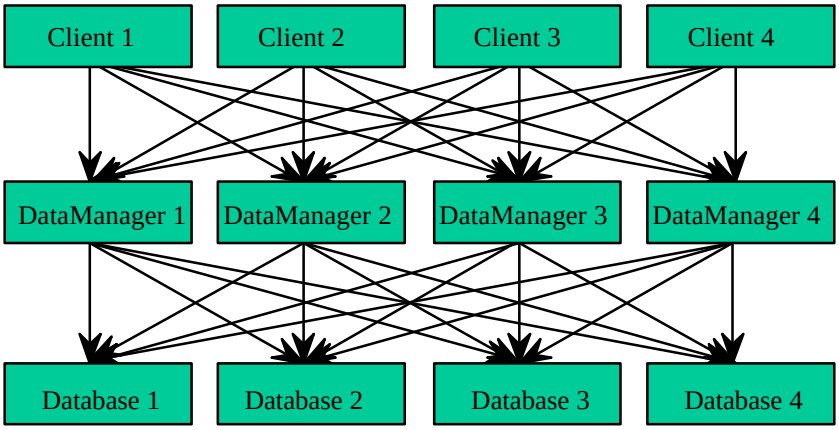
Clearly, distributing the measurement data over multiple computers will reduce the load on any single computer. The data should be distributed such that all computers receive approximately the same load. If the load on one computer becomes too heavy, for example because its database volume is growing very rapidly, it is possible to move part of the data to another database on another computer. To accommodate this Objectivity/DB provides an optional mechanism to replicate the database, named Objectivity/DRO. In that case, there are two databases, located on different computers that contain exactly the same data. The databases are kept synchronized all the time. Objectivity/DRO can distribute the load of requests to read data already in the access phase. The load of requests to store data, however, will not be distributed, as both databases must incorporate any change to the data they contain. Objectivity/DRO is licensed separately from Objectivity/DB.

Distribution of database-tier and middle-tier

Distributing the measurement database using the federation mechanism does not make the middle-tier redundant. The advantages of having a middle-tier, as given in the previous chapter, still stand. Specifically for DynaDemo, removing the middle-tier would make all clients aware of the database implementation. In case of Objectivity/DB clients, they would need a valid software license to use it. This is very costly. They would also lack a security mechanism, which cannot be tolerated for a system that will be used over the Internet, and moving data transparently from one database to another is no longer possible, as this would change the references to the moved objects.

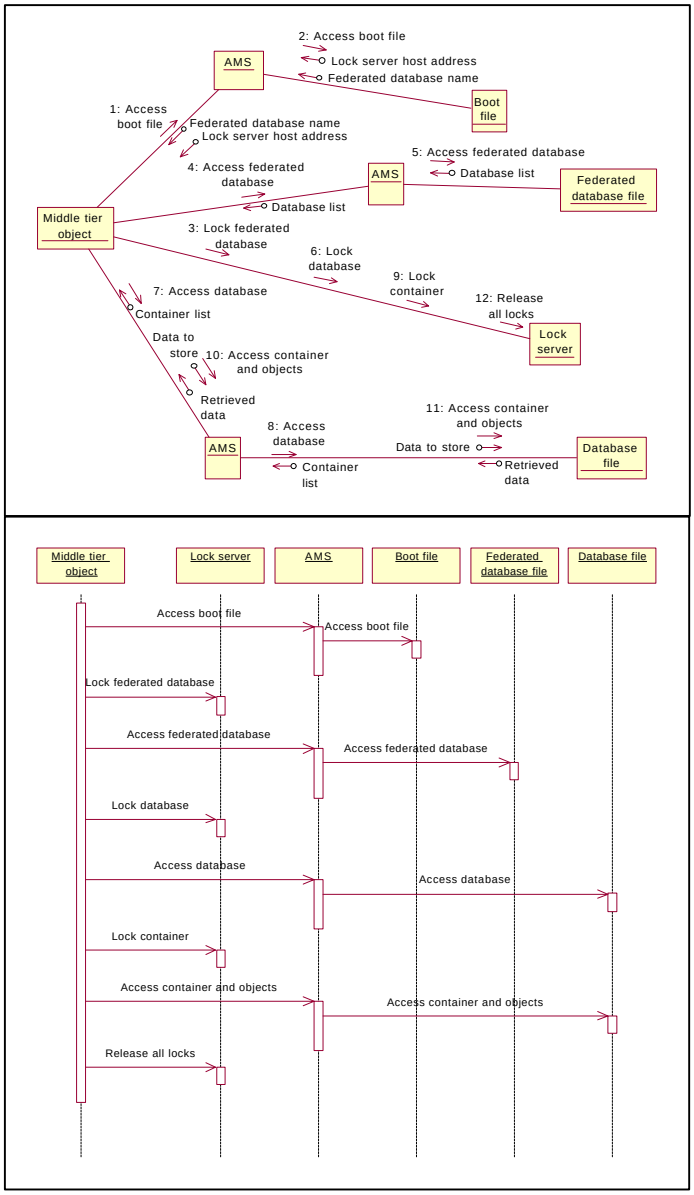
The middle tier should still be distributed. Otherwise, the high-performance bottom layer will only be accessible via a slow middle-tier that becomes the bottleneck of the whole system. The

ideal picture is shown in Figure 3. Clients connect to one of a whole pool of DataManager objects. The DataManager objects in turn access one of the database files.



- Figure 3 Distribution of both the middle-tier and the database-tier of a measurement database. Any client can use any DataManager to contact any database. So whatever DataManager it is assigned, it can always access all data.

Technical details



- Figure 4 Collaboration (above) and sequence diagram (below) showing how a middle tier object can perform operations on an Objectivity/DB database system. Observe that the middle-tier object access the boot file, federated database file and the database files using AMS servers. Therefore, there must be an AMS server running on every computer that contains one (or more) of these files. If there are multiple files on a computer, they can be accessed using a single AMS server. Files on the computer where the middle-tier object is running, can be accessed directly, without using an AMS server.

There are four Objectivity/DB components that work together to make distributed database federations possible. Figure 4 shows how a middle-tier object interacts with these components in order to perform database operations. The following paragraphs describe the components in more detail.

Federated database

The first component is the federated database file. It contains information on which databases together form the federation, and where these databases are located.

Lock server

The second component is the lock server. It is a single process running on one computer. Before a middle-tier object performs any operation on a database or object in a database, it must retrieve a lock for that operation from the lock server. A lock is only granted when the requested operation is safe. For example, when one middle-tier object has obtained a lock for reading a database, another middle-tier object that applies for a read lock on the same database will also obtain one. However, a middle-tier object that applies for a write lock on the same database will have to wait until the all read locks have been released. Otherwise, it might change data while other middle-tier objects are reading it, leaving them with inconsistent data. As there is only one lock server, it is a single point of failure of the whole system, and a potential performance bottleneck. The latter is most likely not a problem, since the lock requests are not very computation- or I/O-intensive. Access times, however, are of importance. If a middle-tier object must obtain many locks in sequence, the access times can add up to form a considerable delay. To make access times as small as possible, the middle-tier should be physically located close to the lock server, preferentially on the same computer platform. Also,

if a middle-tier object knows in advance that it will access a number of database objects in sequence, it can consider locking them simultaneously.

The lock server problems could be solved by another Objectivity product, named Objectivity/FTO. It allows system administrators to create so-called autonomous partitions, which replicate all the functionality of a federated database, including lock servers. If, for any reason, a middle-tier object cannot use one autonomous partition, it will automatically try another one. Like Objectivity/DRO, Objectivity/FTO is licensed separately from Objectivity/DB.

Boot file

The third component is the boot-file. It contains general information, for example on the physical location of the federated database file and the hostname of the lock server.

AMS server

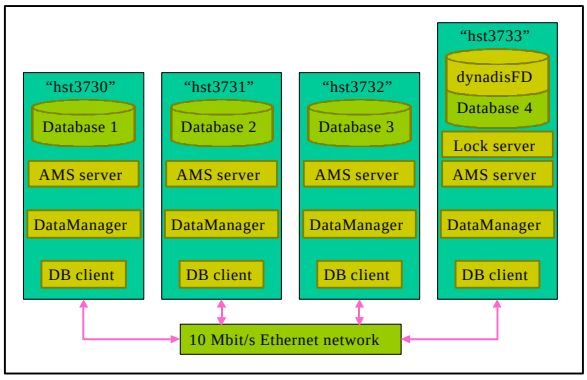
The last component is the Advanced Multithreaded Server (AMS). This server makes Objectivity/DB files, located on one computer, accessible to middle-tier objects on other computers. Middle-tier objects need access to these files, because the Objectivity/DB program code, which uses these files, is hard-linked into the middle-tier objects own program code. The files can also be made accessible via Network File System (NFS) mounts, but the AMS is preferred for several reasons [3].

Performance measurements

There are many tests possible that measure the performance of the demonstrator described above. How useful they are depends mainly on for what purpose they are done. The measurements described in this chapter are meant to show if the distribution mechanisms work. Results of other measurements, which show the performance of a single DataManager object using a single database, have been published before [4]. At the end of this chapter, some other measurements are described, which have not been performed yet, but might be useful.

Distribution over multiple SUN-Ultra-10 computers

The following measurements have been performed on four computers that are part of the GigaCluster [5]. The GigaCluster consists of eight SUN-Ultra-10 computers, running the Solaris 7 operating system, Objectivity/DB version and the SUN workshop compiler version The same computers have been used under the similar conditions for previous measurements, mentioned above. The computers are interconnected using a 10 Mbit/s Ethernet network.



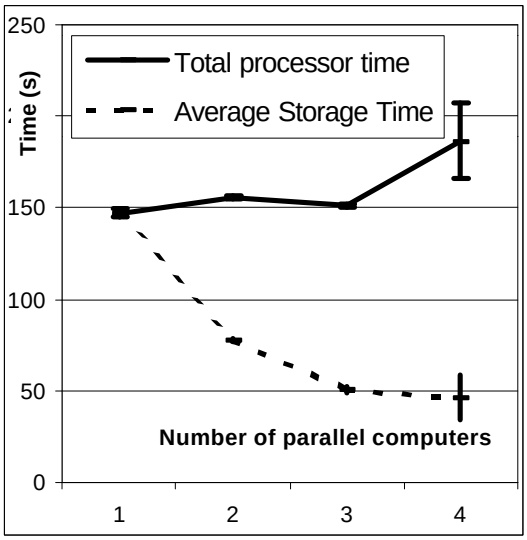
• Figure 5 Set-up for the distributed database performance measurements on the SUN-Ultra-10 GigaCluster.

Figure 5 shows the measurement set-up. The measurement database is distributed over four computers. Every computer has one database file, one AMS server, a DataManager, and a database (DB) client that stores data in the database. The database client uses the DataManager that is running on the same computer, and stores data in the local database file. This is the most optimal situation, as it does use neither the AMS servers nor the network to store data. However, the DataManager objects still need the network to connect to the lock server, and to resolve the references to the databases. Initially the DataManager objects only know the location of the federated database.

Using this set-up, the time a database client needs to store 500 MB of raw signal data has been measured. The measurement has been repeated four times, first with only one database client, running on computer "hst3733", then with two clients, three clients, and finally with four clients. The results are shown in Figure 6.

As can be seen from the figure, the parallelisation of the data storage works very well. The graph showing the total processor time indicates that there is only a few seconds overhead associated with the database distribution over two or three computers. Distributing the database over four computers yields a larger overhead and a much larger uncertainty in the total processor time. The exact reason for this effect is unclear, but it seems plausible that the (shared) network reaches a limit. For example, it might become overloaded and drop packets. This would result in TCP/IP time-outs, which in turn cause a lot of overhead and uncertainty in the total processor time.

In the present set-up the average computer time shows that three computers in parallel are able to meet the performance requirement of storing 500 MB of measurement data within 1 minute. Four computers in parallel should further reduce the time, but the uncertainty in the storage time will become larger. The figure indicates that the requirement of storing 500 MB within 1 minute on 4 computers is not met always. In this test situation distribution of the data storage over three computers seems to be optimal, but would not be exemplary for other configurations.



- Figure 6 The average time it takes to store 500 MB of raw data, using 1, 2, 3 respectively 4 computers in parallel. The total processing time of all participating computers together is also shown. Every point has been measured multiple times. The error bars show the standard deviation in the results of the repeated measurements. Only the one for 4 computer-case stands out, the others are too small to be visible in the graph.

Performance of a 700 MHz Athlon computer running Windows NT 4

The computer industry increases the performance of their architectures at an incredible pace. The tests of the previous paragraph were performed on computers that are over a year old. To have a view of what a single commodity computer can achieve nowadays, another test has been done. It was carried out on a computer with an Asus K7M motherboard [6] running the Microsoft Windows NT Server 4.0 [7]. The CPU was a 700 MHz Athlon-processor [8]. The measurement database was stored on an 18.2 GB Quantum Atlas 10K SCSI hard disk [9]. The test repeated the one in the previous paragraph, but now only for a single DataManager using a single computer. The time necessary to store 500 MB of raw signal was 99 ± 6 seconds in this case.

This measurement indicates that today, two commodity computers working in parallel can achieve the performance goal of storing 500 MB of measurement data within one minute. If the machines would work completely in parallel, it would take them approximately 100 / 2 = 50 seconds to store 500 MB of measurement data. The locking mechanism will increase this time slightly, but, looking at the distribution overhead on the SUN cluster, this overhead would not amount to more than 10 seconds. It is also to be expect ed that in the near future, one single commodity computer will be able to achieve the performance goal all by itself.

Future measurements

Many more tests can be thought of. It is questionable however how much useful information they would yield. Here a set of measurements is listed together with the reasons why the individual measurements might be useful:

1) Repeat the measurements on the SUN-Ultra-10 cluster with the newest versions of the Solaris operating system, the SUN C++ compiler and Objectivity/DB. This would enable the 64-bits capabilities of the SUN-Ultra-10 architecture. It removes the 2 GB database file size limit [10] that is currently experienced, and might improve the performance. The number of computers can also be increased from four to eight in total, just to see how this scales.

2) Repeat the measurements on other computer architectures or operating systems. This might yield information on how the performance test results of commodity computer architectures compare to each other. However, since the performance of commodity computers constantly increases, these tests are only useful to make a short term buying decision. Comparing a state-of-the art Athlon system to a one-year-old SUN-Ultra-10 system yields only a distorted view, as there are much faster SUN systems available today. Useful other platforms to test would be the Compaq platform, or an Intel system running the Linux operating system.

3) Measure the performance of a mixed-platform distribution. It is easy to imagine that the computer system on which the measurement database is installed will need to be expanded in the future. At that time, the computer platform for the expansion computers should be reconsidered. Simple performance tests, for example using the Athlon/Windows 2000 platform together with the SUN-Ultra-10 platform, can show now if there will be negative issues to mixing them in the future.

4) Analysing the bottlenecks of the architecture is also very useful. As an example, it can yield numbers on how many DataManager objects should simultaneously access a single database. This is useful information when deciding to distribute the load of one busy computer over multiple computers. Should the database be split, or is it sufficient to move the DataManager objects to the new computers? It is also interesting to analyse when the network connection becomes a bottleneck, and if Quality of Service protocols can help resolve network-related problems.

5) A last test could focus on the usage of CORBA components [11] within the demonstrator. CORBA components are middle-tier objects that are controlled by automatic ORB functionality instead of by ObjectManager objects. They form an industry-standard that provides its own middle-tier distribution functionality. Many component-enabled ORBs include state of the art load-balancing functionality, which cannot be implemented using the limited resources available to the Demonstrator development community. It is therefore interesting to compare the performance of a demonstrator using these systems to the performance of the current demonstrator implementation.

References

[1] OMG group, “CORBAservices specification”, chapter 3. Available on the Internet via http://www.omg.org/cgi-bin/doc?formal/97-12-10

[2] Objectivity Inc., http://www.objectivity.com

[3] Objectivity Inc., “Objectivity/DB Administration”, chapter 8. Available to Objectivity customers with Info Center access on the Internet via http://info.objy.com/$webfile.send.MANUALS./ADMIN52.PDF

[4] B. U. Nideröst, L. Gommans, G. Kemmerling, M. Korten, C. T. A. M. de Laat, W. Lourens and E. A. van der Meer, "Objectivity / Corba Distributed Database Performance on a Gigabit Sun-ultra-10 Cluster," Real-Time 1999 conference issue of the IEEE Transactions on Nuclear Science. Available from: http://www.phys.uu.nl/~niderost/papers

[5] See Internet, http://www.phys.uu.nl/~wwwfi/gigacluster

[6] Asus K7M motherboard product description: http://www.asus.com/Products/Motherboard/slota/k7m/index.html

[7] Microsoft Windows NT server 4.0 product description available on the Internet: http://www.microsoft.com/ntserver/default.asp

[8] AMD Athlon processor product description available on the Internet: http://krypton.amd.com/products/cpg/athlon/

[9] Quantum Atlas 10K product description available on the Internet: http://www.quantum.com/products/hdd/atlas_10k/atlas_10k_overview.htm

[10] Objectivity Inc., “Objectivity/DB Administration”, chapter 3. Available to Objectivity customers with Info Center access on the Internet via http://info.objy.com/$webfile.send.MANUALS./ADMIN52.PDF

[11] ORBOS, “CORBA Component Imperatives”, May 25th, 1997, available on the Internet: http://www.omg.org/news/610pos.html

Network performance measurements IPP - FOM - UU

Introduction

In this chapter the results of the network performance monitor between IPP - FOM and Utrecht will be discussed more extensively than in the previous report[4]. The attention will be focussed on the results in the first thirteen weeks of the year 2000. However, also the improvements during the complete observation period (22-8-1999 until 2-4-2000) are discussed. Because bandwidth, and not so much availability, is the limiting factor in these connections, mainly throughput results are presented here.

Sites

The results of the throughput measurements between the following sites will be compared:

Connection	BW [Mbit/s]	Weeks
ZAM<=>UU-36	100	34 (99) - 14 (00) 34
IPP<=>FOM	10	(99) - 05 (00)
TEN-DE<=>TEN-NL	100	37 (99) - 49 (99)

The participating sites where placed at the following locations:

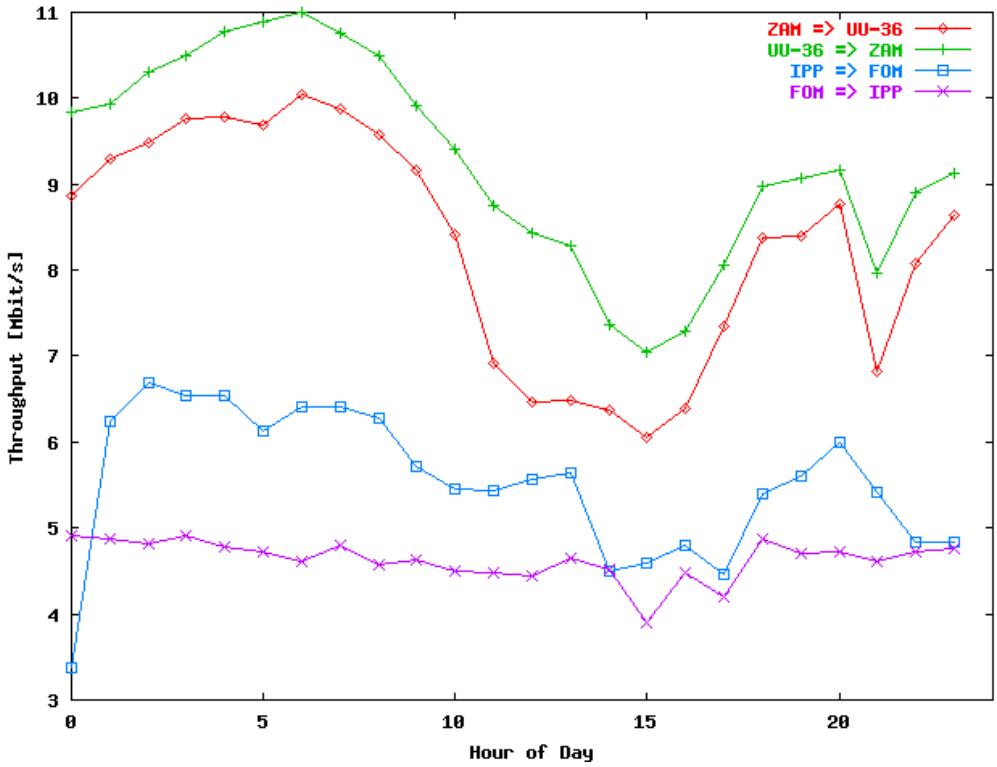
Site	Location
ZAM	ZAM Department, Jülich, Germany.
IPP	IPP Department, Jülich, Germany.
TEN-NL	Dutch PoP TEN-155 network, Amsterdam, Netherlands.
TEN-DE	German PoP TEN-155 network, Frankfurt, Germany.
FOM	FOM Institute Rijnhuizen, Nieuwegein, Netherlands.
UU-36	Computational Physics Uni. Utrecht, Utrecht, Netherlands.

For the connections between these sites the results concerning performance and availability are presented in the following.

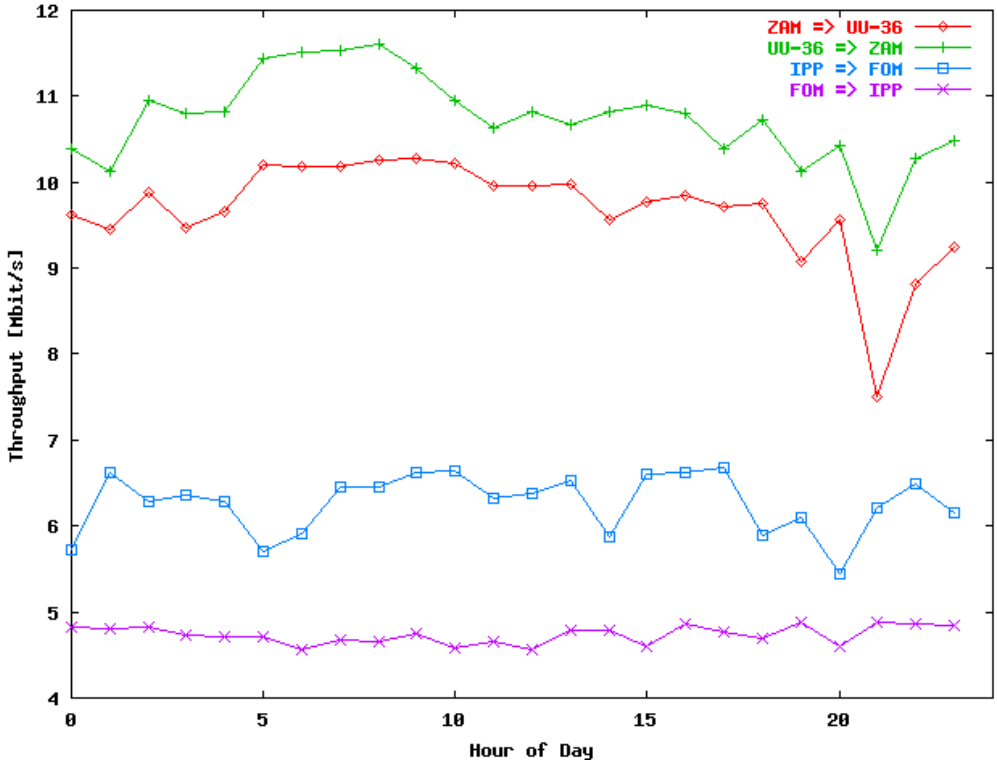
Time throughput averages

In this section the throughput average values, calculated at the hours of the days for the bi-directional connections ZAM <=> UU-36 and IPP <=> FOM will be compared. There are mean values calculated for working days (Mon - Fri) and for in the weekend (Sat - Sun). The results are obtained for the first thirteen weeks of 2000. This implies that the mean value of a workday (weekend day) is the result of averaging 65 (26) throughput values. Figure 7 presents the hourly throughput values during working days and Figure 8 shows the corresponding values in the weekend.

[4] Remote Participation, Report I, Jan 2000



• Figure 7 Mean workday throughput in the network between IPP and FOM



• Figure 8 Mean weekend throughput in the network between IPP and FOM

From both figures the following conclusions can be drawn:

- The striking behaviour is the clear performance decrease at working days between 08h - 18h. This is especially true for the connections between the sites with 100 Mbit/s interfaces (ZAM <=> UU-36), but also the connections between the sites with 10 Mbit/s interfaces show performance diminution. During the weekend the performance difference between day and night is not so very significant.

- We compute the ratio between the minimum throughput during daytime and the maximum throughput at nightfor working days. The table below contains this ratio for the various connections (in the values the non-typical performance decreases for IPP => FOM at 00h and for ZAM <=> UU-36 around 20h have been ignored)

Connection	Min-Tput / Max-Tput
ZAM => UU-36	0.60
UU-36 => ZAM	0.64
IPP => FOM	0.67
FOM => IPP	0.79

- With the exception of FOM => IPP all ratios are about the same value. The explanation for this may be that with congestion at a router, the queuing protocols, sliding window adjustment, etc. are responsible that a proportional part of the received packages will send to the next hop. This implies that the bandwidth to the next hop will be related to the incoming bandwidth. This mechanism breaks down when packets are lost due to heavily congestion at the router. Therefore, these results were less clear found in earlier throughput measurements where the performance of the network was worse.

- The performance decrease at 00h for the IPP => FOM connection is typical for this connection. The result is unknown, but probably local to the IPP. May be backup activity or other regular service jobs, generating local traffic may be the cause. The load of the IPP host at that moment is not larger than otherwise, so it is not a performance feature. The reason that we do not find it in the reverse situation, FOM => IPP, may be due to the overall lower bandwidth of that connection.

- The performance decrease around 20h for the connection ZAM <=> UU-36 is not clear. However, other results show that the cause is probably situated in the Utrecht University network. The performance diminution is found for all days of the week.

Throughput histograms

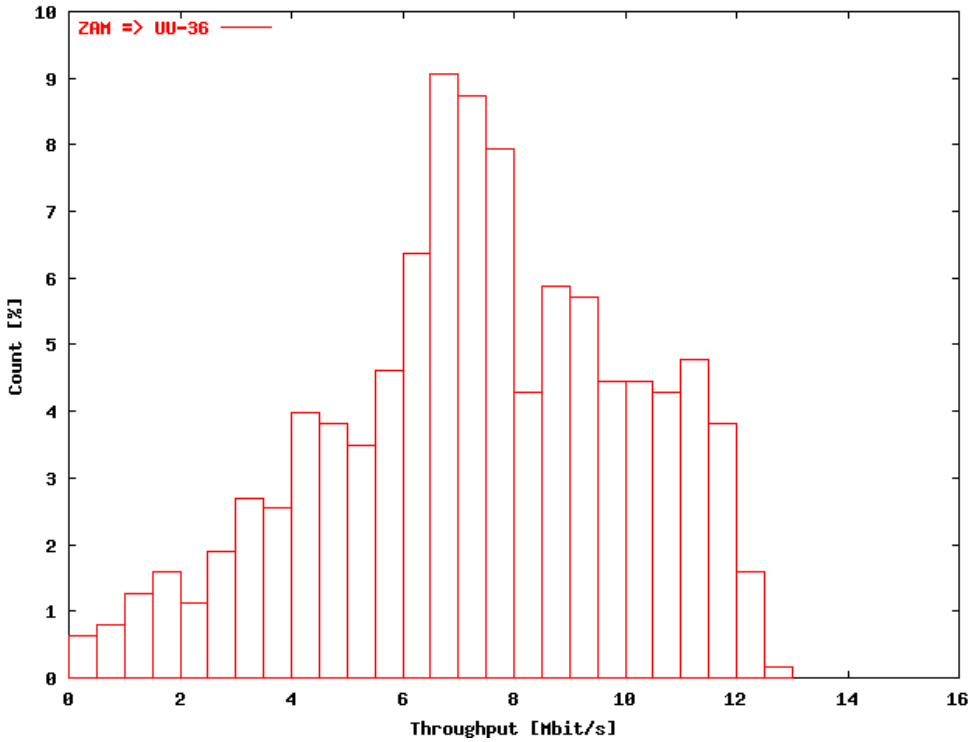
In this paragraph histograms from throughput counting are presented for the connections ZAM <=> UU-36. The bin counts are given as percentage from the total # of observations. The results are obtained for the first thirteen weeks of the year 2000, but only at working days (Mon - Fri)

The following histograms are presented: Figure 9 and Figure 10 show the histograms for connection ZAM => UU-36 and v.v. UU-36 => ZAM during working hours (08h - 18h);

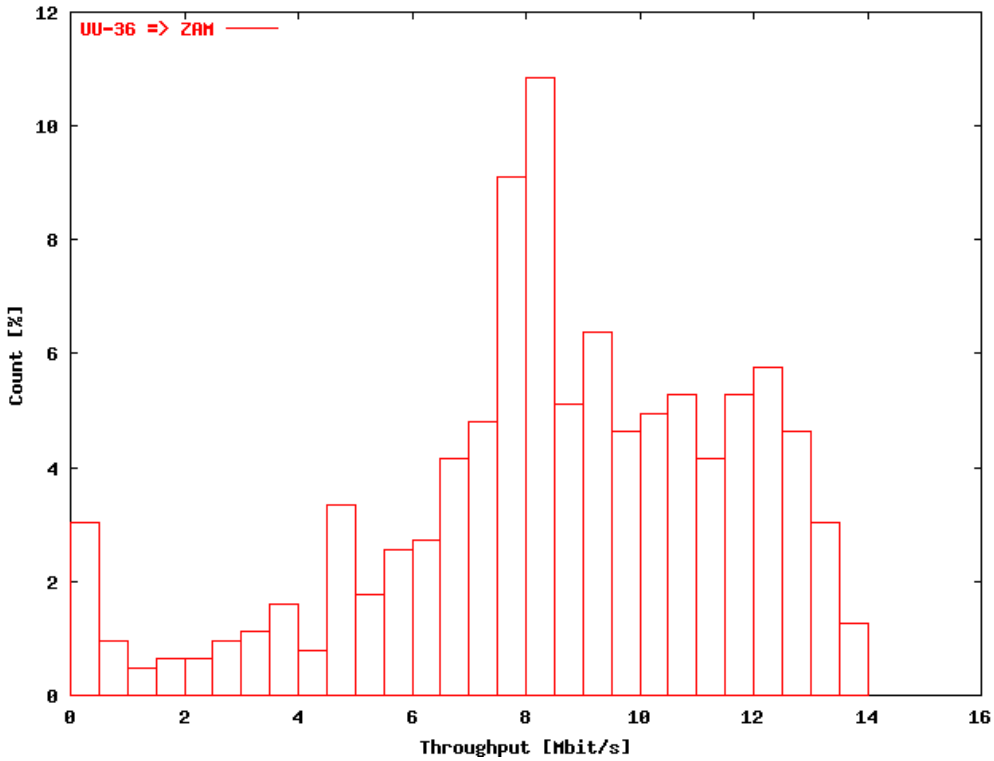
Figure 11 and Figure 12 display the histograms for connection ZAM => UU-36 and the reverse, UU-36 => ZAM, during the evening and night (18h - 24h; 00h - 08h).

The results lead to the following conclusions:

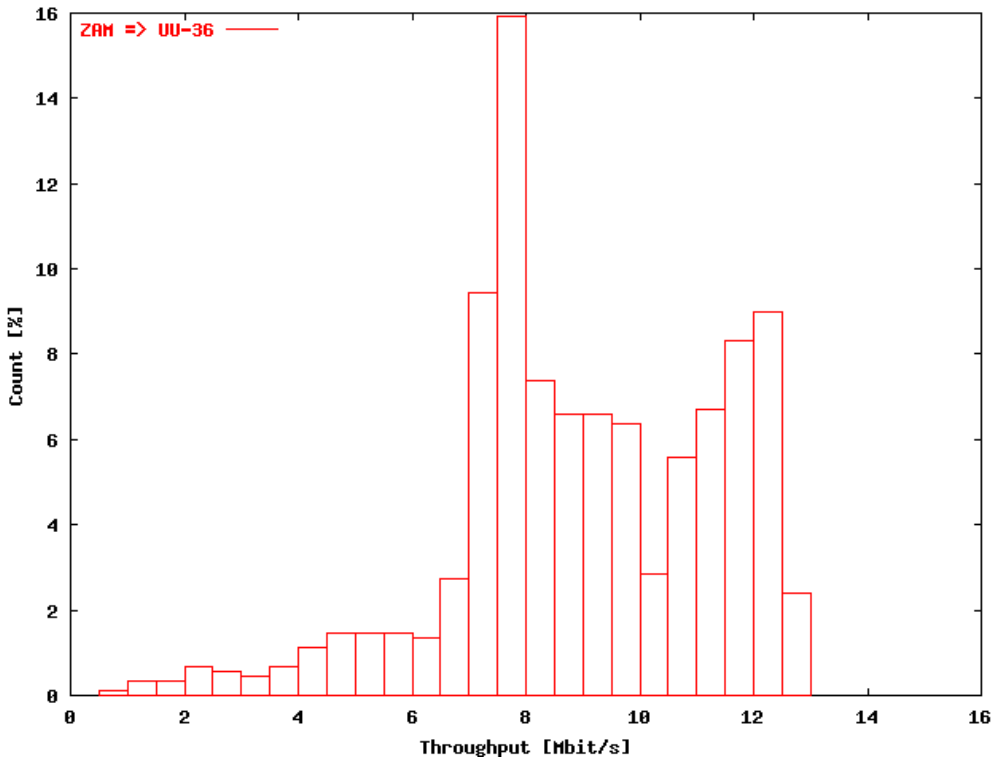
- In the evening and night the higher throughput bins are more frequently represented than during workday, as may be expected.
- As expected, during workday the lower bins (Tput 5 Mbit/s) are more filled, due to congestion, than during the evening and night.
- For the connection, UU-36 => ZAM there exists more heavily congestion (Tput 1Mbit/s) than for the ZAM => UU-36 connection.
- With the exception of ZAM => UU-36 at nighttimes, all histograms show a clear maximum (shifted to a larger bin at night compared to the working hours). The distribution of the higher throughput bins shows a shape similar to a Poisson distribution, probably due to router -> queue algorithms, while there exists a relative flat shape for the lower bins. In this area more incident driven protocols may play a role, like for instance packet retransmission.



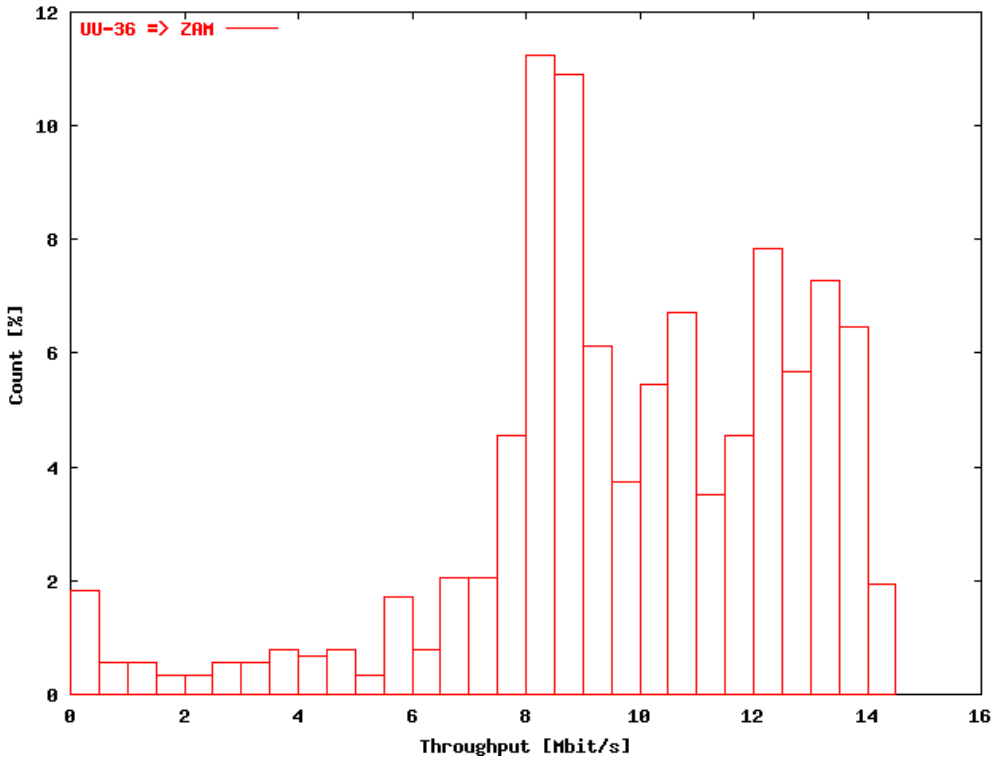
• Figure 9 Histogram of the throughput distribution during working days (08h - 18h) ZAM – UU



• Figure 10 Histogram of the throughput distribution during working days (08h - 18h) UU – ZAM



- Figure 11 Histogram of the throughput distribution during nights of working days working days (18h - 24h; 00h - 08h) ZAM - UU



- Figure 12 Histogram of the throughput distribution during nights of working days working days (18h - 24h; 00h - 08h) UU - ZAM

Overview time throughput averages

In this section we give an overview of the throughput average values, calculated at the hours, from all available workday data for a particular connection. The data are presented in the form of 3D plots, where the x-axis represents the hour and the y-axis the week of year (1999 and 2000). The plots for the following connections at workdays (Mon - Fri) are shown:

TEN-DE <=> TEN-NL

During a couple of weeks at the last half of 1999, hosts at the Frankfurt and the Amsterdam PoP of the TEN-155 network were added to be able to see the influence of router tuning in the throughput performance measurements. Figure 14 displays the performance of the TEN-DE => TEN-NL connection and Figure 15 the reverse connection. In both plots the data are averaged over the workdays of one week.

The following conclusions can be given:

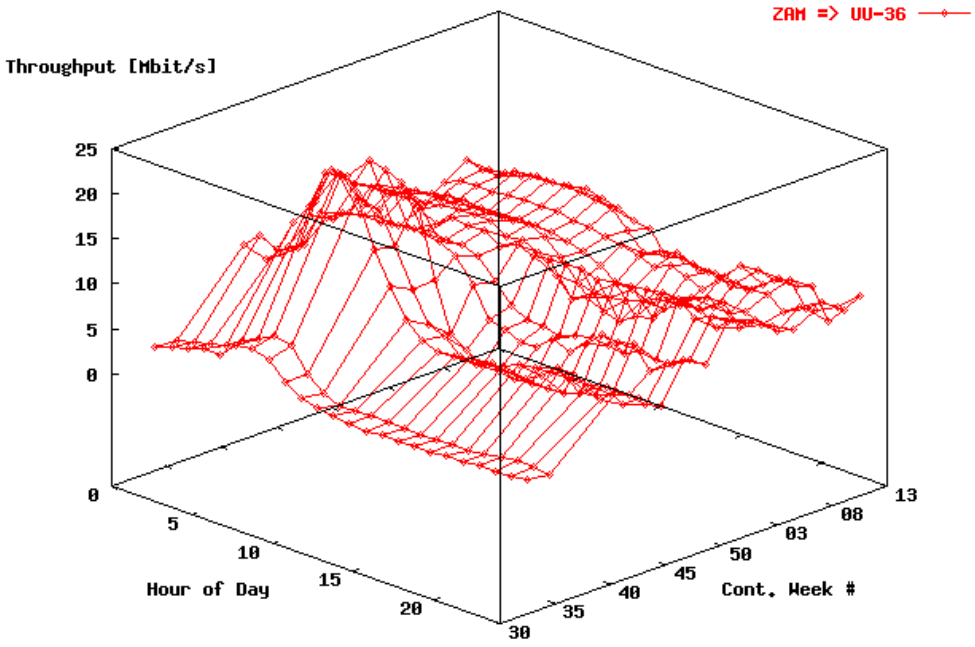
- Both plots clearly show the performance improvements due to the router tuning.
- In fact there were two stages in the tuning: after large improvements around week 42 (1999), there was also a tuning around week 48 where the performance during daytime was improved. Meanwhile also some high performance peaks especially for TEN-DE => TEN-NL) were flattened.

ZAM <=> UU-36

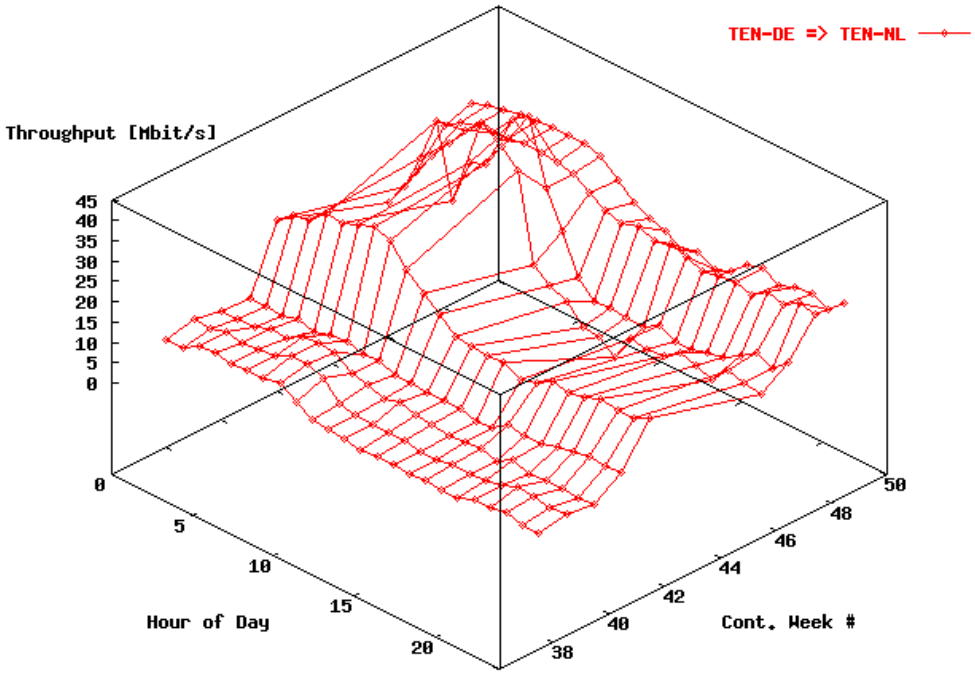
Error! Reference source not found. presents the hourly throughput values for the connection ZAM => UU-36 and Figure 16 for the reverse connection. In these plots the data are averaged over the workdays of two weeks.

The following conclusions can be given:

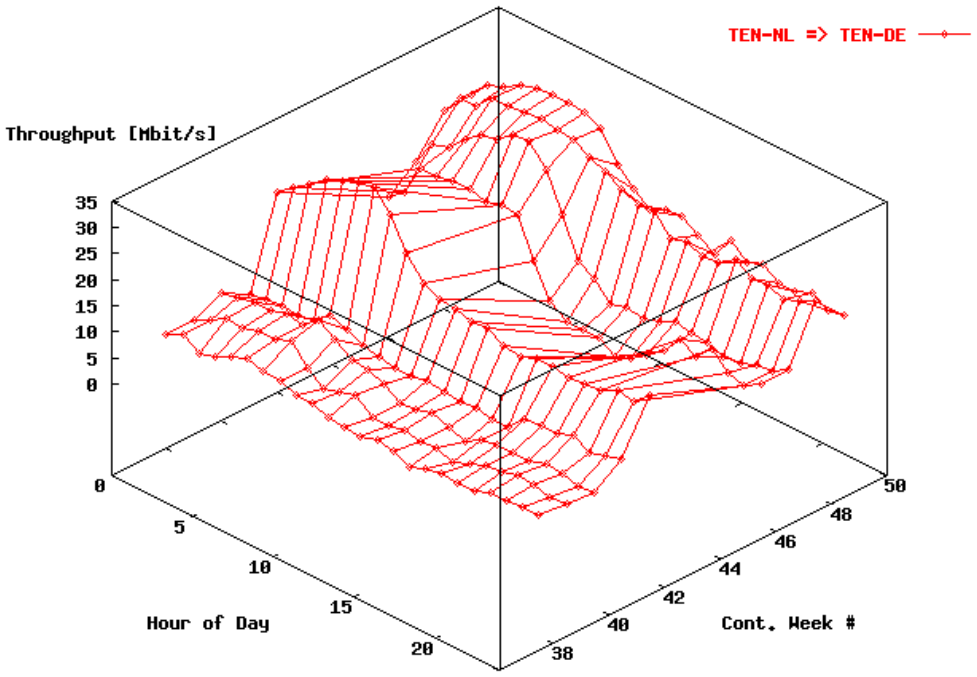
- The same conclusions are valid as for the TEN-DE <=> TEN-NL connections.
- Around week 52 1999 there is a maximum for all hours. This is caused by the traditional low seasonal traffic, especially in the Netherlands, in that period.
- In begin of 2000 there was a further improvement of the performance.



• Figure 13 Performance of the network between ZAM-FZJ Germany and UU, the Netherlands.

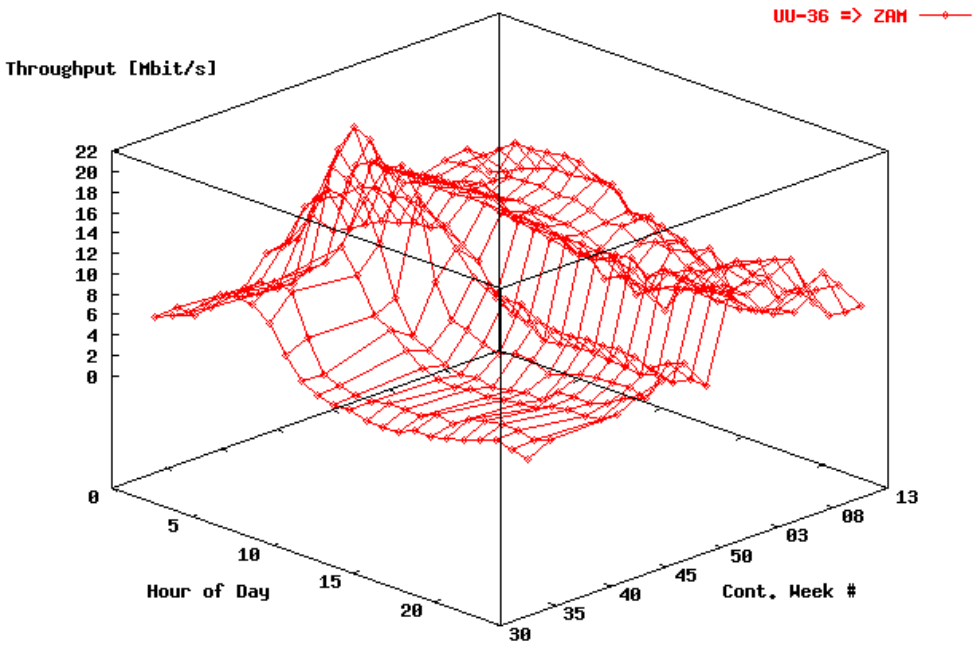


• Figure 14 Performance of the network backbone between Germany and the Netherlands



• Figure 15 Performance of the network backbone between the Netherlands and Germany

•



• Figure 16 Performance of the network between UU, the Netherlands and ZAM-FZJ Germany.

Bad performance events

- Table 1 Events with Tput < 0.5 Mbit/s for the connection ZAM <=> UU-36, vv

Date	Time	Site	Site	Tput	Ping			lost
dd/mm/yyyy	hh:mm:ss	1	2	Mbit/s	min[us]	avg[us]	max[us]	[%]
29/03/2000	17:00:08	UU-36	ZAM	0.32	31.200	57.075	85.500	5.000
29/03/2000	15:00:04	UU-36	ZAM	0.21	29.300	45.431	96.500	7.500
29/03/2000	13:00:02	UU-36	ZAM	0.02	23.500	46.862	144.000	2.500
29/03/2000	12:00:07	UU-36	ZAM	0.03	19.500	27.762	44.500	2.500
29/03/2000	11:00:03	UU-36	ZAM	0.07	30.500	60.181	81.500	5.000
29/03/2000	10:00:03	UU-36	ZAM	0.02	19.100	34.594	63.000	5.000
29/03/2000	09:00:03	UU-36	ZAM	0.08	18.300	23.492	45.000	2.500
29/03/2000	08:00:03	UU-36	ZAM	0.01	17.800	22.600	44.100	2.500
29/03/2000	07:00:04	UU-36	ZAM	0.02	16.500	20.341	112.000	2.500
29/03/2000	06:00:03	UU-36	ZAM	0.15	17.500	20.051	32.300	2.500
29/03/2000	05:00:03	UU-36	ZAM	0.02	17.300	18.611	20.100	0.000
29/03/2000	04:00:04	UU-36	ZAM	0.02	17.200	18.624	21.300	0.000
29/03/2000	03:00:03	UU-36	ZAM	0.01	17.000	19.868	61.000	0.000
29/03/2000	02:00:03	UU-36	ZAM	0.02	17.500	23.264	33.300	5.000
29/03/2000	01:00:03	UU-36	ZAM	0.14	17.400	20.060	52.600	7.500
29/03/2000	00:00:04	UU-36	ZAM	0.01	17.200	19.697	24.700	0.000
28/03/2000	23:00:02	UU-36	ZAM	0.03	17.100	19.224	22.900	2.500
28/03/2000	22:00:01	UU-36	ZAM	0.00	19.100	25.067	35.000	5.000
28/03/2000	21:00:02	UU-36	ZAM	0.00	17.500	20.013	28.800	0.000
28/03/2000	20:00:03	UU-36	ZAM	0.00	17.400	21.742	30.800	5.000
28/03/2000	19:00:02	UU-36	ZAM	0.01	17.700	19.561	23.200	0.000
28/03/2000	18:00:03	UU-36	ZAM	0.01	17.800	19.263	23.700	0.000
28/03/2000	17:00:01	UU-36	ZAM	0.02	18.300	25.151	53.900	7.500
28/03/2000	15:00:02	UU-36	ZAM	0.23	98.500	268.350	448.000	20.000
27/03/2000	18:00:04	UU-36	ZAM	0.34	18.300	20.792	23.800	0.000
22/03/2000	18:00:01	UU-36	ZAM	0.16	25.600	35.103	47.300	0.000
22/03/2000	12:00:06	UU-36	ZAM	***	83.000	121.276	152.000	0.000
20/03/2000	16:00:13	UU-36	ZAM	***	25.200	27.197	33.400	0.000
16/03/2000	16:00:06	UU-36	ZAM	0.10	25.300	31.614	44.700	5.000
16/03/2000	16:00:06	ZAM	UU-36	0.08	25.000	32.243	59.000	2.500
16/03/2000	15:00:05	UU-36	ZAM	0.00	25.100	27.881	34.700	2.500
16/03/2000	15:00:05	ZAM	UU-36	0.00	25.000	28.314	36.000	7.500
16/03/2000	14:00:12	UU-36	ZAM	0.02	24.400	27.229	37.300	0.000
16/03/2000	13:00:08	UU-36	ZAM	0.12	19.100	22.389	31.000	2.500
16/03/2000	12:00:02	UU-36	ZAM	0.04	18.700	21.543	29.000	2.500
16/03/2000	11:00:05	UU-36	ZAM	0.01	18.400	22.418	46.300	0.000
16/03/2000	10:00:02	UU-36	ZAM	0.48	334.000	422.105	505.000	0.000
16/03/2000	10:00:02	ZAM	UU-36	0.21	448.000	527.811	629.000	2.500
14/02/2000	15:00:04	UU-36	ZAM	0.28	44.200	47.850	51.300	10.000
08/02/2000	12:00:07	ZAM	UU-36	0.40	30.000	36.270	43.000	2.500
28/01/2000	09:00:05	UU-36	ZAM	0.46	30.500	36.554	43.300	2.500
24/01/2000	17:00:06	ZAM	UU-36	***	766.000	850.094	919.000	15.000
23/01/2000	14:00:06	ZAM	UU-36	0.03	25.000	52.667	83.000	5.000
20/01/2000	10:00:07	UU-36	ZAM	***	23.000	25.126	30.400	0.000
13/01/2000	16:00:07	UU-36	ZAM	***	17.500	19.463	23.300	0.000

Table 1 shows the monitor parameters for all events where Tput < 0.5 Mbit/s, which is a arbitrairy number, for the first thirteen weeks of 2000. Only the events for the ZAM <=> UU-36 connections are listed.

The following conclusions can be derived from this table:

- There are no structural performance decreases (collapses).
- The performance diminutions are clustered at the same dates. They are probably caused by network problems. This is especially the case for the events during the night.
- The most events are registered for the connection UU-36 => ZAM. They can also be observed as local maxima in the histograms for the corresponding bins.

• Table 2 Failures in the network listed according date / time for the last part of the reported period (end April 2000) for the connection FOM – IPP, vv

Date	Time	Site	Site	Tput		Ping		lost
dd/mm/yyyy	hh:mm:ss	1	2	[Mbit/s]	min[us]	avg[us]	max[us]	[%]
03/05/2000	12:30:05	FOM	IPP	0.08	21.511	31.684	100.014	2.500
02/05/2000	14:30:05	FOM	IPP	0.45	26.727	32.291	39.207	5.000
01/05/2000	01:30:04	FOM	IPP	***	94.804	95.665	96.626	0.000
01/05/2000	01:30:04	IPP	FOM	0.03	93.600	94.621	95.746	0.000
30/04/2000	14:30:05	FOM	IPP	***	94.999	95.503	96.045	67.500
30/04/2000	04:30:04	FOM	IPP	***	18.284	19.087	20.518	0.000
30/04/2000	04:30:04	IPP	FOM	0.31	17.550	18.313	19.540	0.000
29/04/2000	03:30:06	IPP	FOM	***	17.550	24.973	219.375	15.000
25/04/2000	15:30:04	FOM	IPP	0.35	25.996	31.196	35.238	10.000
14/04/2000	15:30:05	IPP	FOM	0.05	26.325	80.163	254.475	22.500
14/04/2000	14:30:05	FOM	IPP	0.20	28.613	36.875	41.483	0.000
14/04/2000	13:30:05	FOM	IPP	0.05	30.705	103.175	267.399	27.500
14/04/2000	13:30:05	IPP	FOM	0.14	31.200	114.903	240.342	30.000
13/04/2000	08:30:04	IPP	FOM	0.03	23.400	84.153	373.724	15.000
12/04/2000	14:30:05	FOM	IPP	0.46	30.107	49.902	137.668	10.000
11/04/2000	12:30:06	FOM	IPP	0.21	30.782	160.100	508.345	37.500
11/04/2000	12:30:06	IPP	FOM	0.16	25.350	108.707	253.500	22.500

Table 2 shows the events with Tput < 0.5 Mbit/s, for the connection IPP – FOM directly. Since the first week of April this connection was monitored once again. There is still not much statistics.

Overall Conclusions

- The connection Jülich - FOM / UU performs quite satisfactory. There are no structural performance decreases.
- The required bandwidth of 10 Mbit/s can only be obtained during the night. However, improvements in the TEN-155 network in the near future may help to improve this picture.

Video conferencing

Introduction

In this chapter we will deal once more with the recommended architecture for video conferencing that uses the "Armada Cruiser" hardware, present at the TEC partners. As explained in the previous report and here, this is still the best solution for point-to-point, quality video conferencing at limited bandwidth. We will describe the way in which the present tools and hardware can be used for multi-cast conferencing. This requires additional hardware at TEC, but could be tested using the Surfnet facilities, present at the UU[5].

We will also briefly comment on the "public domain" software solutions based on Mbone: VRVS, VIC and RAT

VCON MeetingPoint 4.01 with the RadVision MCU-323

From the tests of the last year we can conclude that hardware clients offer a very usable quality. Typically the following performance is measured:

- Video frame rate: 30 frames per second.
- Used bandwidth: 384 kilobit/second excluding data bandwidth (320 kilobit for video and 64 kilobit for audio).
- Video format: CIF, i.e. 352x288 pixels.
- Measured delay: approximately 0.5 seconds point-to-point for long distance connections. Not much difference is measured for European connections and connections from Europe to the U.S.

The tested MCU is a dedicated hardware device supporting up to 15 video calls and up to 24 audio only calls. The MCU comes with a software upload tool to upgrade the software from any windows 9x/NT machine. Our unit was configured software version 1.5 (build 1.5.0.6).

A single MCU, as described above, can support up to 9 simultaneous video calls. The tested MCU is still available as a "free-love" MCU, this means that people can connect to it when it is not used by SURFnet (its owner).

Information on when the MCU should be available and how to connect can be requested by e-mail: h.m.a.andree@phys.uu.nl

Guidelines for the use of the recommended "VC" hard- and software can be found in an appendix ()

[5] We already tested with Dr.Schorn (IPP-TEC) and he agrees that this is a relatively simple system that requires only a mouse click to connect.

VRVS, VIC and RAT

Introduction

The VRVS package (vrvs.cern.ch) includes the VIC (currently version 2.8) and RAT (3.0.29) tools for video and audio respectively. We evaluated the tools mentioned above using point-to-point connections. The use of a VRVS reflector offers multi-point facilities for these tools.

Tests

We tested the VIC and RAT tools on two PC's on different VLAN's but inside the same building. The audio latency, using the RAT tool was up to four seconds, especially when at the same time the VIC tool was running. When only one microphone was un-muted, no VIC tools were running and no audio driver was using full duplex, the delay was about 1 second.

The VIC tool is much quicker although (in case of a point-to-point connection) selection of video device and IP port numbers must be done by the end-user. It is also possible to use a config file, but the use of these tools is far less simple than that of the well-known H.323 systems like NetMeeting and VCON MeetingPoint. The quality of VIC is comparable to that of NetMeeting.

Conclusion

As far as VRVS concerns: VRVS is a server for the well known VIC/RAT tools. The end-to-end delay with RAT (audio) should be 1 second, not including transcoding in the server. This simply lies in the specification of the chosen CODEC for audio. With systems that work well with Netmeeting however, we measured much larger delays. As the VIC and RAT tools offer a quality that doesn't match that of hardware H.323 systems by any means we do not investigate the use of a VRVS server yet. Software conferencing systems may be very promising in the future, but at the moment only hardware systems seem to offer the quality and that is needed in future TEC collaborations.

Appendices

Guidelines for using VCON MeetingPoint 4.01 with the RadVision MCU-323

Introduction

This part describes the use of VCON MeetingPoint 4.01 in combination with the RadVision MCU-323. It is not meant as a general guide to VCON MeetingPoint. For a comprehensive guide to MeetingPoint we refer to the (online and written) documentation that accompanies the software.

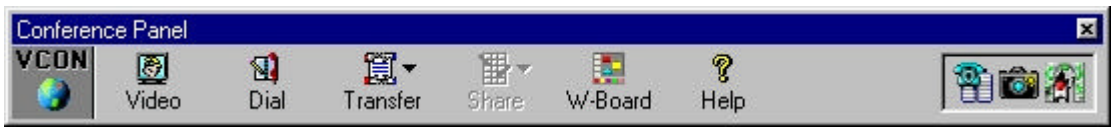
Neither is it intended to be a guide to MCU operators. We assume that within an organization that sets up an MCU, at least one technical person with knowledge of H.323 conferencing is available.

- Figure 17 Configuration panel. NetMeeting

Configuring MeetingPoint

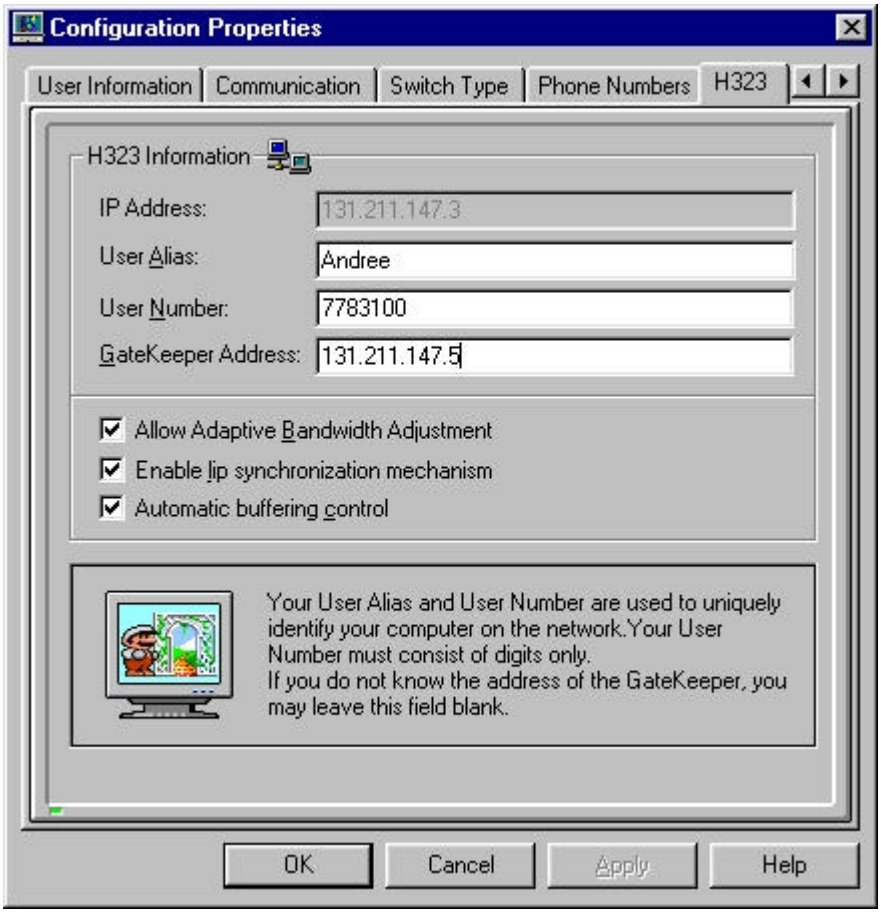
We assume that VCON MeetingPoint 4.01 is successfully installed and that the user is able to set-up point-to-point connections with other VCON systems. This means that the user is able to start the software and dial another user. Furthermore we assume that the MCU is installed and properly managed

If the software is started, the “Configuration Panel” (Figure 17) is visible. Depending on the chosen data-sharing application it can have a different look. The picture above shows the panel with the Microsoft NetMeeting date application whereas the picture below shows the panel for the standard date application. Both applications use the same data stack and are interoperable.



- Figure 18 Configuration panel. Standard Application.

The only MCU specific configuration need is to provide the software with gatekeeper registration information. To do this, click on the VCON logo in the "Conference Panel" and select "Configuration Properties". Then select the "H323" tab (see below) and fill-out the "User Number" and "Gatekeeper Address" fields. In our example the "User Number" equals 7783100 and the "Gatekeeper Address" is 131.211.147.5, but normally the MCU operator must provide these two values.



- Figure 19 Window for Configuration properties.

Now the MeetingPoint software has to be restarted to complete the configuration.

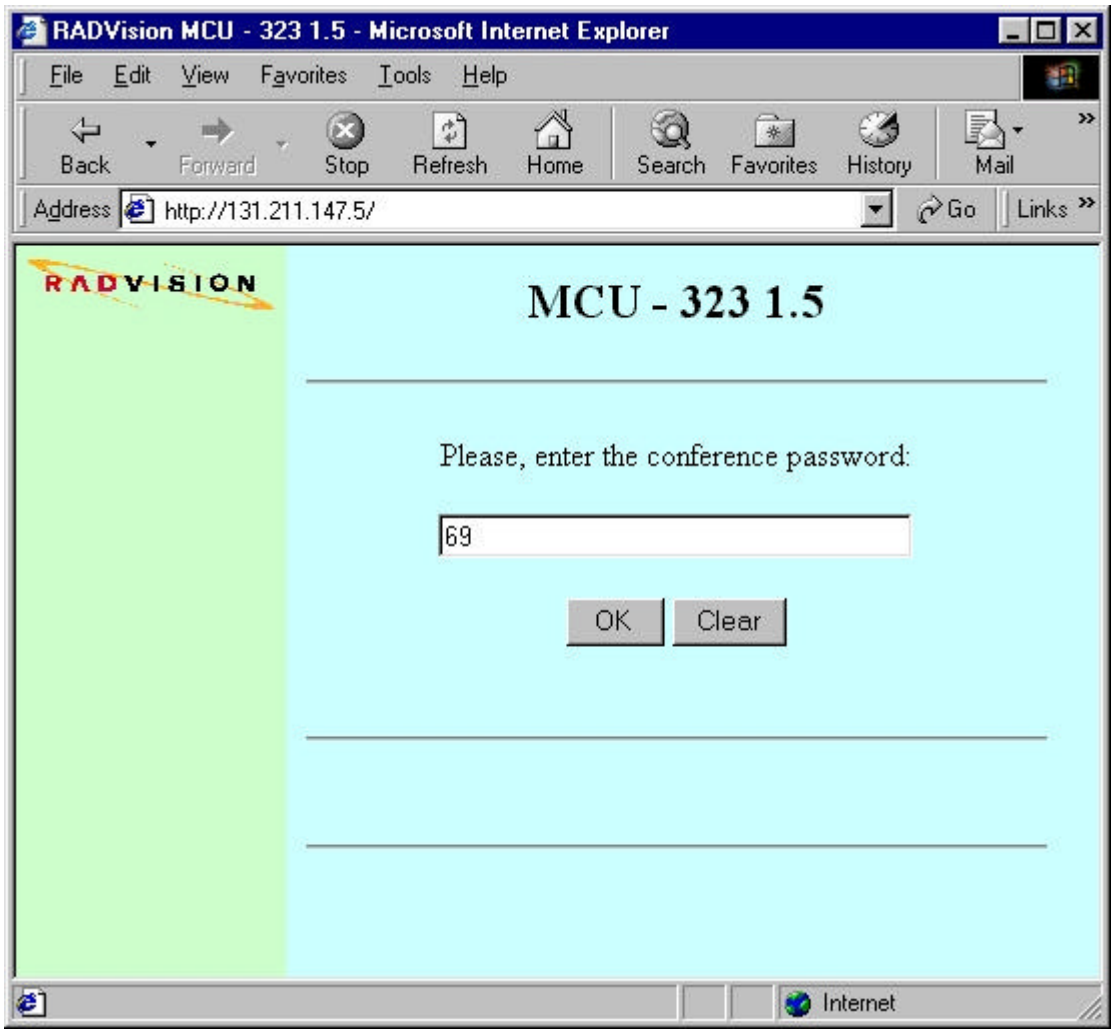
Setting up a call

To set up a call, click “Dial” in the “Conference panel” (Figure 20). This action opens a new window. In this window, click in “LAN Connect”, fill-out the conference number in the dialog box at the right side of the window and finally click on “Dial” just below this dialog box. The conference number (69 in our case) must be provided by the MCU operator. Normally the audio of all participants is mixed and the video is switched to the loudest speaker. Of course this switching does not affect the local video window and the first participant in a conference sees its own video in the remote video window until a second participant connects to the conference.

• Figure 20 Setting up a call

Starting the WWW interface

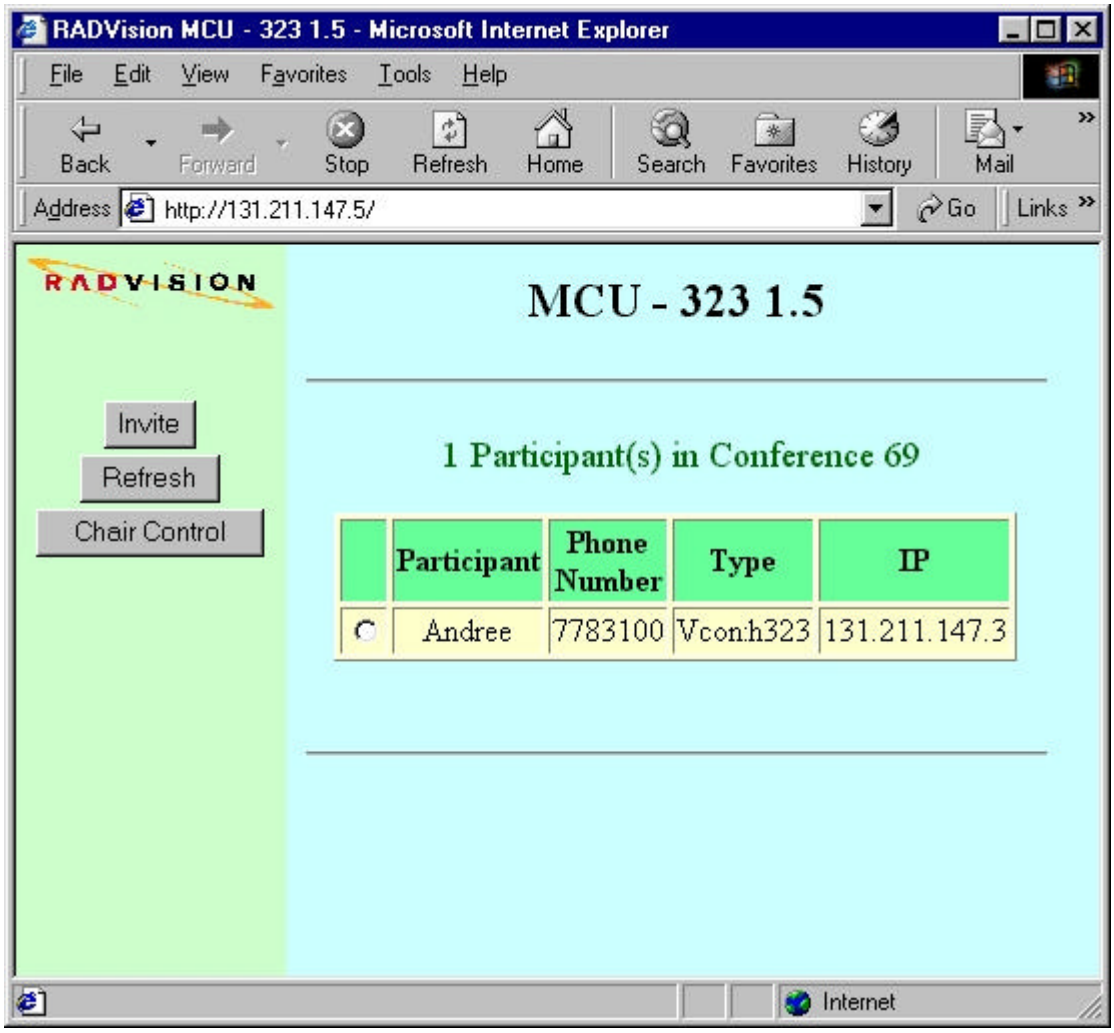
Each participant can use Internet Explorer (version 4 or higher) to see who is connected to the MCU and start data sharing. First the user must start the browser and fill-out the IP address of the MCU. In our case this is 131.211.147.5. (Figure 21)



• Figure 21 The WWW interface

As conference password (see above) the conference number (provided by the MCU operator) must be chosen. This conference number is the same as is used for dialling the conference.

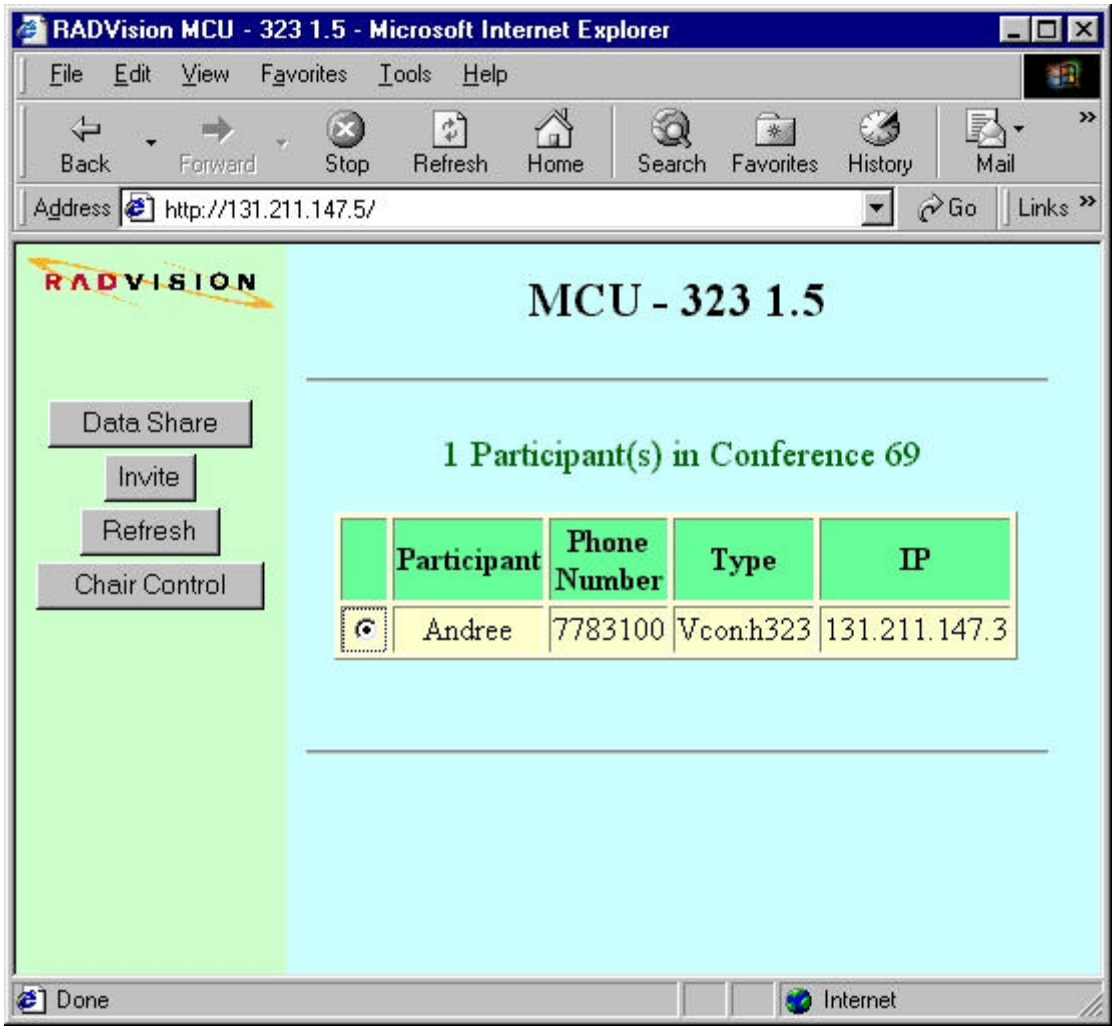
After clicking on “OK”, the participant is logged in to the WWW interface, and a list of connected users (see below) is shown.



• Figure 22 List of users connected to MCU

Setting up data sharing

In case that we want to use data sharing (T.120), the data connection must be started from the WWW interface using Internet Explorer. This feature is not available for Netscape. To start data sharing select another participant and click on the "Data Share" button that is now available in the left frame. (Figure 23)



• Figure 23 Data sharing

This action has to be repeated for every participant that is to be included in the T.120 (data sharing) part of the conference.

IDL Interface Description ObjectManager

ObjectManager.idl

```
// Version 1.0.1 : updated 23-02-2000
//

#ifndef OBJECTMANAGER_idl
#define OBJECTMANAGER_idl

#include "DataManager.idl"
//#include "Diagnostic.idl"


typedef string DataManagerType ;


interface ObjectManager
{
 typedef sequence<string> StringSeq ;
 typedef sequence<long>   LongSeq ;

 enum OmResultType
 {
  OperationSuccesful,
  AuthorisationFailed,
  InvalidRequest,
  UnknownDataManagerType,
  InternalError
 } ;

 exception Error
 {
  OmResultType type ;
  string       message ;
 } ;

 struct OmListStruct
 {
  OmResultType  Result    ;
  string        strResult ;
  StringSeq     Names     ;
  LongSeq       IDs       ;
  long          iSize     ;
 } ;

 struct OmListReqStruct
 {
  CryptoSeq Key     ;
 } ;

 struct OmConStruct
 {
  OmResultType Result    ;
  string       strResult ;
  string       IOR       ;
 } ;
```

```
 struct OmConReqStruct
 {
  CryptoSeq Key       ;
  long      iID       ;
 } ;


 boolean GetList(
           in  OmListReqStruct ListRequest,
           out OmListStruct    ListResult )
         raises( Error ) ;

 boolean GetDataManager(
           in  OmConReqStruct  ConnectionRequest,
           out OmConStruct     ConnectionResult )
         raises( Error ) ;


};

#endif
```

IDL Interface Description DataObject

DataObject.idl

```
#ifndef DATAOBJECT_idl
#define DATAOBJECT_idl

//
// Typedefs
//
typedef sequence<octet> ByteSeq;

typedef sequence<short> ShortSeq;
typedef sequence<unsigned short> UShortSeq;
typedef sequence<long> LongSeq;
typedef sequence<unsigned long> ULongSeq;
typedef sequence<float> FloatSeq;
typedef sequence<double> DoubleSeq;
typedef sequence<string> StringSeq;

typedef long TimeStamp;

//
// Enumerate types
//
enum ObjectType
{
  cUnknown,
  cComment,
  cMimeObj,
  cPolyCalibration,
  cTableCalibration,
  cShortBase,
  cLongBase,
  cScalar,
  cDim1Int8,
  cDim1Int16,
  cDim1Int32,
  cDim1UInt16,
  cDim1UInt32,
  cDim1Float32,
  cDim1Float64,
  cDim2Int8,
  cDim2Int16,
  cDim2Int32,
  cDim2UInt16,
  cDim2UInt32,
  cDim2Float32,
  cDim2Float64,
  cDimNInt8,
  cDimNInt16,
  cDimNInt32,
  cDimNUInt16,
  cDimNUInt32,
  cDimNFloat32,
  cDimNFloat64
};
```

```
enum AccessMode
{
  cNone,
  cRead,
  cWrite,
  cReadWrite,
  cPol,
  cPolRead,
  cPolWrite,
  cPolReadWrite
};

//
// Structs used in the data objects
//

struct Policy
{
  unsigned short gid;
  AccessMode mode;
};

typedef sequence<Policy> PolicySeq;

struct RevInfo
{
  long time;
  string username;
  string description;
};

typedef sequence<RevInfo> RevInfoSeq;

struct SiUnits
{
  long kg, m, s, A, cd, mol, K, rad, sr;
};

struct ObjectHeader
{
  string name;
  unsigned long level;
  unsigned long quality;
  string fullPath;
  StringSeq references;
  ObjectType type;
};

//
// Struct for actual data objects
//

struct Comment
{
  ObjectHeader oh;
  string content;
};

struct ShortBase
{
  ObjectHeader oh;
```

```
  SiUnits unit;
  double start;
  double step;
};

struct LongBase
{
  ObjectHeader oh;
  SiUnits unit;
  DoubleSeq data;
};

struct PolyCalibration
{
  ObjectHeader oh;
  DoubleSeq coefficients;
};

struct TableCalibration
{
  ObjectHeader oh;
  DoubleSeq table;
};

struct MimeObj
{
  ObjectHeader oh;
  string mimetype;
  unsigned long bytecount;
  ByteSeq content;
};

struct Scalar
{
  ObjectHeader oh;
  SiUnits unit;
  double time;
  double content;
};

struct DimNInt8
{
  ObjectHeader oh;
  SiUnits unit;
  ULongSeq sizes;
  StringSeq bases;
  string calibration;
  unsigned short adcresolution;
  boolean sign;
  ByteSeq content;
};

struct DimNInt16
{
  ObjectHeader oh;
  SiUnits unit;
  ULongSeq sizes;
  StringSeq bases;
  string calibration;
  unsigned short adcresolution;
  ShortSeq content;
```

```
};

struct DimNInt32
{
  ObjectHeader oh;
  SiUnits unit;
  ULongSeq sizes;
  StringSeq bases;
  string calibration;
  unsigned short adcresolution;
  LongSeq content;
};

struct DimNUInt16
{
  ObjectHeader oh;
  SiUnits unit;
  ULongSeq sizes;
  StringSeq bases;
  string calibration;
  unsigned short adcresolution;
  UShortSeq content;
};

struct DimNUInt32
{
  ObjectHeader oh;
  SiUnits unit;
  ULongSeq sizes;
  StringSeq bases;
  string calibration;
  unsigned short adcresolution;
  ULongSeq content;
};

struct DimNFloat32
{
  ObjectHeader oh;
  SiUnits unit;
  ULongSeq sizes;
  StringSeq bases;
  FloatSeq content;
};

struct DimNFloat64
{
  ObjectHeader oh;
  SiUnits unit;
  ULongSeq sizes;
  StringSeq bases;

  DoubleSeq content;
};

#endif
```

IDL Interface Description DataManager

DataManager.idl

```
// Version 1.2.1 : updated 23-02-2000
//

#ifndef DATAMANAGER_idl
#define DATAMANAGER_idl

#include "DataObject.idl"

//
// Typedefs
//
typedef sequence<octet,128> CryptoSeq;

interface DataManager
{
  enum DmErrType
  {
    NoTransaction,
    NestedTransaction,
    PermissionDenied,
    IllegalPath,
    IllegalMode,
    NoSuchObject,
    ObjectExists,
    LockTimeout,
    LockNotActive,
    InvalidType,
    InternalError,
    ServiceNotAvailable,
    SecurityError
  };

  exception Error
  {
   DmErrType type ;
   string message ;
  };

  exception CannotProceed
  {
   DataManager NewContext ;
   string      RestOfPath ;
  };

  enum Interpolation
  {
   None,
   Average,
   MinMax
  };

  const unsigned long maxIdleTime = 3600 ;
  readonly attribute unsigned long idleTime ;

  // Transaction operations
  void start(in CryptoSeq signature)
```

```
    raises(Error);
  void commit(in CryptoSeq signature)
    raises(Error);
  void abort(in CryptoSeq signature);
  void commitAndHold(in CryptoSeq signature)
    raises(Error);

  // Data object operations
  void store(in any obj, in string path, in CryptoSeq signature)
    raises(Error);
  void update(in any obj, in string path, in boolean headerOnly,
         in string info, in CryptoSeq signature)
    raises(Error);
  RevInfoSeq getHistory(in string path, in CryptoSeq signature)
    raises(Error);
  PolicySeq getPolicies(in string path, in CryptoSeq signature)
    raises(Error);
  ObjectHeader getHeader(in string path, in CryptoSeq signature)
    raises(Error);
  any getProperties(in string path, in CryptoSeq signature)
    raises(Error);
  any getData(in string path, in CryptoSeq signature)
    raises(Error);
  DimNFloat64 getDim1Data(in string path, in unsigned long first,
                  in unsigned long npoints, in unsigned long
interval,
                  in Interpolation how, in CryptoSeq signature)
    raises(Error);
  DimNFloat64 getDim2Data(in string path,
                          in unsigned long x_first,
                          in unsigned long x_npoints,
                          in unsigned long x_interval,
                          in unsigned long y_first,
                          in unsigned long y_npoints,
                          in unsigned long y_interval,
                  in Interpolation how,
                  in CryptoSeq signature)
    raises(Error);
  void setPolicy(in string path, in Policy p, in CryptoSeq signature)
    raises(Error);
  void rm(in string path, in CryptoSeq signature)
    raises(Error);
  void lock(in string path, in CryptoSeq signature)
    raises(Error);
  void unlock(in string path, in CryptoSeq signature)
    raises(Error);
  void link(in string srcpath, in string dstpath, in CryptoSeq
signature)
    raises(Error);

  // Directory operations
  StringSeq list(in string path, in CryptoSeq signature)
    raises(Error, CannotProceed) ;

  // Other operations
  void keepAlive(in CryptoSeq signature)
    raises(Error);
  oneway void shutdown(in CryptoSeq signature);
};

#endif
```