ANNALS OF THE FACULTY OF ENGINEERING HUNEDOARA

2005, Tome III, Fascicole 2

CLIENT SIDE ARCHITECTURE OF HIGH SPEED DATA ACCESS SERVER IN DISTRIBUTION MANAGEMENT SYSTEMS

Aleksandar ERDELJAN, Dragan S. POPOVIĆ,
Imre LENDAK, Darko ČAPKO

UNIVERSITY OF NOVI SAD,
FACULTY OF TECHNICAL SCIENCES,
SERBIA & MONTENEGRO

Abstract:
The aim of this paper is to describe the client side architecture of a high-speed data access server managing data in a Distribution Management System (DMS). The presented solution is adapted to the specific DMS requirement, and it simplifies client applications. The main design issue was to enable both fast server response and access to large numbers of items in an efficient way. The design of the interfaces is based on existing DAIS/OPC standards, slightly modified to satisfy the specific DMS requirements. The HSDA server was implemented using CORBA middleware and tested on various Microsoft Windows and Unix/Linux operating systems, including HP Tru64 Unix on Alpha machines.

Key words:
Data Access servers, Distribution Management Systems, DAIS, CORBA

1. INTRODUCTION

Distribution has the highest software demands in the electric utility industry, dealing with large numbers of items (usually hundreds of thousands). To deal with this plethora of items, powerful Distribution Management Systems (DMS) [1] are needed. They have to monitor and control power delivery equipment, ensure system reliability, voltage management, demand-side management, outage management, work management, automated mapping and facilities management. All these concepts have to be represented in DMS software, and they have to be easily reachable by a wide range of client applications.

This paper partially presents the design of a High Speed Data Access server, which serves as a link between the data in a DMS and its clients. Due to the lack of space, this paper focuses only on the client side interfaces of the HSDA server component. The internal structure of the server is not discussed.

The task of the HSDA server is to manage the actual network state data consisting of frequently changing values, e.g. measurements, switchgear statuses, tap changer positions etc. Although a smaller part of these values is automatically collected from field devices (through SCADA subsystems), the majority of network state data consists of items whose values are manually entered/changed by operators.

Typical clients of the HSDA server are applications that run a set of DMS analytic functions or GUI operation management applications which implement a graphical representation of network state and DMS function results. Whether new item values come from the field or are entered manually, the network state is changed. These changes should be quickly dispatched to all clients in order to trigger the recalculation of DMS analytic functions.

The network state consists of a collection of variables - items, and the role of the HSDA server is to enable clients to read and write these items' values in an efficient way. DMSs use different item types: measurements, switch statuses, tap changer and capacitor changer positions, fuse statuses, relay protection settings, etc.

The proposed HSDA server stores the values of these items in structures, e.g. the switchgear structure contains: switch status (open, closed, in transition) that is normally read from a field device, and "locked for operation" and "out of order" fields that can be set by the operators.

DA server interfaces should enable:

- Transfer of many item values of different types in same vector,
- Client subscription to item value changes,
- Redundancy to avoid single point of failure.

1.1. Technologies used

A middleware was needed for the communication between the HSDA server and its clients. Although OPC [2] is a *de facto* standard for inter-process communications in industrial software systems, it has one major weaknes: its tight coupling with DCOM, which is a middleware implemented on MS Windows platforms only.

As one of the mandatory requirements was that the HSDA server should work on various operating systems and that it should support cross platform connections between clients and servers, OPC was ruled out as inappropriate.

Recent OPC specifications like OPC DX, OPC XML allow cross platform communication utilizing XML [3]. Unfortunately, due to a request for a fast server response, the overhead, which XML introduces, is not acceptable for this kind of applications.

1.2. The solution

The solution, which seems the most appropriate basis for a cross-platform HSDA server is the DAIS (Data Acquisition from Industrial Systems) [4] specification. It adheres to OPC, simplifying the porting of existing OPC DA solutions to DAIS, and it is implemented on top of CORBA middleware (Common Object Request Broker Architecture) [5]. CORBA is an industry standard defined by the Object Management Group (OMG), which has operating system independent and open source implementations (e.g. omniORB [6]).

DAIS describes both interfaces and an information model. Because of the specific needs imposed by the DMS, certain changes needed to be made to the interfaces and model proposed by DAIS:

1. *It was necessary to use structures for data transmission instead of DAIS's SimpleValue type, because DMS item types are structures that can't fit into a SimpleValue. Another reason for this change were possible performance and memory usage issues*
2. *Item groups were left out, because DMS clients need all item changes, not only a subset of those*

Another important reason for not complying fully to DAIS was the complexity of DAIS, which would have prolonged development time. Therefore it was decided to modify DAIS for the specific needs imposed on the HSDA server by a DMS (high speed, structures instead of SimpleValues, no groups).

2. CLIENT SIDE ARCHITECTURE

In this chapter the client side architecture of the High Speed Data Access server will be proposed and discussed. Client-server communication is realized through a set of interfaces, published to various client applications.

The design of these interfaces is based on the DAIS specification, but the specific DMS needs made it necessary to make slight modifications. We will discuss the differences between the proposed solution and the interfaces defined in DAIS.

Figure 1. shows the interfaces which are needed to be implemented by the client (on the left hand side of the picture), and by the server (on the right hand side of the picture) in order to address the requirements stated.

Client software components need to implement two interfaces, through which they can receive asynchronous data callbacks and important server messages (e.g. shutdown): *IAsyncIOCallback* and *IServerCallback*. These will be discussed in detail below.

The most important interfaces, frequently used by clients are the following: *IServer*, *IDASession*, *ISyncIO* and *IAsyncIO*. These interfaces will be discussed in detail

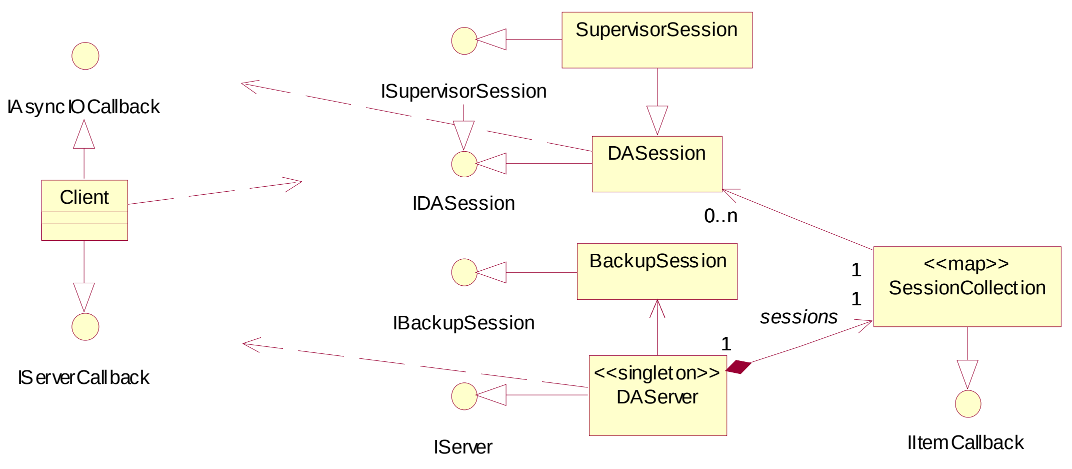


Figure 1. Client interfaces and server session interfaces

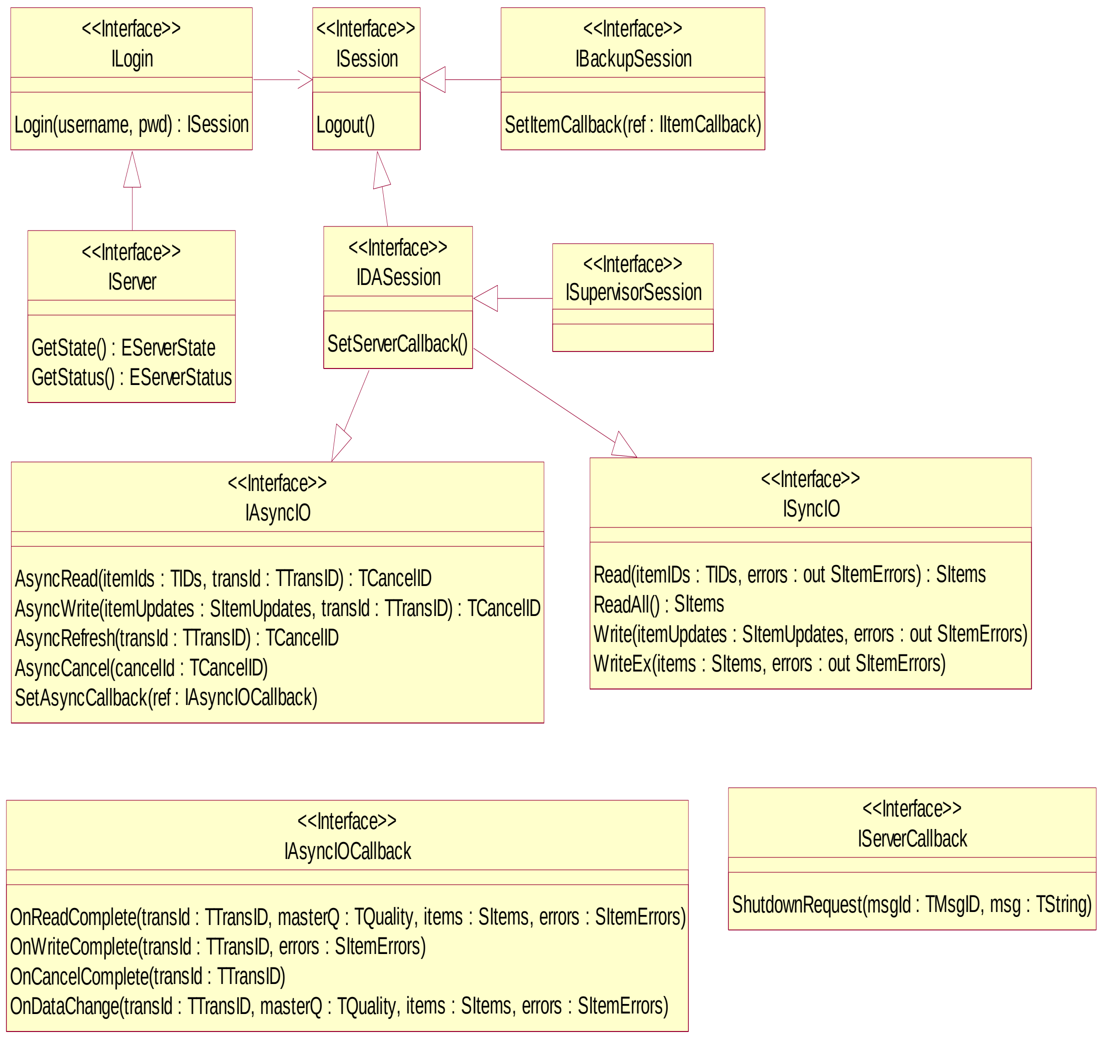

<<Interface>>
ILogin
Login(username, pwd) : ISession
<<Interface>>
ISession
Logout()
<<Interface>>
IBackupSession
SetItemCallback(ref : IItemCallback)
<<Interface>>
IServer
GetState() : EServerState
GetStatus() : EServerStatus
<<Interface>>
IDASession
SetServerCallback()
<<Interface>>
ISupervisorSession
<<Interface>>
IAsyncIO
AsyncRead(itemIds : TIDs, transId : TTransID) : TCancelID
AsyncWrite(itemUpdates : SItemUpdates, transId : TTransID) : TCancelID
AsyncRefresh(transId : TTransID) : TCancelID
AsyncCancel(cancelId : TCancelID)
SetAsyncCallback(ref : IAsyncIOCallback)
<<Interface>>
ISyncIO
Read(itemIDs : TIDs, errors : out SItemErrors) : SItems
ReadAll() : SItems
Write(itemUpdates : SItemUpdates, errors : out SItemErrors)
WriteEx(items : SItems, errors : out SItemErrors)
<<Interface>>
IAsyncIOCallback
OnReadComplete(transId : TTransID, masterQ : TQuality, items : SItems, errors : SItemErrors)
OnWriteComplete(transId : TTransID, errors : SItemErrors)
OnCancelComplete(transId : TTransID)
OnDataChange(transId : TTransID, masterQ : TQuality, items : SItems, errors : SItemErrors)
<<Interface>>
IServerCallback
ShutdownRequest(msgId : TMsgID, msg : TString)

Figure 2. Client side interfaces of the HSDA server

2.1. IServer

The *IServer* interface is the main entry point of the server. Clients need to acquire a pointer to this interface in order to register a session at the server. Sessions are set up by logging in through the *ILogin* interface. After logging in through this interface, the client can set up:

1. a data access session defined in the *IDASession* interface, or
2. a supervisor session defined in *ISupervisorSession*

The most important interfaces, which can be acquired through IServer are shown in Figure 2.

2.2. IDASession

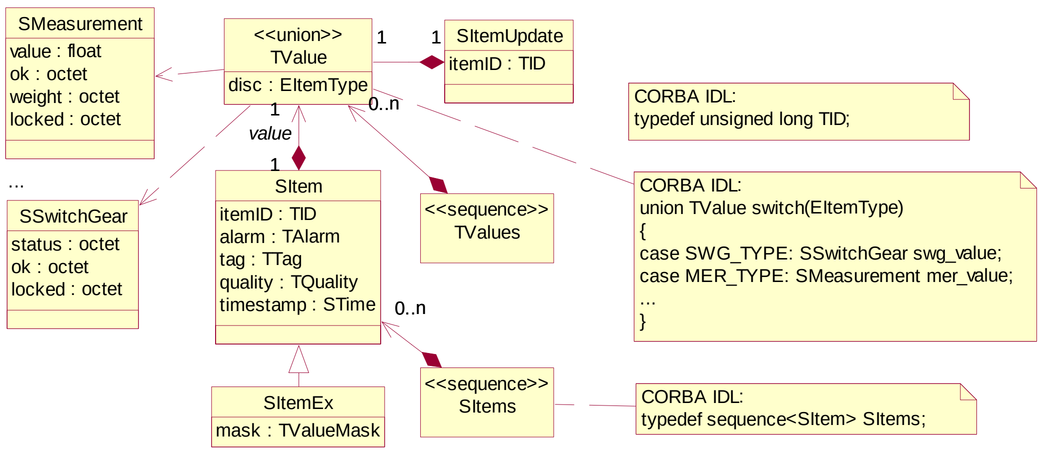
In the majority of cases clients will register data access (DA) sessions at the server. This type of session is defined in the *IDASession* interface. It provides the client with access to item values. Since the HSDA server enables connections of several clients at the same time, the number of instances of *DASession* objects depends on the number of concurrently connected clients and their activities. These sessions are held in a *SessionCollection* container, which also has the role of dispatching new item values received on the internal *IItemCallback* interface.

A typical client application uses the *ISyncIO* and *IAsyncIO* interface to read or write items, and implements the *IAsyncIOCallback* interface, through which it subscribes to item changes. When the server is shutting down all clients are informed through their *IServerCallback* interfaces, i.e. kindly asked to release all server interfaces they are holding at the moment.

It is important to notice that the solution proposed by this paper introduces a considerable amount of changes to DAIS's item description mechanisms. The various "find" methods proposed by DAIS were left out in order to shorten the time needed for development. Also, instead of DAIS's Item interface, we propose three different kinds of item structures (see Fig. 3):

- *SItemUpdate* holds an item's ID and its value (*TValue* type);
- *SItem* extends S*ItemUpdate* by adding the folowing values: alarm, tag, quality and timestamp
- *SItemEx* further extends the *SItem* structure with a "value mask" field to indicate which element of the structure has changed.

Another cause for the introduction of these item types was that the DMS must handle complex data types, some containing sequences of other values. This fact made it necessary to model these irregular item types with the above listed three special types.



```
CORBA IDL:
typedef unsigned long TID;
```

```
CORBA IDL:
union TValue switch(EItemType)
{
case SWG_TYPE: SSwitchGear swg_value;
case MER_TYPE: SMeasurement mer_value;
...
}
```

```
CORBA IDL:
typedef sequence<SItem> SItems;
```

Figure 3. Proposed item types

To make data transfer more efficient, it was decided not to transfer single items, but to group multiple items into CORBA sequences. These types were named by adding an additional 's' to the end of the item type name, e.g. a sequence holding *SItem* values is named *SItems* (see Fig 3.). Communication between the server and the clients can be synchronous or asynchronous. These two types of communications are realised through methods of the *ISyncIO* and the *IAsyncIO* interfaces.

2.4. ISyncIO

Synchronous input/output operations are grouped into the *ISyncIO* interface. The methods of this interface allow to read/write a certain item held in the HSDA server and to read all item values in one call. The ReadAll method is not included in DAIS, and although its functionality can be achieved by DAIS's Read method, it was added because there are use cases when clients need to read all item values in one call (usually at client startup).

2.4. IAsyncIO

The methods in the *IAsyncIO* interface allow asynchronous data access. *IAsyncIO* implements the methods proposed by DAIS (*AsyncRead*, *AsyncWrite*, *AsyncRefresh*, *AsyncCancel*) with the exception of special exception handling.

2.5. Additional interfaces

Although the *ISupervisorSession* and *IBackupSession* are not used in direct data communication between the server and clients, they will be described in a few words:

- the *ISupervisorSession* interface contains methods which are used by administrators for monitoring and administrative purposes, e.g. suspend or shutdown operations.

- the methods of the *IBackupSession* interface are used for establishing a connection between two HSDA servers enabling them to run in a redundant configuration.

2.6. Client connection primer

Figure 4. shows all the steps the client and the server perform during session setup. The first step in setting up a data access session is client login. After a valid login, the server creates a new data access session, and adds it to its *SessionCollection* container.

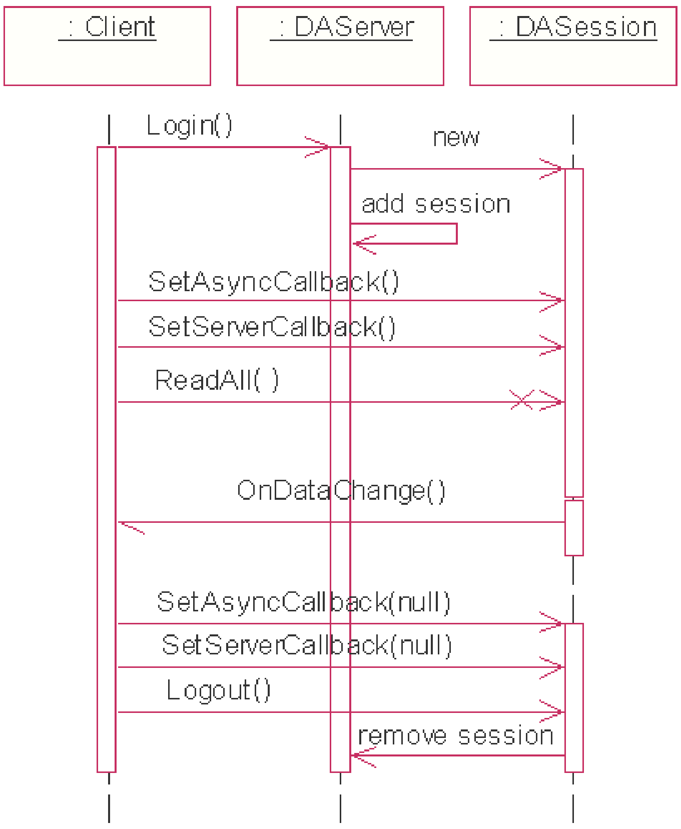


Figure 4. Client session setup

Clients initialize their view of the network state by acquiring all item values in one call (*ReadAll*). In order to receive future data changes in an asynchronous manner, clients register their *IAsyncIOCallback* and *IServerCallback* interfaces at the server by calling *SetClientCallback* and *SetServerCallback*.

It is important to notice that this type of communication, in which after an initial acquisition of all item values in a single call, clients receive future item changes through asynchronous callbacks, is very efficient and has low bandwidth requirements.

Should a client need to disconnect from the server, it asks the server to release its (the client's) callback interfaces (calling *SetServerCallback* and *SetAsyncCallback* with a NULL arguments) and initiates the release of the data access session by logging out (*Logout*).

3. CONCLUSION

The proposed architecture of the client side of the HSDA server has its advantages. It was designed with performance and resource usage in mind, therefore implements only a slightly modified subset (no alarms and events, no DAIS exceptions) of the functionality proposed by DAIS.

The flow of messages between the server and the clients is natural: login, create a DA session, do synchronous/asynchronous read/write operations and receive asynchronous data change updates. This simplicity in turn can ease client side application development..

Another major advantage of the proposed solution is that its implementation, written in C++, is fully cross-platform, tested on top of omniORB on different operating systems, including various Windows and UNIX/Linux operating systems, with extremely good results. The decision to use structures instead of SimpleValue objects (as proposed by DAIS) has resulted in an extremely fast HSDA server, which can meet the needs of a large DMS with hundreds of thousands of items.

Further work on the proposed client side architecture of the HSDA server could involve changing it in a way to make it CIM [7] compliant.

4. REFERENCES

[1.] D.S.Popovic, “Power Application - A Cherry on the Top of the DMS Cake”, *DA/DSM DistribuTECH Europe 2000*, Vienna, Austria, October 10-12, 2000, Specialist Track 3, Session 3, Paper 2.

[2.] OPC Foundation, “OPC Data Access Custom Interface Specification 2.0”, 1998

[3.] “Extensible Markup Language (XML) 1.0”, W3C Recommendation, Feb.1998, http://www.w3.org/TR/REC-xml

[4.] “Data Acquisition from Industrial Systems (DAIS)”, OMG specification, http://mantis.omg.org

[5.] “Common Object Request Broker Architecture (CORBA) 2.6”, OMG specification, http://www.omg.org/

[6.] omniORB (version 4.0.5) documentation, http://omniorb.sourceforge.net/

[7.] “Common Information Model (CIM): CIM 10 Version” EPRI, Palo Alto, CA, 1001976, 2001.