ANNALS OF THE FACULTY OF ENGINEERING HUNEDOARA
2003, Tome I, Fascicole 3

AN OPC DATA ACCESS SERVER DESIGNED FOR LARGE NUMBER OF ITEMS

Aleksandar ERDELJAN, Nebojša TRNINIĆ, Darko ČAPKO

FACULTY OF ENGINEERING, NOVI SAD

Abstract *- Servers that support OPC specifications are part of big distributed control systems of SCADA systems nowadays. The paper describes the design of an OPC Data Access server which has an address space that contains tens of thousands of items. Two scenarios are considered. The first design approach takes into consideration a big number of clients, and in the second approach, the number of concurrent clients is small but quick server response is expected. In both cases it is assumed that all clients are interested in all server items. It is also assumed that item values rarely change. Server response time, processor workload and memory usage are considered.*

1. INTRODUCTION

The OPC (OLE for Process Control) technology helps to open connectivity software for automation, IT and enterprise-wide management environments [1]. It provides a common way for applications to access data from any data source, e.g. a device on the factory floor or a database. The information architecture for the Process Industry involves field management, process management and business management levels. Providing information in a consistent manner to client applications, regardless of the management level, minimizes the effort required to provide the integration. So far, hundreds of plants and thousands of applications already depend on OPC solutions and some of them involve huge number of items.

OPC is based on Microsoft's OLE/COM technology [2]. Overall architecture utilizes DCOM technology (distributed COM) to facilitate clients interfacing to remote servers and to communicate the data to any client application in a standard way. To eliminate the need for custom interfaces between disparate computing solutions, the OPC interfaces are given in the specifications.

OPC Data Access (DA) is a specification that deals with online data access [3]. It is designed primarily to take snapshots of current real time process or automation data and to allow efficient reading and writing of data between an application and a data source like process control device. It specifies the behavior that the interfaces are expected to provide to the client applications that use them, but leaves the implementation of those interfaces to the programmer. Therefore, the design and implementation of such interfaces varies from one server implementation to another.

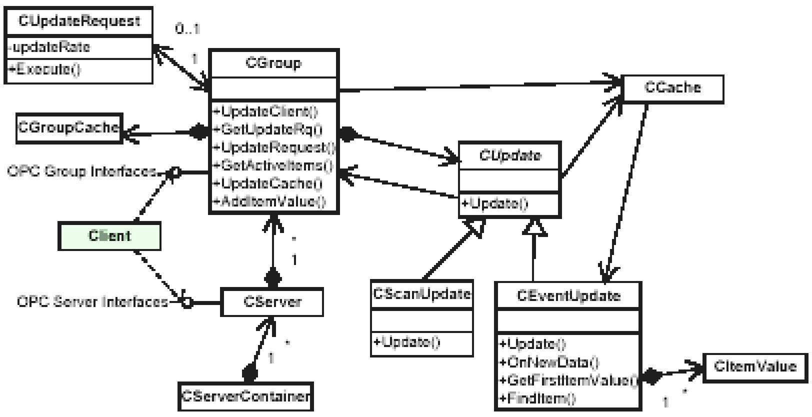
In practice, when OPC DA Server method calls are observed, except Read and Write methods, all the other server (and group) methods are called rarely. If we assume that the clients are subscribed to data updates, then a single call to a Write method, issued by any client, results in a number of update calls to all the clients in order to refresh their item values. The server acts in a similar way when new values arrive from a data source. This paper is focused on client update process inside the server; because it is a time-consuming process and it can significantly increase the CPU workload. Therefore, only a part of server design that concerns client updates is presented. UML notation is used and few class diagrams and sequence diagrams are given to clarify the design.

2. SERVER DESIGN

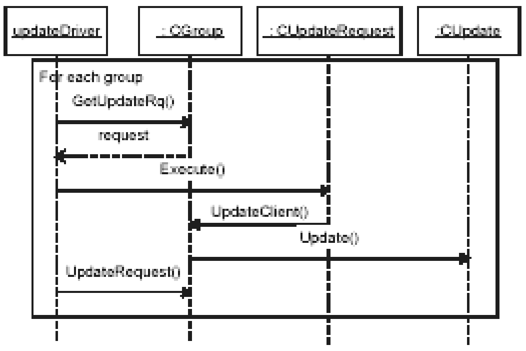
The OPC DA specification describes the COM objects and their interfaces that should be implemented in an OPC server. At a high level, an OPC DA Server comprises of several objects: the server, the group, and the item. Picture 1 shows the design where CGroup class encapsulates OPC Group and implements its interfaces; C Server class is introduced as a type of OPC Server object to implement OPC Server interfaces and to serve as a container for OPC group objects. From the custom interface perspective, an OPC Item is not accessible as an object by any client.

Therefore, there is no external interface defined for an OPC item. Every access to items is via an OPC Group object that contains references to item definitions. Within each CGroup the client can define one or more item references, which provides a way for clients to organize data. An instance of CGroupCache class is used by CGroup object to contain and logically organize such items. It also keeps copies of all item values that were sent to the client. Such copies are needed for the server to compare newly obtained values with the values which have already been sent to the client, to avoid unnecessary communication with the client.

In a typical application many clients concurrently communicate with the server and each of them expects the server to maintain its particular information. Therefore, an instance of CServer object is created every time a client connects to the server. All CServer objects are kept in a container CServerContainer. So far, the design presented here reflects the basic OPC architecture and it is expected to be found in almost any DA server.



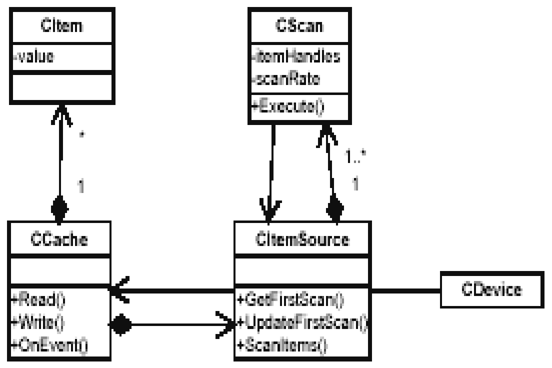
PICTURE 1. CLASS DIAGRAMM



PICTURE 2. SEQUENCE DIAGRAM - PERIODIC UPDATE OF OPC GROUP CACHE

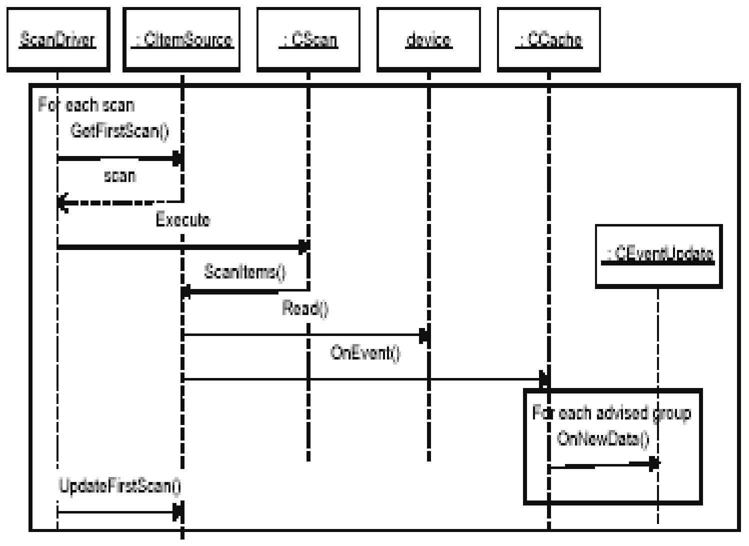
To improve performance when a number of clients is concurrently using the server, it is suggested that server implementation should read data into some sort of "cache", which should reflect the latest values of all items [3]. CCache object is introduced to contain and logically organize all items in the server and CGroup is aware of it. There are several ways for a client to obtain cached item data from the server. Client can perform either a synchronous or asynchronous read from the cache via CGroup (simple and reasonably efficient), but this might be appropriate for clients that are reading relatively small amounts of data and where maximum efficiency is not a concern, which is not observed in this paper. "Event driven" based connections can also be created between the client and the items in the group. This is a more efficient and complex way. It uses the callback mechanism when the client subscribes to changes in data. The clients which are considered are implemented to use the callback mechanism (IOPCDataCallback: :OnDataChange()).

In order to model polling or event driven CCache and CGroupCache updates inside of the server, abstract class CUpdate is introduced. It enables implementation of particular update techniques in derived classes: CScanUpdate and CEventUpdate. Later, two Update methods specialized by CScanUpdate and CEventUpdate classes will be described. CGroup class aggregates an instance of CUpdate object and each call to UpdateClient method is delegated to CUpdate::Update (Picture 2). An OPC client can configure the rate at which an OPC Group should provide data changes. CUpdateRequest object executes updates (requests) at given times. It is implemented as "concrete command" from the command pattern [4].



PICTURE 3. CLASS DIAGRAM - SERVER CACHE

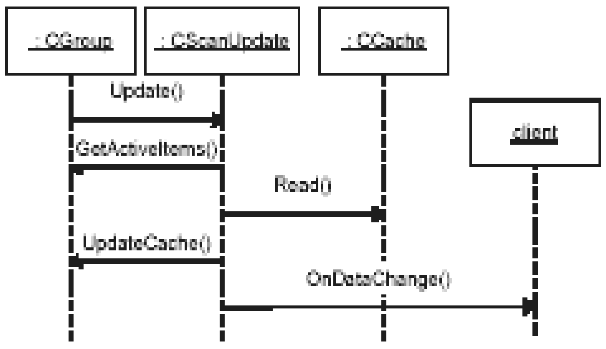
An important part of any DA server is server level cache, which should contain latest known values taken from data source(s). Server cache can be updated in different ways: by periodical scans of data sources, i.e. polling, or event driven method that receives events sent from data source. Picture 3 shows the class diagram of server cache. CItem class represents OPC Items at the server level, and among other attributes it contains the last known item value. CItemSource and CScan classes are used for periodical cache updates from data source, i.e. CDevice. Sequence diagram (Picture 4) shows periodical scans of data source and updates of server cache and clients (at the very end). When data is read from device, Ccache::OnEvent() method is called to update values in CItem objects. In case of an event driven solution, when device detects a new item value it could call OnEvent method directly (not shown in the picture).



PICTURE 4. SEQUENCE DIAGRAM OF PERIODIC SERVER CACHE UPDAT

3. SOLUTION A: CLIENT (OPC GROUP) CACHE UPDATES BASED ON PERIODICAL SCANS

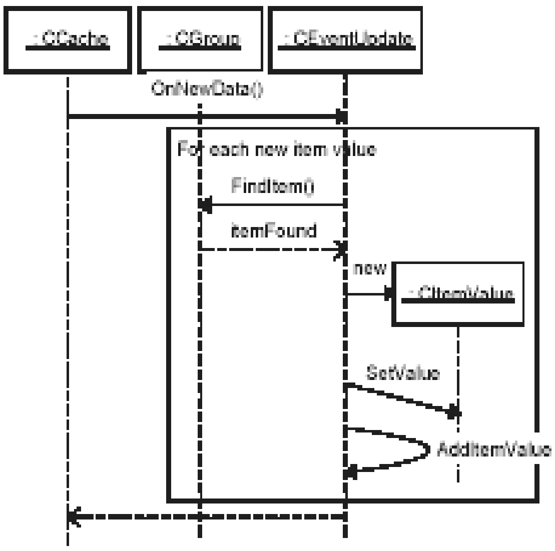
Item values in CGroupCache object can be updated by periodical comparison of all values in the group cache with values from the server cache. CScanUpdate object performs comparison in time instances determined by update rate attribute (CUpdateRequest: :updateRate). According to sequence diagram shown in Picture 5, values are compared in UpdateCache method and then group cache is updated and OnDataChange method is called to send new values to the client. Additional client parameter "percent deadband", prevents sending of values that are not changed over the given limits. This keeps the amount of data sent to the client small, when item values are changed rarely.



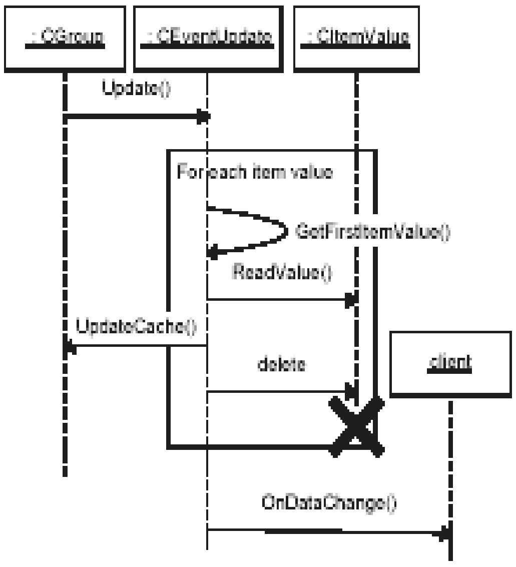
PICTURE 5. SEQUENCE DIAGRAM - PERIODICAL CLIENT CACHE UPDATE

4. SOLUTION B: CLIENT (OPC GROUP) CACHE UPDATES BASED ON EVENTS

When fresh item value is taken from data source, it is written into the server cache. If the cache object is aware of all OPC groups registered by clients, then new item value can be sent to all CGroup objects immediately. Comparison of new item values with old values which are stored in CGroupCache objects occurs, but sending changed values to remote clients may be a lengthy process, which involves calls to clients' callback functions. Therefore, a kind of buffer is introduced. A queue object is assigned to each CEventUpdate object and all new values which arrive, are put into such queues (values of items not registered in a group are ignored). Such behavior is shown in Picture 6. Later, values are taken from queue by a separate thread of execution; they are compared with values from CGroupCache object and are either sent to the client or are thrown away (Picture 7). Item values are updated in a loop, which is similar to previous solution where the loop iterates over all client items, but here the loop iterates only over queued items. This approach is more efficient when assumptions are fulfilled, i.e. when there are many items in groups and when item values are changed rarely.



PICTURE 6. SEQUENCE DIAGRAM - EVENT DRIVEN SOLUTION: INFORMATION ABOUT THE CHANGE OF A VALUE IS PUT IN QUEUE



PICTURE 7. SEQUENCE DIAGRAM - EVENT DRIVEN SOLUTION: CLIENT UPDATE

5. IMPLEMENTATION AND COMPARISON

Both solutions for updating group cache are implemented by the same OPC server. A configuration parameter exists which enables the user to choose the update technique. According to the user selection, CScanUpdate or CEventUpdate object is created when client adds a new OPC group. The server is implemented in C++ programming language as out-of-process ATL COM component that supports MTA COM threading model. Vector templates from STL class library were used for all cache implementations.

Solution A was designed and implemented first. From the very beginning it was assumed that server should run different kind of clients with rather small number of items in groups where update rates are measured in seconds (not milliseconds). According to such assumptions the number of concurrent client was expected to be even 100 or more. Unfortunately, small update rate on OPC group with tens of thousands items completely occupies CPU, which induced solution B for event driven group cache updates.

Performances of both solutions was tested in small 10 Mbit Ethernet computer network where the server was running on Windows 2000 1.4 GHz P4 computer and all clients were running on Windows 2000 1GHz P2 computers. All of them were equipped with 512 MB of RAM. The client program was written for testing purposes only and basically it measures server response time when OPC methods are called. When initiated by a user, the client writes 10 item values to the server (calling asynchronous Write method on IAsyncWrite interface), and the time of call is recorded.

The time when update call is completed is also recorded for each client. The difference between these time instances (the end of OnDataChange call and the beginning of Write call) is denoted as server response time. The server configuration had 42798 items. For all tests OPC group update rate was set to 100 [ms], which means that scan periods were set to 100 [ms] whether values were read directly from the server cache (solution A) or values were taken from the queue (solution B). Also, refreshing of OPC server cache from device was turned off during the testing.

When group cache was refreshed on event basis (solution B) and 4 clients were working concurrently, average measured response time was 42 [ms] and max was 78 [ms]. After initialization, server occupied about 45MB of RAM and for each new client additional 8.5MB was allocated. CPU usage was very low, only a few percent. When cache refreshing was turned on with cycle of 500 [ms], CPU usage raised for about 10-15%.

When group cache was refreshed on periodic scan basis (solution A), CPU usage was significantly higher. In situation when 4 clients were working concurrently 100% CPU usage was encountered almost all the time. Average measured response time was 344 [ms] and max was 1128 [ms]. With 3 clients CPU usage was about 97% on server and average measured response time was 204 [ms] and max value was 440 [ms].

6. CONCLUSION

Presented design and implementation of OPC DA Server has no limitations on the number of items, number of OPC groups and connected clients, but in practice CPU usage limits these numbers. Solution A clearly divides server cache updates and group updates, which keeps design simple. On the other hand, comparison is done for every single item in all OPC groups, which makes CPU more engaged. This solution is possible only when the total number of OPC groups is small or/and number of items in each group is small. The design of solution B, which is based on events, is more complex, but presented results show solution B as a much better choice. When item values are changed often and number of OPC groups is big, item source scan could be delayed and memory consumptions for queues could be significant, too. This solution should be extremely useful for applications where clients are interested in all server items and item values are changed slowly. In practice, such architectures can be seen in distributed management application for electric power systems. Even though the presented results prefer client cache updates based on events, both solution should be taken into consideration when new OPC server application is being planned.

This work has been financed by Ministry of Science and Tehnology gant IT. 1.22.3212.A.

REFERENCES

[1.] OPC Foundation, "OPC Overview, Version 1.0", 1998.
[2.] Don Box, "Essential COM", Addison-Wesley, 1998,
[3.] OPC Foundation, "The OPC Data Access Custom Specification, Version 2.04", 2000.
[4.] Erich Gama, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns - Elements of Reusable Object-Oriented Software", Addison-Wesley Publishing Company, Inc., 1995.