USING OPC DATA EXCHANGE IN SIMULATION ASSISTED AUTOMATION TESTING

Jyrki Peltoniemi, Matti Paljakka, Tommi Karhela
VTT, Technical Research Centre of Finland
P.O.Box 1301, FIN-02044 VTT
email: Jyrki.Peltoniemi@vtt.fi

KEYWORDS
Process simulation, automation testing, OPC data exchange, software design, performance

ABSTRACT

Dynamic process simulation models can be used for testing automation - both control and logic - before the commissioning. This activity requires a flexible, fast and robust connection between automation software and process simulation engines.

OPC Data eXchange (DX) specification provides an open and standardized means for configuring connections and exchanging data between various kinds of automation components, e.g. dynamic process simulators and automation software.

This paper first presents an analysis of the DX specification from the perspective of large-scale simulation use. Then, a DX server design is introduced that provides high performance, without compromising component reuse and portability. Then, the throughput of a prototype DX server is evaluated. A performance test is carried out to demonstrate the applicability of DX-based communication in simulation assisted automation testing purposes.

INTRODUCTION

Process simulation can be used in various phases of an automation delivery project e.g. to verify process and automation design in specification phase and to validate automation implementation in factory acceptance tests (FAT) at the end of the implementation phase. By using simulation in FAT, one can rehearse the commissioning of the system and to catch the flaws in the application that would normally be caught on the site. This leads to shorter commissioning times and better quality.

In simulation assisted FAT, the automation application reads the measurement values from a simulation model and writes the control values to the model. Typically, the automation application and the simulation model run in separate systems. The connections that are configured between these two systems form the basis for the low-level communication, required to exchange data during the testing procedure.

For obvious reasons, there is a need for a vendor-independent standard for both configuring and executing data exchange in the testing environment. If open data exchange specifications are used, applications that have been implemented on any platform that conforms to the standard can be tested in the same environment. Similarly, simulation tools by different providers can be connected to the same environment, which may be desirable in case the model comprises submodels that represent different domains, or in case different parts or the model are provided by different companies. Open interfaces to configure connections and exchange data between the applications in this kind of testing environment is hence desirable from several perspectives.

Open interfaces, however, are not enough for successful simulation aided testing if the implementations cannot provide reasonable performance. The main objective of this paper is to introduce a design that can provide large throughput without compromising component reuse or portability. This design and the suitability of the DX specification are then evaluated using simple performance tests that are carried out using a prototype implementation.

Figure 1 presents an example on the co-use of multiple automation and simulation products. The distributed simulation system is controlled by a Simulation Manager application, through which one can define data connections and control simulation. For automation testing use, the system may naturally also include a database for test case definitions and software for the analysis of test run data.

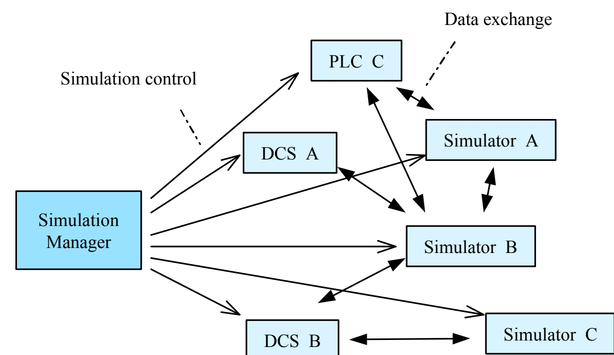

Figure 1: an Example System for Simulation Assisted Testing

DATA EXCHANGE SPECIFICATION

The aim of the OPC Foundation is to promote open connectivity in industrial automation. The Foundation has developed and released a number of interface specifications for the exchange of data, alarms and events etc., and as these specifications have been implemented in most automation products, OPC has become a popular way to integrate data across the enterprise.

To standardize the horizontal data exchange between automation components, OPC Foundation has released a new specification, Data eXchange (DX) (OPC Foundation 2003a). Earlier, special OPC clients had to marshal the data between two OPC data-access (DA) servers (Laakso 2003; Karhela 2002). The setup had basically two problems. Firstly, each client had a product-specific way to configure the connections, and the connections made with one product could not be re-used with another one. Secondly, due to the client, the communication architecture was unnecessarily complicated and far from optimal in performance. The new specification in principle solves both problems as it removes the need for a special client.

The Data eXchange specification defines how to make connections between a number of DX and DA servers. Configuring a bi-directional connection between two servers requires that both participants conform to the DX interface specification. Information about connections is as well distributed as each server has a database of its own. The runtime communication is based on the earlier DA specification or the newer XML-DA specification.

A DX server mainly consists of an database for connections, a COM (Component Object Model) or WSDL (Web Services Description Language) interface for updating the database, source access components for DA or XML-DA servers, runtime target item components and support for monitoring and controlling the existing connections. These concepts are quite clearly covered in the DX specification.

A DX database consists of DX connections that can be arranged under DX branches. A DX connection is composed of a target item id, a source item id, a reference to the source server and a number of other attributes. DX clients can modify each connection separately or affect on the behavior of several connections by modifying attributes of both branches and source servers.

The structure of the DX database is quite complicated. All servers have to support e.g. vectors of strings and

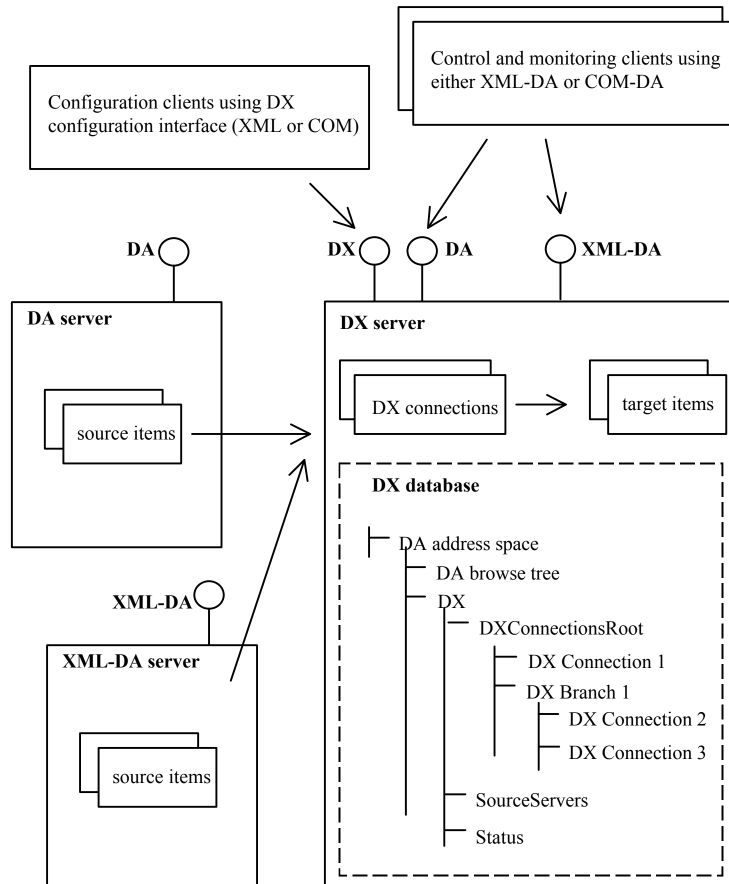

Figure 2: Architecture of DX System (OPC Foundation 2003a)

branches that can simultaneously be items. The database also includes some redundant information, as some structural data can be accessed through the composed string items or through individual simple items. Also the fact that connections may have several parent-branches may be problematic. One feature that clearly makes the server-side implementation rather complex is that there are essentially two distinct methods to affect on the run-time behavior of connections. Using the configuration interface is an obvious way, but controlling the run-time behavior can also be made by writing to some special items that exists in the DX database. Both methods are available either through the COM or the WSDL interface. The run time activity can thus be configured using also plain data access clients. Figure 2 illustrates the DX architecture.

One important requirement from the server is that the operations that use the services of source servers have to be asynchronous. The operations first update the database, and then return the control back to the client. After that, the DX-server asynchronically makes the required operations by calling appropriate source servers. Depending on the status of the source server, the results of the operation may be reflected to one or more items in the database of the corresponding DX server. The client that made the original operation may see this if it is currently monitoring proper items.

From the server-side design perspective the specification is quite complicated, although the configuration interface is quite simple.

A compliant DX server provides a rich set of operations to clients that establish and monitor connections. There can be several clients simultaneously making and monitoring connections. The connections are not client-specific i.e. it is not relevant which client has made the connection. Furthermore all clients may see the whole database. The clients use the standard DA and XML-DA interfaces for monitoring purposes. There is a wide set of operations that clients can make and the effects of the operations greatly depend on the structure and complexity of the database on the server side. The idea is that by creating a meaningful structure in the connection database, clients can conveniently and effectively observe and control data exchange. However, a client has no means for preventing other clients from changing any parts of the database.

The DX specification provides a sufficient set of operations to configure data exchange for co-use of automation and simulation software. The extra complexity originates of the support for two different interface technologies, COM-IDL and WSDL.

Simulation control and synchronization interfaces, that are needed to administrate non-real time simulators, are still lacking standardized approach.

DX SERVER DESIGN

Adding the features that allow DA servers to act like OPC clients, needs not to be a complicated task. Also the interface that allows making such connections can be very simple. This task became more complicated mainly because the DX working group under the OPC Foundation wanted that connections could also be configured using a web-services (WSDL) interface. The XML-DA interface (OPC Foundation 2003b) is also provided for data exchange purposes. Some features and extra complexity also reflects the fact that DX specification extends the concepts already defined in the earlier DA specification.

Requirements for Design

Before the DX server design is introduced, the basic requirements for the DX server are listed. An obvious requirement is to implement specification as precisely as possible. For the interoperability reasons, also the optional parts should be implemented whenever it is possible. The second important requirement is to be able to make such server deployments that allow porting web-services part for non-Windows platforms also, particularly to Linux. This is probably the most restrictive requirement. Old COM-based DA components that have already been developed and tested should be able to be re-used as effectively as possible. The performance of the COM-based data exchange should be sufficient for large-scale use. The performance should not suffer considerably e.g. because of the support for connection monitoring or because of the persistence requirements for the DX database.

The implementation should also provide an opportunity to configure and monitor connections simultaneously using the web-services interface and the COM interface. This kind of functionality is, of course, required only on Windows platform.

Essentially, there are three major requirements, that design of DX server should reflect: portability, component reuse and performance of data exchange.

Overview of the Component Design

Figure 3 illustrates the component view of design under study. It consists of seven components, which build up one executable. The *simulation engine* uses the data-access framework interface (*framework*) to link components that provides standardizes means for external connections for the simulation model.

The *COMKit* exposes standard DA-interfaces and DX-configuration interface implemented as a COM-interface. The *SOAPKit* handles SOAP requests and exposes both the OPC XML-DA interface and the DX configuration interface declared using WSDL.

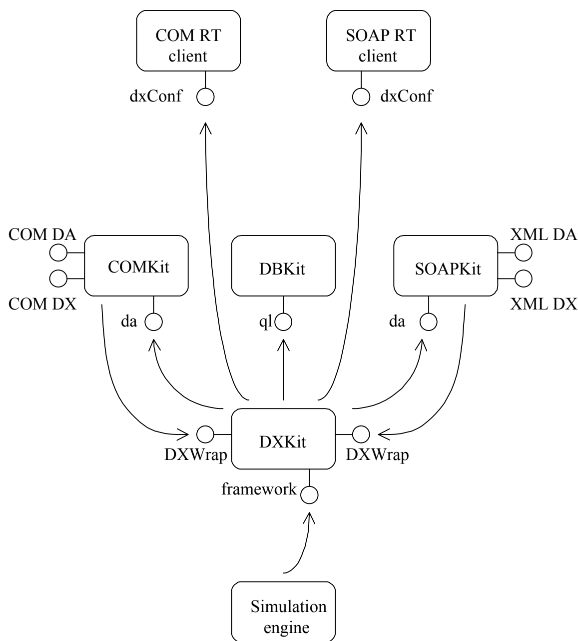

Figure 3: Component View of DX Server Design

The DXKit (*DXKit*) contains major parts of the DX server functionality and marshals data to and from the simulation engine using the framework interface. The COMKit and the SOAPKit marshal DX configuration requests to the DXKit by using a C++ interface *DXWrap*. This internal interface is a one-to-one mapping of the standard OPC DX interface. The DXKit uses a Database Kit (*DBKit*) to build up the persistent database for DX connections. The DBKit provides simple XML-based query language (*ql*) interface to access database entries that consist of DX connections and various simulation models.

The COM run time client (*COM RT client*) is responsible for subscribing data from appropriate source servers if source items are located in a COM DA server. Similarly, the SOAP runtime client (*SOAP RT client*) builds up the appropriate SOAP requests if source items are found on an XML-DA aware source server. The DXKit uses the C++ interface *dxConf* to command these two components. The dxConf interface is also a bi-directional interface. These two runtime client components notify the DXKit whenever they get new data from source servers.

Core DX Functionality

After a brief overview on the component design, a closer discussion about functionality and interactions between these components is examined.

The most central and most multifunctional component in the DX-server design is the DXKit. It is a portable component and one of its responsibilities is to marshal data to underlying simulation engine or to the DBKit. The DXKit hides the actual location of the data from the COMKit and the SOAPKit. There are essentially three different types of items that can be monitored either through the COM DA or the XML-DA interfaces. The first and most evident type of items is those that are currently loaded to the solver of the underlying simulation engine. The second type of items is those that are not currently loaded to the simulation engine but rather exist in the persistent database of simulator, i.e. in the DBKit. The DBKit can contain several models that can be simulated. These currently not simulated items should also be observable and connectable using standardized techniques. Thirdly, the DX specification defines that every DX database has a similar kind of a structure and e.g. each DX connection can be observed also using standard data-access interfaces. The DXKit hides these details from the COMKit and the SOAPKit using the da interface (C++ data access interface) that provides a means for browsing and transferring data. This simplifies the structure of the COMKit and the SOAPKit. It also improves the reusability of the DA server implementation available in the COMKit.

For all active DX connections the DXKit keeps up volatile run time objects as well. A particularly interesting special case is the status information that each connection has. This status information consists of e.g. quality information, timestamps and source item value. The connection status information changes constantly during the run time data exchange. Persisting these values during run time would definitely compromise performance. Depending on the status of the DX connection some data items that the connection has created can be found either in the database (DBKit) or in the volatile memory. This kind of behavior is also hidden from the COMKit and the SOAPKit. Neither component can see the actual location of any individual data access item.

The DXKit hides the actual location of data from monitoring and controlling clients, and it also hides it from the COM RT client and SOAP RT client components. The target item of each active connection may exist either in the solver of simulation engine or in the persistent database (DBKit). Typically the target data of the active connections can be located in the solver of underlying simulation engine, rather than in the persistent database.

Although the DX configuration interface does not contain many operations, the overall result of each operation heavily depends on the structure of the DX connection database. A single operation made by using either the DX configuration interface or through the control items may affect the status of several connections. A good example of this kind of behavior is a case, where a configuration client modifies the attributes of some DX branch that has several child connections. The source items of connections may exist in several separate source servers. Furthermore, each of these source servers may be either COM-DA or XML-

DA servers. The DXKit component resolves this kind of dependencies and commands either the COM RT client or the SOAP RT client component using the simple C++ interface dxConf. Through the dxConf interface the DXKit can create new connections, remove connections and modify the status of each connection. However, the dxConf interface is much simpler than the standard DX configuration interface. Ideally RT components should be as lightweight as possible, and their sole purpose is to get data and to marshal data from source servers to the DXKit. Whoever implements these run time client components, does not need to be aware of the constructs defined in the DX specification.

It is also required that DX connections can be modified and controlled using SOAP and COM clients simultaneously. Centralizing all intelligent functionality in the single component is the easiest way to fulfill this challenge. Otherwise there may arise troublesome inconsistencies and synchronization problems. In this kind of a design all decisions that can affect the run time behavior are made in the same portable component irrespective of the interface type that the configuration and monitoring clients are using.

Finally, the most important justification to centralize the main functionality to the DXKit is to avoid coding similar functionality twice. Clearly, if only e.g. a COM-based DX implementation is needed, tighter integration of a DA server, a DX configuration component and a run time client part, would result in a more compact realization. Similarly, if platform independence of the SOAP-based solution is not an issue, different kind of solutions may be reasonable and more effective.

Because of the challenging requirements, the overall design consists of rather many components. As interfaces between these components are basically bi-directional, the overall structure and the control paths during the operation take quite a complicated form.

Runtime Data Flow during Data Exchange

The data flow from source items to targets can begin after the necessary data structures have been created to the DXKit as well as to either of the run time components, and to the persistent database of the DBKit. The appropriate run time client component marshals data to the DXKit. The DXKit marshals the data to the appropriate location, typically to the simulation engine. Neither the COMKit nor the SOAPKit participates in data exchange.

In addition to marshalling data, the DXKit is responsible for updating necessary status items that are associated with each DX connection. These status items can be used to observe the current state of the connections and the data flow. Although the status items are contained in the DX database, the status items cannot be persisted during data exchange. Doing such persisting operations at run time would drastically lower the data exchange performance. The RT clients marshal the connection status information and make appropriate processing only if the DXKit requires that. This behavior is essential to optimize the throughput for the most demanding data exchange needs.

PERFORMANCE METRICS

Throughput is the most critical performance quantity, when large automation applications are connected with process simulators. An obvious test case consists of two DX servers that are connected symmetrically using COM-based communication. Similar tests have earlier been done for DA-based communication. (Peltoniemi 2001)

A PC with a 1.2 GHz processor and 512-MB RAM was used to carry out the test case. Both DX servers were located on the same computer using Windows 2000 operating system. Creating equal numbers of DX connections in both DX servers created a bi-directional connection between two simulators. Event based (DA2) data exchange was used to retrieve data from the source server. All double precision source items in both servers were continuously changing and the update rate that was used during data exchange was 200ms (Figure 4).

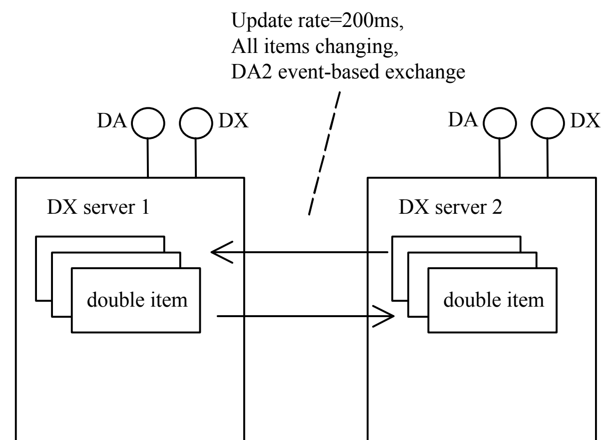

Figure 4: Arrangement of Test Case

None of the connections or items was observed during data exchange. If monitoring clients are simultaneously observing plenty of connection status data, this significantly affects the performance, depending on the needed data type conversions and other properties of item set under monitoring. A particularly heavy load may be generated if plenty of complex DX connection items or connection status items are observed simultaneously. Hence, when performance aspects are critical, also the behavior of monitoring clients is important.

As discussed preceding study (Peltoniemi 2001), it is expected that the throughput depends linearly on the number of connections. This seems to be a valid assumption also in this case, see Figure 5.

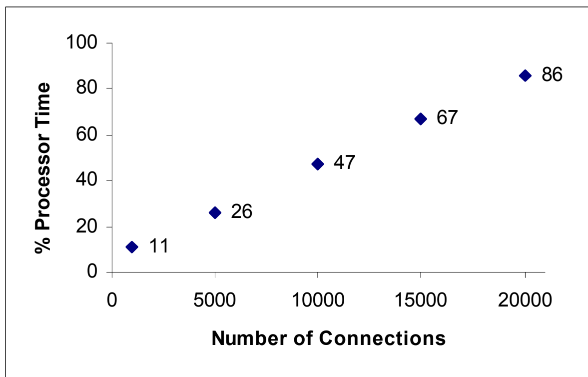

Figure 5: Total Processor Time Using Different Number of Connections

The throughput is significantly better compared to the performance that was achieved in earlier tests, where a separate cross-connector client application marshaled data between two OPC DA servers. This earlier test case was a little bit different, as connections were created for one direction only. The processor load was 81% when 11000 items were transferred from one server to another. The throughput of the DX-based communication is over three times better than the throughput that was achieved using a separate client application to marshal data.

If extremely large-scale models have to be connected with external applications using e.g. Linux platform, the web-services-based communication may not provide a reasonable performance. An insufficient throughput may be a problem for the COM-based communication as well. Proprietary communication could be done e.g. by using an optimized socket-based solution. In that case the configuration of connections could still be made through the standard DX interfaces, only data exchange being done using run time socket components.

CONCLUSIONS AND FUTURE PROSPECTS

The discussion above concentrated those aspects that are relevant when the components based on the OPC Data eXchange specification are used in simulation assisted automation system testing. A detailed design was illustrated and the design philosophy was justified. A design that provides support for the entire OPC DX specification, including both SOAP-binding and COM-binding was introduced. Component-based design allows reasonable throughput without compromising portability of SOAP-based solution and reusability of existing COM DA server components.

The performance of the data exchange was studied to have a better understanding about the suitability of the DX-based communication in large-scale simulation aided automation testing purposes. The performance of COM-based data exchange should be reasonably good if both server components are designed in a manner that allows a large throughput.

COM-based data exchange forms the basis for the data exchange between automation system and process simulator in the foreseeable future. However, SOAP-based data exchange defined in the OPC XML-DA specification may become a more attractive choice in forthcoming years. The suitability of XML-DA -based communication for large-scale use will be studied using a similar arrangement.

The ability to effectively and flexibly exchange data between automation software and process simulation models is a fundamental requirement for successful simulation aided automation testing. Using OPC Data eXchange specification, simulation systems can be built that meet this requirement. However, solving the purely information technology related challenges is only the first step in the take-up of simulation in automation testing. In addition, defining proper working methods for simulation aided automation testing and building tools to support the new working methods are essential research challenges in near future.

REFERENCES

Karhela, Tommi. A Software Architecture for Configuration and Usage of Process Simulation Models. Software Component Technology and XML-based Approach. Espoo, 2002, Technical Research Centre of Finland, VTT Publications 479, 129p.

Laakso, Pasi; Peltoniemi, Jyrki; Karhela, Tommi; Paljakka, Matti. The use of OPC in Simulation Systems – experiences and future prospects. The Proceedings of the 3rd International Symposium on Open Control Systems 2003, Helsinki, September 9-10 2003.

OPC Foundation. OPC Data eXchange Specification, 1.0, March 5, 2003.

OPC Foundation. OPC XML-DA Specification, 1.0, July 12, 2003.

Peltoniemi, Jyrki; Karhela, Tommi; Paljakka, Matti. Performance Evaluation of OPC-based I/O of a Dynamic Process Simulator. The Proceedings of the 2001 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Orlando, Florida, July 15-19 2001, p. 231-236, SCS, ISBN: 1-56555-240-7.