OPENPROD

ITEA 2
INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

Report

Tuomas Miettinen 9.6.2011

OPC Interfaces in OpenModelica – Technical Specification (Task 5.3)

Version 2011-06-09

.

TABLE OF CONTENTS

Abbreviations

AAA	Authentication, authorization, and accounting
Adda	Advanced data access
COM	Component Object Model
DA	Data Access
DCOM	Distributed Component Object Model
DCS	Distributed control system
ERP	Enterprise resource planning
HMI	Human machine interface
I/O	Input/output
ITEA	Information Technology for European Advancement
MES	Manufacturing execution system
OLE	Object Linking and Embedding
OMI	OpenModelica Interactive
OPC	Open connectivity via open standards (formerly OLE for Process Control)
OPC UA	OPC Unified Architecture
PLC	Programmable logic controller
SCADA	Supervisory control and data acquisition
SDK	Software development kit
SOA	Service-oriented architecture
UA	OPC Unified Architecture
WS	Web services
XML	Extensible Markup Language

Summary

Modelling and simulation tools are needed as a key component in both designing and controlling modern technical systems. To connect a simulation tool to hardware devices within such a system, commonly used protocols need to be used. Incorporating OPC communication interfaces into the OpenModelica simulation environment will enable OpenModelica to be utilized within a wide variety of technical systems, including process control and manufacturing automation systems in particular.

OPC is an established interface specification for accessing field devices within control and automation systems; it has become a de facto standard throughout the industry. The newest of OPC specifications, OPC Unified Architecture, was developed to improve OPC and to unify all of the functionalities of OPC under one interface. Its intended purpose is to substitute the regular OPC, as well as some other communication solutions, in new but also in already existing control and automation systems. Moreover, OPC UA is designed to be utilized in a broader domain of technical systems.

The main goal of this project is to implement OPC interfaces, including especially the new OPC UA interface, to OpenModelica and thus allow OpenModelica to be used with already existing systems as well as systems of tomorrow. In addition, as a minor goal of the project, the OPC UA interface is incorporated into Apros modelling and simulation environment by reusing parts of the effort.

In this report, the basics of the subjects of the project are presented: the Modelica modelling language and the OpenModelica environment are gone through in general terms whereas the OPC and OPC UA specifications are discussed in depth in order to explain their function within control systems. Technical details of the subjects are discussed to the extent which is necessary in deciding between the main architectural approaches available for the implementation.

Two different architectural approaches for the implementation are introduced. The first solution is to integrate the OPC UA interface to the simulation executable to enable maximum computational performance for the connection. The other solution is to let the OPC UA server operate independently from the simulation, thus allowing the simulation model to be altered without disconnecting external applications being connected to the OPC UA interface. The former solution is chosen to be implemented first and is therefore examined more closely. However, it is suggested that the latter solution can be implemented once the initial implementation is completed.

1. Introduction

OpenProd [1] is an ITEA 2 European project [2] which aims to provide a holistic whole-product model-driven rapid development and design environment for both software and hardware. The goal is achieved by using open source tools and components with standardized model representation of products. The project described in this report is carried out as a part of the OpenProd project Work Package 5 **–** Interoperability, the goals of which are to allow interoperability between tools and better reuse of tool components and to enable modelling and simulation with model components in different formalisms.

Contemporary technical systems have become large, complex, and mathematically difficult. When it comes to building and controlling such systems, the conventional tools are becoming obsolete. In the meantime, the exponentially growing computational speed has paved the way for modelling and simulation tools development. Compared to performing experiments on real systems, modelling is cost-effective, fast, and safe, and to control and to modify the system is a lot easier with models than with real-world systems. Modelling has thus become an essential tool in constructing such systems.

Within technical systems there is a need of communication between different parts of the systems. With modern systems, the amount and complexity of data that needs to be transferred between those parts is increasing all the time. To handle the situation, well-defined procedures must be utilized.

OpenModelica is a simulation environment used to build and interpret models written in Modelica, an equation-based modelling and simulation language. OPC and its latest version OPC UA are specifications that can be used in communication among both software and hardware components in technical systems, especially in process control and manufacturing automation systems. In this report a method for incorporating the OPC interfaces, especially OPC UA, into OpenModelica is proposed.

In the remaining of this chapter the main subjects of this report are discussed on a higher plane. Their backgrounds and characteristics are explained and some examples of their applications are presented. In addition, a motivation for the research and development theme is given.

The subsequent chapters of this report consist of a more detailed description of the problem: Chapter 2 explains what is to be achieved in general. In Chapters 3 through 5 the technical details of the implementation issues are considered: the interfaces and the boundaries set by them are discussed, the different architectures are weighed up, one of them is chosen for the implementation and inspected more closely as well as the architectural choices made are justified. Chapter 6 concludes the report and its results.

1.1. Modelica and OpenModelica

1.1.1. The Modelica Language

Modelica is an open standard modelling and simulation language developed in an international effort started in 1996. The Modelica Association, an international non-profit organization, has been developing the open standard since then. [3]

The Modelica language is intended to be used in modelling the dynamic behaviour of technical systems which consist of components from different domains. It can be used especially to model large, complex, and heterogeneous systems. It is an object-oriented high level language which can be used with systems that need high computational performance. [4] The Modelica language has three key differences to most other simulation languages. These features are as follows.

Firstly, in typical programming, modelling, and simulation languages, the functionality of a program is described with assignment sentences. When talking about physical equations, information is lost with this kind of an approach. Modelica, however, is a declarative language using equations instead. The equations can be algebraic, differential, or discrete. To use equations means that real-world physical objects can be modelled as such in the language; the modeller does not have to consider in which way the equations are used which would have to be done when using languages allowing mere assignment. The generalization of the equations yields both simpler models and more efficient simulation. [4][5]

Secondly, most modelling languages are good at only a few technology domains. Modelica, however, can be used to model systems of different kinds. Systems such as electrical, mechanical, thermodynamic, hydraulic, biological, control, event, real-time, etc. can be modelled and connected to each other to construct hybrid models. Moreover, Modelica is well suited for both low and high level numerical algorithms [6]. [4]

Thirdly, Modelica is an object-oriented language with a general class concept. Added with the equation-based approach it allows creating physically relevant and easy-to-use model components which are employed to support hierarchical structuring, better reusability and interoperability of ready-made model blocks. In other words, the class concept facilitates reusing and exchanging models and model libraries. [4]

1.1.2. The OpenModelica Environment

OpenModelica is an open-source environment, the purpose of which is to provide tools to build, compile, and simulate models made by using the Modelica language. It is intended to respond to both industrial and academic demands. The development and promotion of OpenModelica is supported by the non-profit organization Open Source Modelica Consortium. [7] [8]

The OpenModelica system has both short and long-term goals. The short term goals include developing an efficient interactive computational environment for the Modelica language and a rather complete implementation of the language. The long-term goal is to have a complete reference implementation of the Modelica language, including simulation of equation based models and additional facilities in the programming environment, as well as convenient facilities for research and experimentation in language design or other research activities. However, to achieve the performance and quality of the commercial products is not an objective. [6]

The OpenModelica environment consists of several subsystems, as is shown in Figure 1. Currently, these subsystems are the Interactive Session Handler, The Modelica compiler, the Execution and run-time module, the various model editors, and the Modelica debugger. [6] The most important functionalities of these systems are as follows.

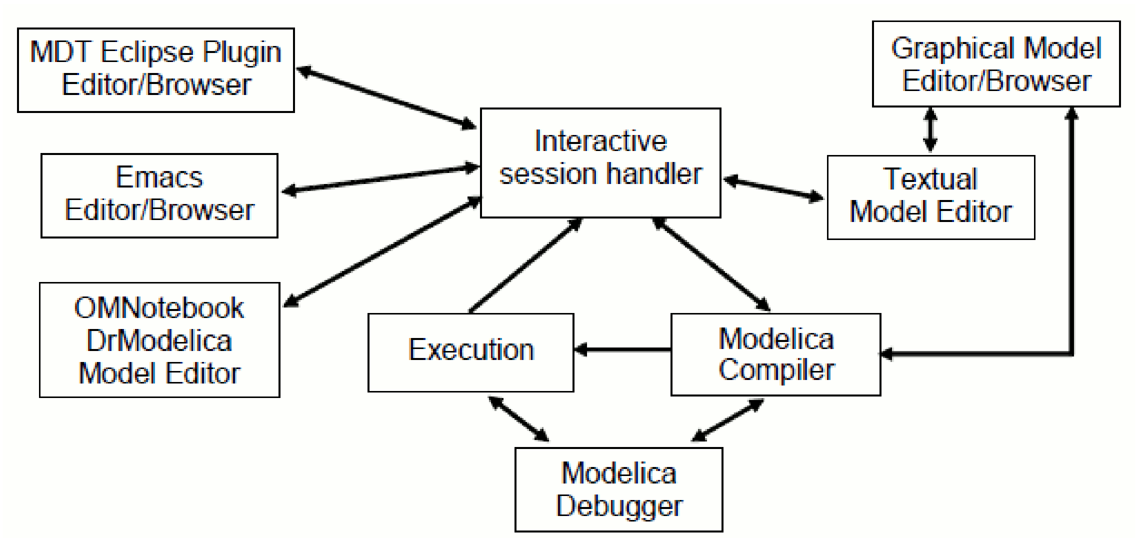
MDT Eclipse Plugin Editor/Browser
Graphical Model Editor/Browser
Interactive session handler
Emacs Editor/Browser
Textual Model Editor
OMNotebook DrModelica Model Editor
Execution
Modelica Compiler
Modelica Debugger

Figure 1. The architecture of the OpenModelica environment [6]

The interactive session handler parses and interprets commands for evaluation, simulation etc. *The Modelica compiler* translates Modelica expressions to C code. In addition, it includes a Modelica interpreter for interactive usage and constant expression evaluation. *The execution and run-time module* executes compiled binary code from translated expressions and functions as well as simulation code from equation based models. The set of *model editors*, besides providing editing and browsing capability of the Modelica models, have several features that make building models easier than with a general purpose text editor. *The Modelica debugger* is still in a somewhat immature state providing debugging ability for only certain types of models. [6]

OpenModelica can be utilized as such to build and simulate Modelica models. In addition, since being free software, OpenModelica or parts of it can be integrated to existing systems as plugins or developed further to fit better to the target system [6]. For instance, in the Simantics software platform this sort of a plugin approach is utilized. The Simantics platform will be further described later on in this document.

1.2. OPC

1.2.1. Idea of OPC

Until the early 1990s, when a process control hardware was to be connected with a software application, a driver for that particular piece of hardware had to be written and attached to the software application. Then, if the hardware component was to be replaced by a differing component, the driver had to be rewritten. On the other hand, if different software application wished to communicate with the piece of hardware, corresponding drivers had to be written for each of the applications. OPC was a solution presented to this problem. [9]

OPC is a standards specification which defines a common interface for communication between software packages and hardware devices of different kinds. When communicating through the OPC interface, there is no longer need to create a driver for each application–device pair. All that needs to be done is to create only one driver, which is compatible with the OPC interface, for each software and hardware component. In the following of this section, practical aspects of OPC are discussed and analyzed in a more detailed fashion as well as the history of OPC is presented.

1.2.2. History

OPC is an open specification being introduced in 1996 by an industrial automation industry task force. After the initial release, the OPC Foundation was created to continue the development of the set of specifications [10]. The first version of OPC consisted only of what is now called the Data Access Specification, hereinafter abbreviated as DA. Since then the number of standards specifications completed or in development has increased to nine.

The term OPC was originally an acronym for *OLE for Process Control*, where OLE stands for *Object Linking and Embedding*. However, since OPC is nowadays used widely not only in process industry but also in discrete manufacturing and due to the fact that the current UA specifications are platform independent, the Foundation now calls it *open connectivity via open standards*.

The initial purpose of OPC was to allow process control hardware to be used with Windows based applications. The OPC technology was build upon the OLE COM (*Component Object Model*) and DCOM (*Distributed Component Object Model*) technologies developed by Microsoft. These technologies specify interfaces that can be used in passing objects between processes which can be implemented in various programming languages and may be either located on the same computer or communicating over a network connection.

1.2.3. Basis and Uses

OPC was created to specify standard interfaces for connection between factory floor devices and monitoring and control software applications in the domain of process control and manufacturing automation systems. From the beginning, it was meant to be used to transfer real-time data between control devices, such as PLCs (*programmable logic controller*) and DCSs (*distributed control system*), and display clients, such as SCADAs (*supervisory control and data acquisition*) and HMIs (*human machine interface*). Later the set of specifications was extended to cover the passing of other types of data as well. The specifications define a standard set of objects, interfaces and methods which enable vendor-independent interoperability between software and hardware [9]. The specification has no restrictions concerning either the type or the source of the data. OPC has become the de facto standard for industrial integration and process information sharing [11]. [12]

The OPC technology is based on a server–client architecture. The intended purpose of use of OPC is that the software application acts as a client communicating with a separate server application which in turn is coupled with the hardware device. The client sends requests to the server which in turn processes the request and sends corresponding response back to the client. These requests can be, for example, reading values or sending commands. [12]

As mentioned earlier, traditionally, when a control hardware device was wanted to be connected to a software application, a special purpose driver had to be written. Using OPC makes the situation a lot simpler: Since there are ready-made server applications compatible with most of commercial hardware devices on the market, all that needs to be done is to configure one of such applications to communicate with the physical device. After that the software application can communicate with the hardware device by acting as an OPC client.

Figure 2 depicts the situation without OPC. Each of the arrows represents a unique driver written solely for the purpose of connecting a certain application to a certain piece of hardware. Even with a system this small the solution becomes tangled. In a scope of even a medium-sized control system with some complexity in its hardware devices it becomes obvious that to manage the communication within the system becomes rather impossible.

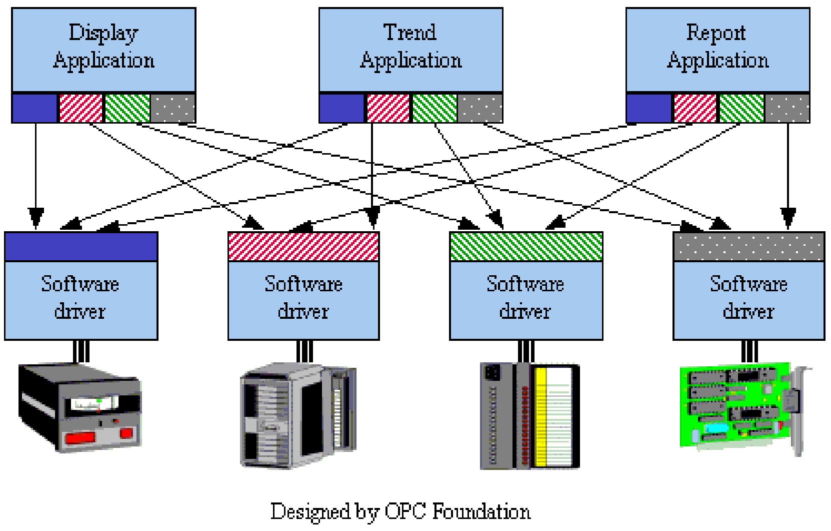

Figure 2. Software applications connecting hardware devices with special purpose drivers [13]

Figure 3 shows the same system with OPC servers connected to the hardware devices and OPC clients to the software applications. The number of pieces of software that has to be written decreases by five since each part of the system needs to be connected to only one interface. In addition, since it is not unusual to be able to acquire ready-made server application for the hardware devices, the only thing the developer needs to do is to implement OPC client behaviour for each application. With the OPC servers provided by external vendors, the total amount of work is decreased to implementing the three OPC clients and configuring the four OPC servers.

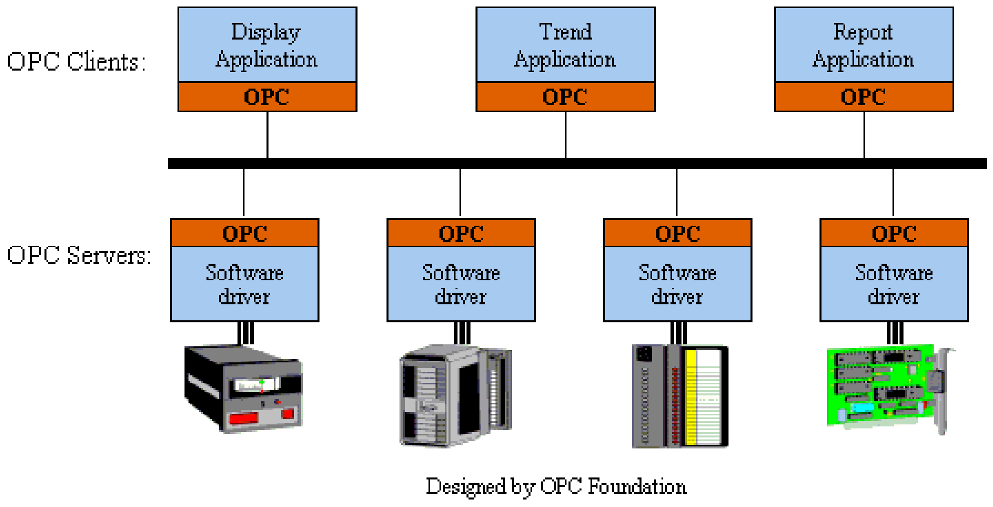

Figure 3. Software applications connecting hardware devices by using OPC [13]

In real-world systems with hundreds of pieces of software and hardware, other issues arise as well: Firstly, the replacing of a single piece of either software or hardware becomes rather impossible. Secondly, it is likely that two or more applications want to access the resources of a same hardware device simultaneously. OPC provides a solution for both of these problems: To replace a part of the system by a corresponding part, even made by another manufacturer, is a trivial procedure. With an OPC server being fully in charge of controlling a device, no synchronization issues between clients will emerge; the server can simultaneously serve requests of many clients using the device.

An example of a real-world application using OPC could be as follows is depicted in Figure 4: A PLC controlling an assembly line is configured by using external tools. It is likely that there are commercial OPC servers available for the PLC device used. Hence, to gain real-time control over the system, a commercial SCADA or HMI system with an OPC client interface is acquired. After interconnecting the systems, all that needs to be done to pass information from the process to the SCADA system is to configure the OPC interface to give meanings for the signals from the PLC. All the data transmission related problems are solved by the commercial products and do not need to be taken care of by the end user. After the OPC connection is established, the SCADA system human machine interface can be designed and the system is ready to use.

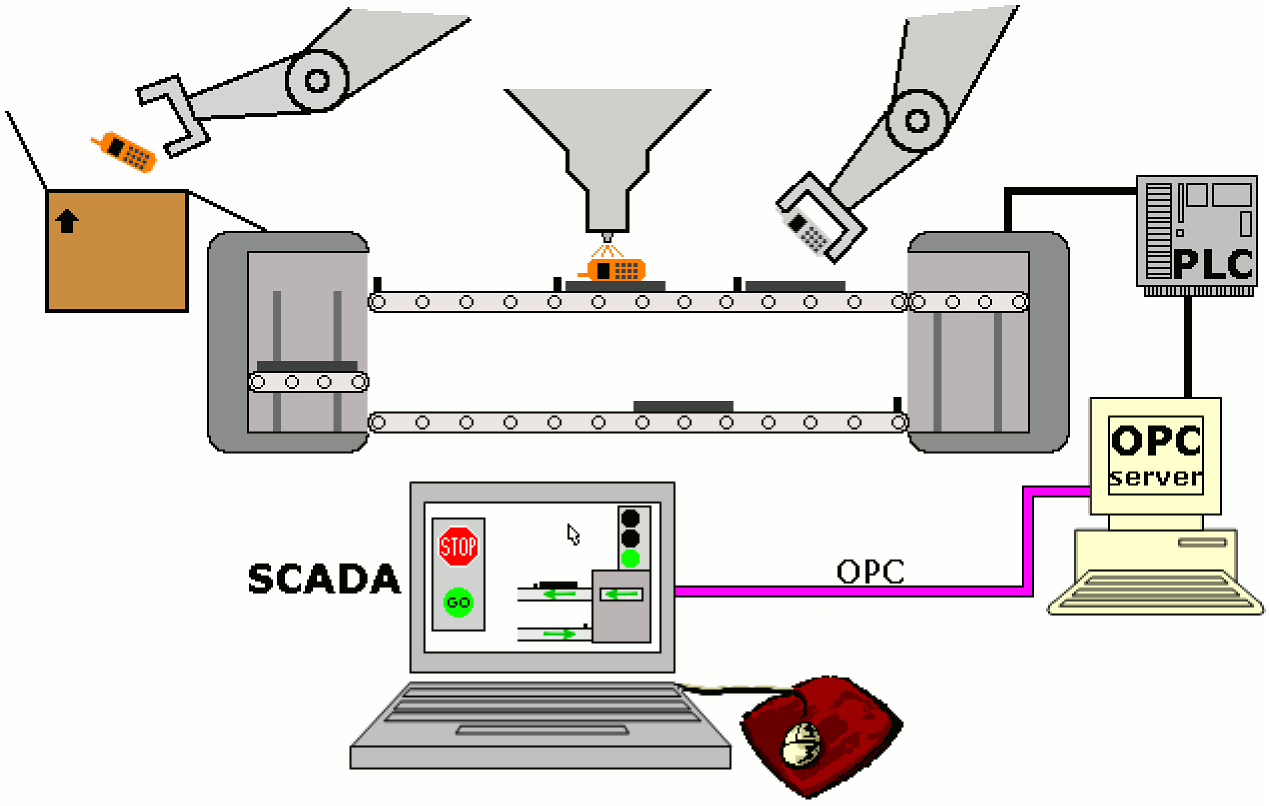

Figure 4. A mobile phone is finished off on an assembly line controlled by a PLC. The PLC receives its instructions from the OPC server. The SCADA system is connected to the OPC server through the OPC interface, thus being in control of the elevator.

Besides being able to solve the original driver problem, the OPC specification has been developed further to include other features. The six other completed specifications include support for the following: alarms and event based communication, specialized interface for batch processes, server–server architectures, accessing history data, security, and communication based on XML (*Extensible Markup Language*). The two unfinished specifications add support for more complicated data types and extended use of commands. [9]

1.2.4. Strengths of OPC

In this subsection, the strengths of OPC are summarized and analyzed from a practical view. In the following, three different viewpoints are presented: one of a software engineer, one of a hardware manufacturer, and one of a customer purchasing the software and hardware components for building a control system. Some benefits common to all are listed in the final paragraph.

The use of OPC decreases the workload of a software engineer in several phases of the project: Since OPC is an open specification, anyone is allowed and able to create an OPC client or server. Moreover, no effort is needed to design the communication protocols. Implementation is also easier by reusing ready-made server applications. Even the maintenance phase is simpler, because even if the hardware component was upgraded, few changes would have to be done to the software.

The most important advantage for a hardware manufacturer is that to serve most of the customers, only one common interface has to be supported. Thus little software related work needs to be done for an end product and the focus of product development can be aimed at the core competences of the company.

OPC gives flexibility to customers building a system out of both software and hardware components on the market. Because OPC is usually well supported throughout the scale of selectable software and hardware components, the interoperability of different components from different vendors is achieved. Customers can therefore switch between hardware devices, replace damaged devices, or upgrade them, in both the constructing and the maintaining phase. A software update or changing to a completely different software application is also possible with ease.

Be it a hardware manufacturer, a software engineer, or a customer, the following benefits apply: Using a common interface which the software and hardware can communicate through, both the hardware manufacturer and the software engineer are independent of each other. This also gives the customer the independency to choose between the different components solely based on their core properties instead of on the certain hardware or software they support. Another thing is that OPC is widely adopted and used. Thereby the performance and functioning of OPC has been verified in a wide diversity of real world applications which has enabled improving the properties and quality of the specifications.

1.2.5. Weaknesses of OPC

It is obvious that OPC is not a solution to communication in each and every technical system; for instance, small systems with special purpose components may not get advantage from an additional middleware component. Nevertheless, the specification has also some definite flaws, the most remarkable of which are described further on.

Many of the weaknesses of OPC are related to its origins: the OPC specification was developed in mid-’90s, when the field of technical systems was rather different. The improvement in computational performance and the development of network and communication technologies has given new prospects for industries but on the other hand has created new requirements for technologies used to handle the situation.

The specification was originally built for simple message passing between software applications and process control hardware. After that, the specifications have been extended piece by piece to make possible to use it with various different kinds of systems and with more suited communication methods. The outcome has therefore become somewhat fragmented. [14]

The fragmentation of the specification has led to a situation, where the different parts of the specification had no connections to each other; in order to utilize different parts of the specification, an individual server for each of the parts had to be created. What it means is that, for instance, a value read from a hardware device through the Data Access Specification was not in any way connected with the very same value read through the Historical Data Access Specification. [14] Another manifestation of the fragmentation is that with different functionalities divided between different servers, the address space models and data hierarchies can be different in different servers [15].

Despite the further development, the OPC specifications can still be used mainly for low level communication only; the ability for a comprehensive control over the whole automation system from the plant device level to the ERP (*enterprise resource planning*) level is managed poorly. Hence, with OPC in use on lower levels, additional protocols have to be utilized to pass information to, from, and between higher levels.

Even though there is a part in OPC specification that allows secure connections, the Microsoft DCOM technology, which the specification is based on, provides no support for security whatsoever. Hence the level of security cannot be considered sufficient to meet the needs of the application area of OPC [16].

There is even a bigger issue in the original specification being based on DCOM technology. Even though OPC is an open specification, the base technology practically determines that a Windows operating system has to be used [11]. In addition to the operating system constraints, the base technology with a treelike data hierarchy is becoming obsolete [15]. The users have also found DCOM technology to be difficult to use [11]. In addition, the DCOM technology is proprietary and thus its source code cannot be accessed; its defects cannot be fixed and to debug the whole system becomes rather difficult.

A solution for many if not all of the weaknesses listed above is presented by the OPC Foundation. This solution is called OPC Unified Architecture.

1.3. OPC Unified Architecture

OPC Unified Architecture, abbreviated as OPC UA or UA, is a specification which on one hand is based on the original OPC specification but on the other hand differs substantially from it. UA is not a new supplemental part for the already fragmented OPC specification but a unifying specification based on a new, different architecture. It combines the functionalities of the different parts of OPC under one specification adding new features as well. UA is intended to be used wherever OPC can be used but also in domains not so well suited for OPC. Since many basic concepts of OPC UA are similar to OPC, the focus of this section is on the main differences.

1.3.1. Origins of OPC UA

The development of OPC UA started in 2002 and the first complete version was released four years later in 2006. With the base DCOM technology becoming obsolete and with even Microsoft endorsing the use of cross-platform capable solutions, the goal of the development was set on creating a completely new specification based on different technologies than what was used in OPC. The purpose was to create a specification which would combine all the functionalities of the regular OPC while adding new features as well [17]. Whereas the regular OPC specification consists of nine separate parts only loosely connected to each other, UA, even though divided into separate parts, too, is an entity in which all of the parts work not in separation but as a unity [18].

The main reason to create a completely new specification from scratch was to improve the original OPC specification by correcting its deficiencies presented in the previous section; building upon the base technology of the original OPC would not have made this possible. As the development process was to be started out of nothing in any case, other aspects could be taken into account as well: the base technology was chosen to enable forward compatibility with future technologies and adding new features.

The major weak points to be improved were as follows: The fragmentation issue was to be solved to achieve better quality for the specification and to allow developing improved properties. Secondly, the Windows operating system bondage was to be cut to achieve genuine platform independence which would enable the specification to be used in a broader scope of platforms. Thirdly, the security was given a high priority in the designing of the specification to allow communication in networked systems.

1.3.2. Technical Differences between OPC and OPC UA

Even though the higher level functionalities of OPC UA are similar to the equivalents of OPC, the technical differences beneath the surface are substantial. In this subsection, the differences between OPC and OPC UA are examined from a more technical point of view and the practices to achieve the goals of the development are presented. What benefits result for a user is discussed in the next subsection.

Instead of the DCOM technology, UA is based on SOA (*service oriented approach*) paradigm. It means that the UA server exposes all of its functionalities as sets of services which can be used by UA clients. The specification defines two different ways of communication between a server and a client: XML WS (*Web services*), which is a widely used standard technology in communication between computers, and UA Native, a special purpose binary representation. The former enables communication with any system that can communicate using Web services, the latter enhances the speed of the connection at the expense of flexibility. [11]

The data structure has been completely reformed to allow a lot richer definition of the system. The main differences are as follows. Instead of a treelike structure in OPC, in the information models of UA the nodes form a full mesh network. The amount of metadata has increased vastly; there is a great amount of structural, semantic, and diagnosis data describing the actual measurement data [11]. UA has a class concept supported: real world items can be modelled as instances of classes (also known as objects), a similar approach commonplace in object-oriented programming languages. This allows custom types to be defined, too, since even real world objects such as actuators and sensors can be types [14]. A more detailed view of the data structure of OPC UA is discussed in Section 3.1.

UA emphasizes security. Unlike in OPC, in UA the security mechanisms are a fixed part of the communication. In UA, security, reliability, and AAA (a*uthentication, authorization, and accounting*) are fully integrated into the specification. The level of security that is provided is sufficient to enable safe use over an internet connection. With the security services implemented in the communication protocol, there is no need for an application programmer to pay attention to it. [11] Nevertheless, for systems with tight performance requirements, it is still possible to switch the security off [19].

UA introduces several important fixes to OPC, many of which are related to reliability issues. In addition, a lot of useful features have been added. Other new features than what are discussed above have little importance in the scope of this research theme, though.

1.3.3. Advantages and Uses

The technical differences allow UA to be used in a broader field of application areas than OPC. In this subsection, the advantages of UA are discussed from a more practical view.

A major benefit of UA compared with OPC is its platform independence. The removal of the operating system limitation has enabled UA to be used in various platforms. The SOA architecture provides interoperability among many kinds of devices [11]. With a wide diversity of devices such as PLCs, intelligent modules, and even some embedded devices supporting Web services, the potential installation base of UA servers is a lot wider than of OPC servers [20]. UA can thereby be used in connecting not only a hardware device to a software application, but also a software application to another software application or even a hardware device to another hardware device.

Another major advantage of OPC UA is its capability to integrate an overall information system of a company both horizontally and vertically, the latter being rather impracticable by using OPC. The capability is due to the information model of UA containing not only the values of the variables of the system but also a large amount of metadata describing the variables and their interdependencies in the system. Hence the information acquired from the factory floor level can be utilized with ease even by higher level systems as well. With Web services as the basis of communication, this allows the overall information system of the factory to be integrated by using merely the OPC UA specification as the means of communication. This means that there would be a connection all the way from the factory floor devices through the process control systems (SCADA, HMI) and up to the process and business management systems (ERP), or even to systems in partner companies. [11]

An example of a company information system communicating via the OPC UA interfaces is shown in Figure 5. The PLCs, DCSs, and such controlling the factory floor devices are integrated together horizontally with HMI and SCADA systems using the Plant Floor Network. The Plant Floor Network is integrated vertically to Operations Network through a UA server, the sole purpose of which is to combine the functionalities available for upper level systems, such as MES (*manufacturing execution system*), and expose them through the OPC UA interface. The Operations Network is in turn connected to the Corporate Network via another UA server in order to provide the ERP systems, as well as possible systems beyond an internet connection, with the desired functionalities.

The platform independence and the elaborate information model alone do not assure safe usage over networks. A company's data system is often distributed to the outside of a factory, too, and may be controlled through an internet connection. The use of Web services with efficient security procedures allows UA to be used over network connections as well [11]. Hence the connection between field devices and ERP systems can be established using already existing multipurpose physical data connections.

OPC UA solves a few DCOM technology related issues making the work of an application programmer easier. Since there are no closed interfaces, the application programmer has a better control over the system and the debugging of the system is easier. Naturally the defects and inconveniences of configuration of the DCOM interface give no trouble either with UA.

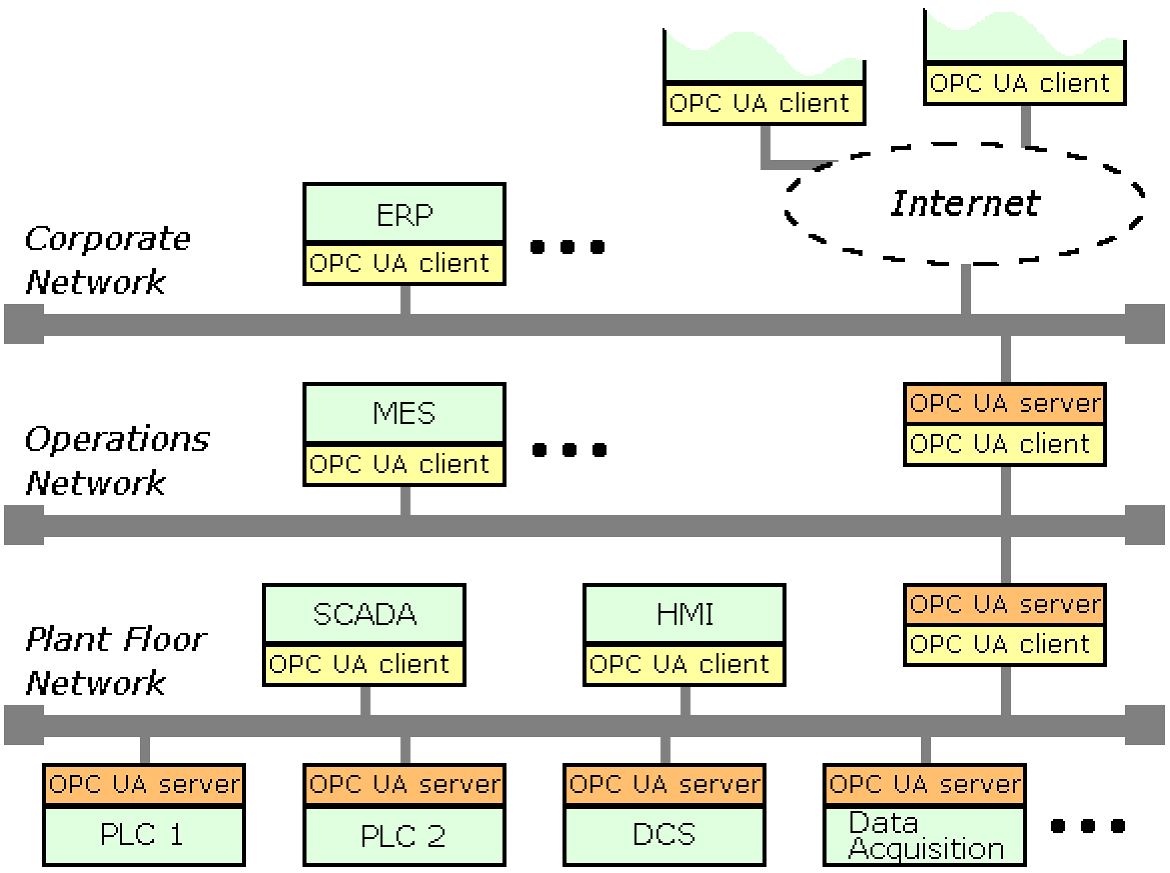

Figure 5. A typical information system of a company integrated both horizontally and vertically with OPC UA

OPC UA enhances the reliability of connection as well. For example, a lost connection can be detected a lot earlier with UA than with OPC and a connection failure can be recovered from by using buffers and resends of data. [11]

The amount of new features allows UA to be used in a much wider area of applications. However, one of the goals in the development process of UA was to retain all of the functionalities of the original OPC [17]. Hence OPC UA can be used practically wherever OPC can be used. As there is already a broad adopter base using the regular OPC, it is also crucially important for the new specification that the migration to the new specification is made easy. That is why the OPC Foundation has assured backward compatibility between the two specifications.

There are two possible ways to equip an OPC capable system with the OPC UA communication methods: One solution is to create a UA server which is directly connected to the control system interfaces. The other solution is to use a middleware to adapt the already existing OPC interface to behave like UA [11]. The former option is naturally the more time-consuming way. However, in order to use fully all the features of UA, such an approach has to be chosen [14]. The latter option is the easy way since the OPC Foundation provides wrappers and proxies; using middleware does not require any changes to the existing server.

1.3.4. Disadvantages and Criticism

Compared to the regular OPC there are still some disadvantages in OPC UA. Many of them are practical aspects which are due to the novelty and the extent of the new specification. One of the biggest concerns of the new technology, though, is the performance. The main performance issues concern data transmission and computational requirements.

Transfer rates of data networks are constantly increasing but so are the system requirements. With text-based XML messages used in transforming information, the majority of the bandwidth is gone wasted from the information throughput viewpoint. Thus transferring a certain amount of information is substantially slower than with the regular OPC; with large quantities of data, the regular OPC can be roughly 20 times faster [21]. To remedy the slow speed of information, the UA Native binary representation can be used. It provides a lot faster communication; the speed is nearly the same if not better than with the DCOM technology [22]. Since the Web services technology is supported by many of hardware devices and software applications on the market, the use of the binary representation decreases interoperability and additional work needs to be done in order to interpret the binary representation. Hence it is at its best in applications with high information transfer requirements.

The other performance issue is security. The practice has shown that the security functions require a huge amount of processor time. The security can be turned off but then other practices need to be used to ensure the trustworthiness of the connection.

For an application programmer, the new specification gives additional workload. Because of the extent and elaborateness of the specification, a lot of work has to be done even when creating small applications [15]. Even though the whole interface does not have to be implemented, the amount of work to achieve at least some minimal functionality is much greater than with OPC. The foundation offers help by providing communication stack reference implementations in C, C#, and Java languages and sample client and server applications for C#. However, to build a software application upon a mere communication stack is very time-consuming. If C, C++, or Java is chosen as the language, using a commercial product to implement the required service sets is advisable.

Another issue is the installed base. Since UA is a fairly recent specification, there are not yet as many products supporting UA than OPC. Thus the interoperability cannot be fully utilized. There is no guarantee that UA will gain a similar status than what its predecessor has and only time will tell whether the new features will prove to be necessary enough that users of old systems will change over from OPC to UA and new systems will be designed with OPC UA as the means of communication.

1.4. Motivation

Modelling and simulation have become essential tools in developing today’s technical systems. They can be used in designing and optimizing, virtual prototyping, and verifying the system, to name only a few. However, to realize the full potential of modelling in both the development and the use of a technical system, a link between the model and the system is to be established.

Incorporating OPC and OPC UA interfaces into OpenModelica makes possible to utilize the information provided by the OpenModelica simulation in the controlling of the corresponding real-world system. Since OPC UA can be used not only with hardware devices but also with other software components, incorporating the UA interface would enable OpenModelica to be connected to different kinds of software applications as well. These applications can be used, for example, to monitor or to control the simulation. An example of how the UA interface connection can be used in two different ways is shown in Figure 6.

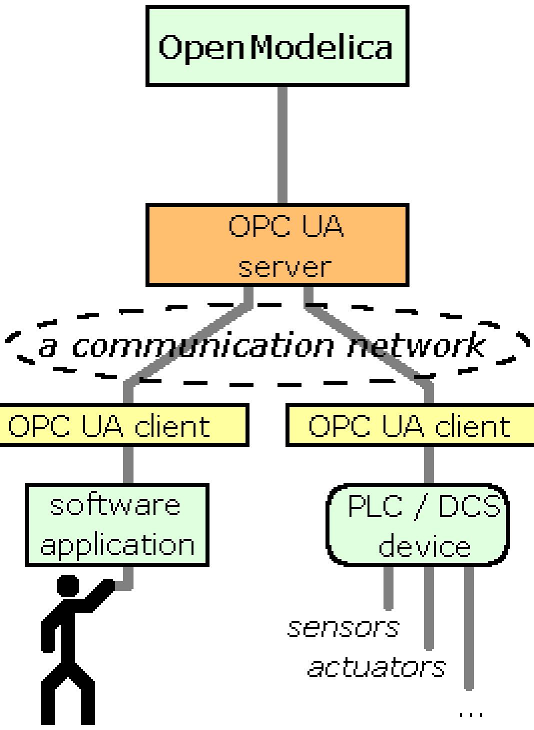

Figure 6. Two different ways to utilize OpenModelica through the OPC UA interface[*]

The first example of how to use the UA interface is to connect the UA server of OpenModelica to a software application capable of acting as an OPC UA client. The software application can be used to monitor and to control the simulation. In addition, the data can be used in whatever way the user decides and since the data from OpenModelica has a defined form, it can be further processed by the application.

The second example of utilizing the simulation model created and run by OpenModelica is to use it in model-based control. In this case, the overall control system consists of field devices, such as various kinds of sensors and actuators, a PLC, which the field devices are connected to, and OpenModelica itself. The PLC acts like an OPC UA client and since there is an OPC UA server connected to OpenModelica, the PLC can communicate with the simulation model. The PLC can, for example, read the value of any variable in the simulation, set parameter values, and run and interrupt the simulation. In the scope of this example, the most important operation would be to run the simulation one time-step forward. This would allow the PLC to give control signals to the actuators on each time-step based on the simulation result, inputs from the sensors, and pre-defined control laws.

In Chapter 2, the goals, including the minor ones, of the project are examined. A few additional motivations for the work are presented as well.

---

[*] A notable difference to Figure 5 is that the PLC device acts like an OPC UA client instead of a server. However, even when not explicitly depicted in Figure 6, the PLC can simultaneously act as a server as well in order to receive commands from higher level systems.

2. Goal

The main task of this project is to design a method for OpenModelica to communicate through the OPC UA interface. The target is to enable the communication method to be used as a part of even fairly large systems containing a great amount of variable information to be processed and transferred. Hence the most important quality attribute of the implementation is computational efficiency. The project has also a few other goals which can be achieved relatively easily by reusing parts of this work in the advantage of other software systems. In this chapter, the objectives of the project as well as some background information about those relating software systems are presented.

At present, OpenModelica has an I/O module called OMI (*OpenModelica Interactive*). However, the adding of the UA interface would vastly increase the I/O capabilities of OpenModelica. For instance, it would enable plug-and-play standard connectivity, provide a wider set of features like security, and improve performance. The UA interface is intended to not use the OMI interface but to be implemented alongside of it.

Once the UA interface is successfully configured to OpenModelica, it can be used in any system in which OpenModelica is embedded. The big picture of this project includes that OpenModelica with the UA interface is used as a plugin in the Simantics platform developed by VTT Technical Research Centre of Finland. Simantics is a simulation environment which consists of various simulators as plugins within an Eclipse based graphical user interface, the integration of the different solvers being a major strength of the platform. [23]

One of the goals of the project is to have a UA connectivity added to the Apros software developed by VTT in collaboration with Fortum, a Finnish energy company. Apros software is multifunctional software for modelling and dynamic simulation of processes and different power plants. It can be used for, for example, safety analysis, process design, training or automation testing. [24] To enable connecting the UA interface to Apros, the Adda (*Advanced data access*) interface [25] supported by Apros is to be used. Hence a new interface layer is added in between OpenModelica and the OPC UA interface. The Adda interface will be further analyzed in Section 3.3.

Another goal of the project, besides developing the OPC UA interface to OpenModelica, is to enable OpenModelica to be used through the regular OPC as well. Even when this can be achieved using wrappers, another solution is proposed. *OPC COM DA Kit* and *OPC XML DA Kit* are OPC implementations for the Adda interface built to enable OPC connectivity in Apros. By utilizing the OPC Kits with the Adda interface, no extra work needs to be done to achieve OPC compatibility. Furthermore, by using the Adda interface, the OPC UA layer can be bypassed and thus excess computational overhead can be avoided.

Figure 7 depicts a completed version of the previously shown picture now including the new interface layer as well as new peripherals: In the picture, the Adda interface is inserted between the OPC UA server and OpenModelica. The two components, Apros and the OPC wrapper, are connected to the Adda interface, thus enabling both the PLC supporting the regular OPC to be connected to OpenModelica and Apros to be connected to the OPC UA interface.

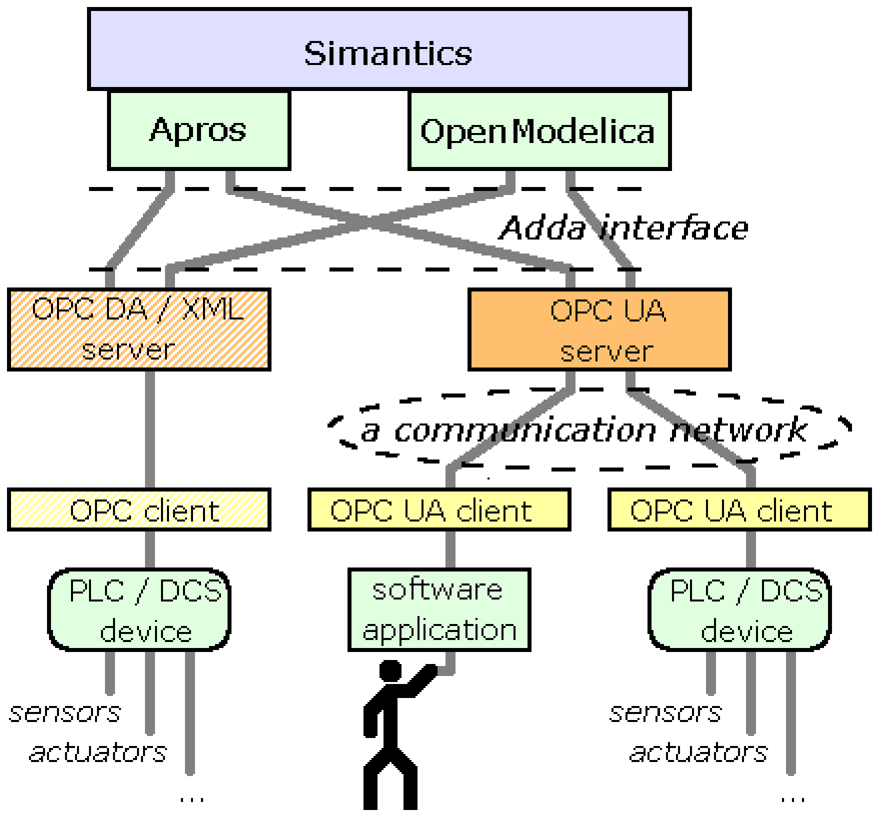

Figure 7. Apros software is connected to the OPC UA interface and the OPC interface is added to OpenModelica.

3. Interface Descriptions

In this chapter, the functionalities provided by the interfaces are discussed more deeply yet still from a practical perspective. An overall view of each interface is presented as well as the key concepts are explained where necessary. The main focus is on the functionalities most important for this project. The matters concerning the differences of the interfaces and how they are connected together are covered more thoroughly later on in Chapters 4 and 5. However, certain comparisons between the interfaces are made already in this chapter.

3.1. OPC UA

3.1.1. Address Space Model

For a better understanding of the functionalities the OPC UA interface provides, it is good to have some basic knowledge of the address space model of UA. An OPC UA server contains an address space model which, when properly configured, is a representation of the real system behind. The precise definition of the address space model can be found in the Specification Part 3: Address Space Model [26].

Within the OPC UA address space model, there is a lot of metadata describing the properties of various real-world system items and phenomena. As briefly mentioned before, a user can create self-made types to define the items in the system. These types define the general properties of the items they are associated with, not any specific data of a certain item. As an example, a *ThermometerType* type could be created to describe what a temperature sensor object is and how it is linked with other parts of the system.

The address space of UA consists of nodes. Each node belongs to a node class (Variable, Object, Method, etc.) specialized from the BaseNode class, as is depicted in Figure 8. For example, the *ThermometerType* type definition presented above would belong to the ObjectType node class since it describes the properties of an object. Based on the type definition, a *thermometer* object, belonging to the Object node class, can be instantiated to stand for a temperature sensor in a real-world system.

Nodes are in relation to other nodes via references. In this way, different nodes can be linked together in the address space model in a similar fashion to how their counterparts are connected in real world. For example, a *thermometer* object could be connected to its *temperature* variable. The relation could be defined as the *temperature* being a component of the *thermometer*.

Nodes have attributes. Attributes serve as the descriptions of a given node. The use of attributes is strictly fixed in each of the original node classes' definitions thus preventing the user from creating or using custom attributes. For example, the *temperature* of a *thermometer* has a Value attribute (each variable has a Value attribute) which contains a floating point number, such as *22.0*, expressing the result of the measurement.

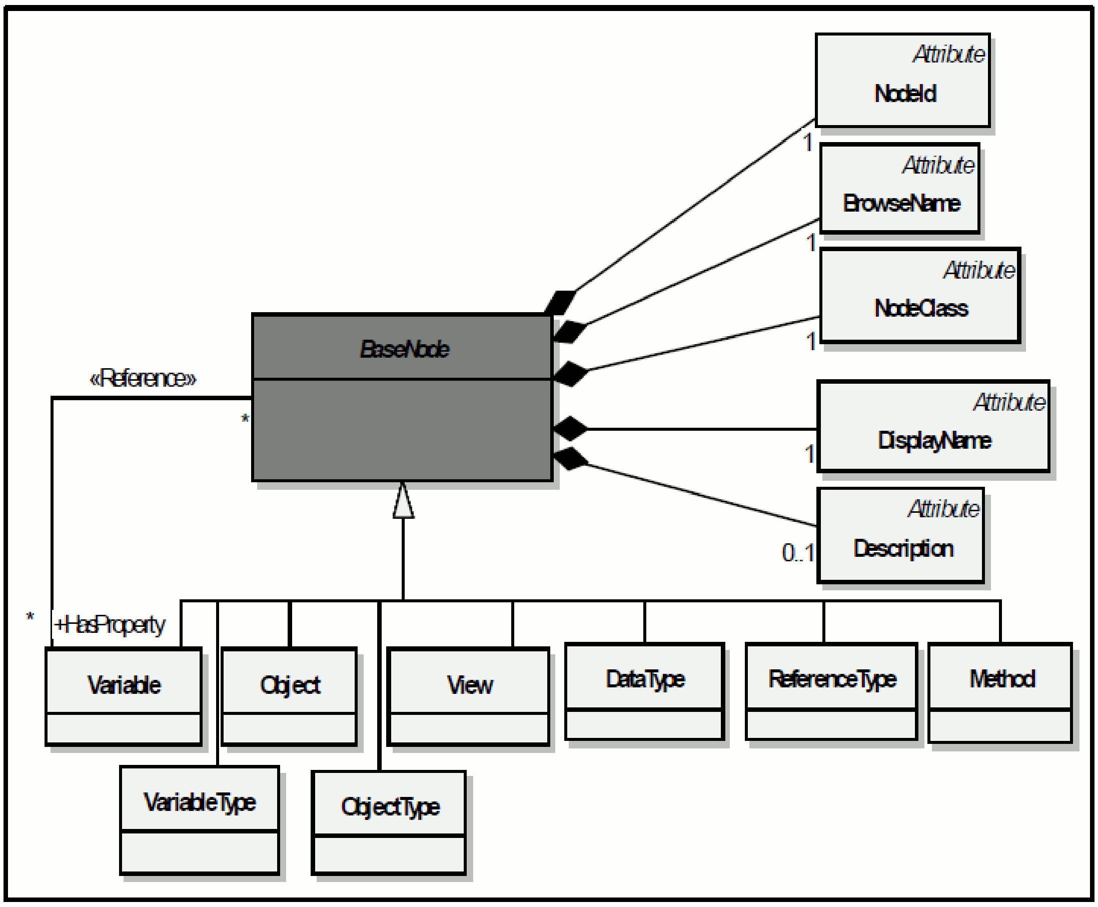

Figure 8. A node in the OPC UA address space model belongs to one of the node classes and has a fixed set of attributes depending on the node class. [14]

In Figure 9 is shown a slightly reduced view of what the address space of the UA server in the thermometer example could look like. As can be seen, the address space forms a non-treelike graph since loops in references are allowed. The address space has a root node, marked with /, of which type definition is *FolderType*; such nodes can be called folders. The root node organizes *Objects* and *Types* of which type definitions are also *FolderType*; hence the root folder has two subfolders. In the *Types* folder are gathered all the type information needed to describe the system, including definitions for folders, thermometers, and temperatures. The *Objects* folder contains the objects and the variables of the system, in this example there is only one object, namely *thermometer*, which in turn has a *HasComponent* reference to *temperature*. Both *thermometer* and *temperature* have references to their definitions organized by the *Types* folder. The reading of *thermometer* is found in the Value attribute of *temperature*. Even when not in full view, all the other nodes as well have a defined set of attributes included.

Even when the data structure of the address space is a graph, it can be flattened to a tree. Hence the address space of UA can be used to model treelike structures as well as be viewed by software understanding merely treelike structures. The discussed thermometer example, for instance, can be flattened to a tree by making copies of such structures which more than one reference is pointing to. For example, the *TemperatureType* type is referenced by both *temperature* and *ThermometerType* nodes. Hence two *TemperatureType* types need to be created, one for each node to be referenced by.

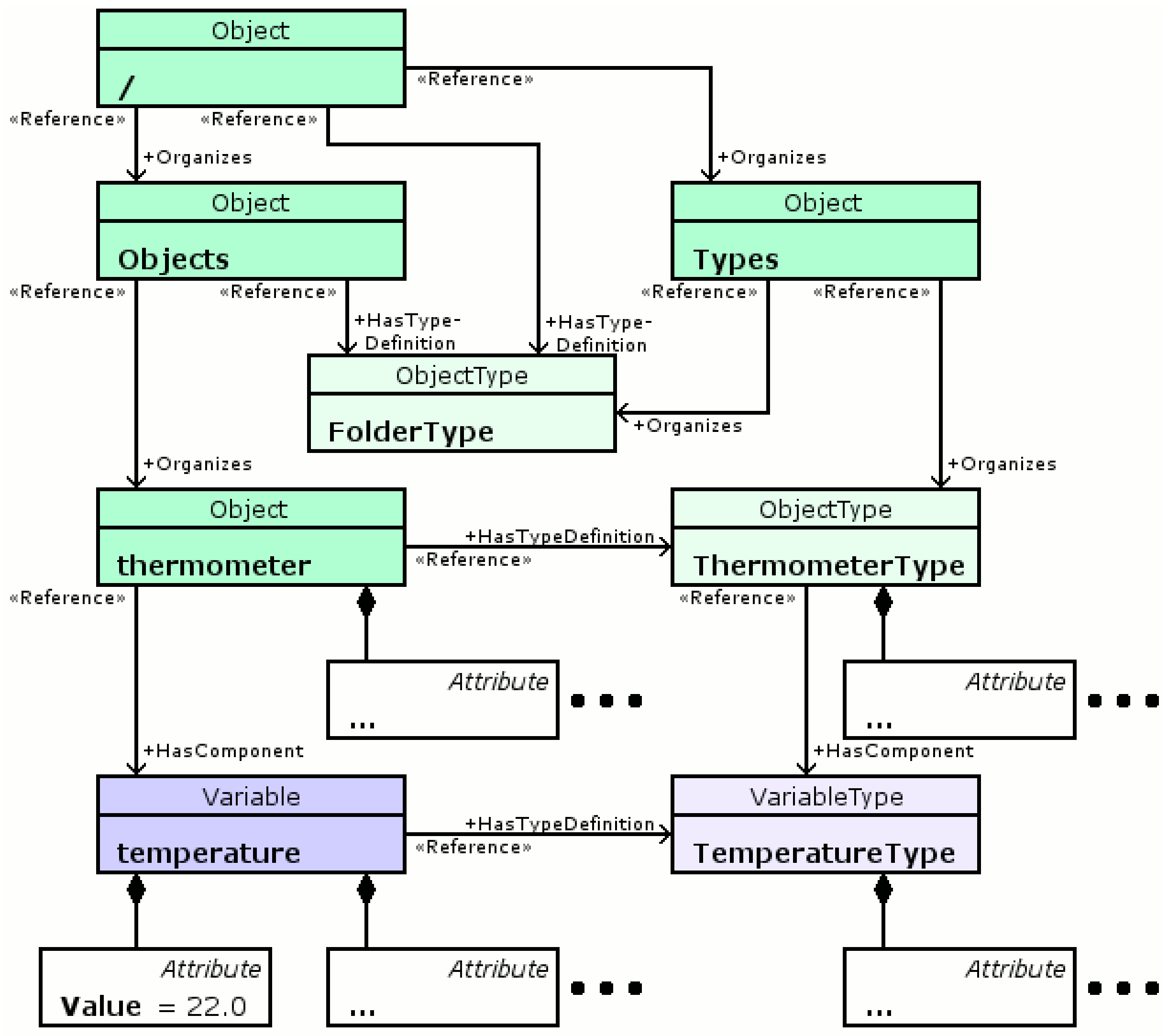

Figure 9. The address space of the thermometer example

3.1.2. Services

The functionalities that OPC UA provides are called services. The services are grouped into different service sets. The services are described in more detail from an abstract point of view in OPC UA specification Part 4: Services [27]. In the following, the service sets are grouped further to correspond better to the functionalities shown to end product users.

- The *Discovery*, the *SecureChannel*, and the *Session* service sets are used in establishing and maintaining the communication channel. The Discovery service set is used to get information about a server, the SecureChannel service set to open the connection between a client and a server, and the Session service set to handle the connection in the context of a session. These service sets are not discussed further.

- The *View* and the *Query* service sets can be used to browse the address space. By using these services the objects, the variables, the methods, their relationships, and what attributes they have can be discovered. Using these services is somewhat analogous to browsing a file system of an operating system, the difference being that in this case the data system is a graph instead of a tree. The two service sets enable, besides acquiring the real-time state of the address space, accessing an address space of a certain point in time in history.

- The *Attribute* service set allows reading and writing the values of attributes, including the Value attribute. The service set allows an access to history of attributes and events, too.
- The *Method* service set allows calling methods of objects. The most important methods in the scope of this project are starting, interrupting, and continuing the simulation.
- The *MonitoredItem* and the *Subscription* service sets are responsible for monitoring the values of attributes. The monitoring can be based on either polling or events. A basic use case in the polling-based approach is that the UA server transmits the value of the item being monitored at a user-defined interval. In an event-based approach the value of the item being monitored is transmitted to the client when a big enough change in the value has occurred. Moreover, for both approaches there are a lot of different ways of how to use the service sets; these different modes of operation are not described further in this report.
- The *NodeManagement* service set provides tools for adding and deleting items and references between them. In the scope of this project, this feature is not used and shall thus not be discussed further.

3.2. OPC

Many of the features and qualities of OPC are similar to those of OPC UA from a practical viewpoint. Thus only the main differences between the interfaces are presented in this section.

The address space model of OPC is somewhat different than of UA and it is not discussed thoroughly. The main difference is that whereas the address space of UA can form a full mesh network consisting of objects, variables, and so on, OPC has a hierarchical treelike structure consisting of folders, items, and properties. Additionally, an OPC client has to divide the items it needs to access into groups through which all of the data transmission is managed. As already mentioned before, using metadata in OPC is supported to a lesser degree than in UA, too.

The functionalities provided by OPC are nearly equivalent to what is provided by UA. The main differences are the lacking of functionality similar to what is provided by the NodeManagement service set and the dissimilar procedures related to opening and maintaining a connection. The former makes reforming of the model impossible through the OPC interface and the latter is shown to the end user as different properties of connection.

A notable practical aspect of the OPC interface in this scope of this project is that even when there is a support for basically all of the essential operations in the OPC specification itself, the OPC Kits, the OPC wrapper software used in this project, implement only two of the OPC specifications: Data Access and XML-DA. With this limitation, to try to use certain functionalities, especially method calls and events, through the OPC interface is not feasible. To patch up the problem, the wrapper software is equipped with an additional proprietary interface for simulation control. This interface provides functions for controlling the execution of the simulation and adds support for events. The simulation control interface is not discussed further.

3.3. Adda

As mentioned earlier, Adda is the interface used in the Apros software for external I/O. The Adda interface is also utilized by the OPC Kits thus enabling OPC connectivity in Apros. Furthermore, by using this software, any application with the ability to communicate through the Adda interface can be made OPC compatible as well.

The Adda interface is a relatively small definition being merely a collection of functions which a software application can use to communicate with simulator software and vice versa. Adda defines a couple of very basic data structures, but otherwise no specific communication protocols are defined. Adda does not thereby specify as comprehensive a communication solution as OPC.

The Adda interface is developed for communication quite similar to what can be achieved by using OPC DA interface. In addition, a set of commands is added for simulation control purposes. A more detailed view of the Adda interface is found in its technical report [25].

The data structure of Adda has a lot in common with the regular OPC; Adda as well can be used to process treelike structures only. The database behind the interface consists of branches and items. A file system analogue for a branch is a folder and for an item a file. The sole functionality of the branches is to organize items. An item consists of the value of that item and some metadata describing it.

The Adda interface consists of functions of four different kinds:

- The first set of functions is the utility functions, the main purpose of which is to initiate and terminate the connection.
- The second set of functions is for browsing the underlying database: the functions provide a similar means of scanning the database than what is achieved by using OPC.
- The third set of functions is for accessing data: to be exact, these functions allow merely writing the data to the simulator as reading a value is executed by accessing the data directly by using a pointer to the corresponding data item inside the simulator. The data access functions also incorporate functions used in the reverse direction: the simulator can inform of a change in the model or in values of the variables in communication.
- The fourth set of functions is for simulation control purposes: the means to start, interrupt, and continue are provided among other functionalities. This set of functions adds general support for events, too.

Despite its differences to OPC UA, the Adda interface can be fitted in a relatively straightforward manner between the OPC UA interface and OpenModelica. When the OpenModelica simulation is running, the interconnections in the model can be expressed using a tree. Thereby the Adda interface gives no constraints in this respect. In spite of Adda providing only a small amount of metadata, it is still sufficient for the relatively modest demands of OpenModelica. Thus using Adda does not give trouble in this respect either. However, Adda sets certain limitations related to implementation details and to what features of the OPC UA specification can be used. These limitations will be further discussed later on where relevant.

3.4. OMI

The OMI interface is the means of I/O at the moment in OpenModelica. However, the interface is rather simple enabling only the most basic communication. [28] The use of OMI as a part of the project is not encouraged because of the reasons to follow.

After the initiation sequence, the OMI interface provides the following functionalities: the simulation can be started, interrupted, stopped, and rewound to a specific time; parameter values of the simulation can be changed; and the group of parameters that are to be monitored can be altered. Other than that, the OMI sends the values of the parameters that are to be monitored to the client after every simulation step. The abovementioned are all of the functionalities provided by OMI, thus the interface is very limited.

The data structure in OMI is not visible through OMI. The database behind the interface is shown as a tree like in OPC and Adda. However, it is impossible to navigate in the data structure by using OMI. Hence other means of acquiring data from behind OMI have to be used anyway. The use of metadata in OMI is completely non-existing. Hence using OMI as the only way of providing information from the simulation is insufficient.

Another reason, besides the lacking of features and metadata, not to use OMI as a link between OpenModelica and Adda is that it would lead to a system with poor performance: the communication is solely based on strings of characters. For example, for a transmission of an integer value, the value must be first converted to a string, transferred to Adda, and converted back to an integer again.

4. Different Architectural Approaches

To solve the problem of incorporating the OPC UA interface into OpenModelica, two different architectural approaches were considered and are described in this chapter. As mentioned earlier, regardless of the architectural choice, a server–client architecture is used: OpenModelica acts as a server and the applications contacting with it as clients.

The first alternative solution is to integrate the UA server as a part of OpenModelica. The second alternative solution is to let the OPC UA server act independently from OpenModelica with having its own means of communicating with OpenModelica. Among other things, the different solutions offer the following mutually exclusive properties: the former enables performance in the shape of high computational efficiency whereas the latter allows flexibility to modify the simulation model without disconnecting the UA server from clients. A closer look at the different architectural options is taken in the following sections where the pros and cons of the solutions are weighed up as well. In Chapter 5, the different solutions are compared and the details concerning the implementation of the chosen approach are discussed further.

4.1. Static Solution

4.1.1. Basis

The first alternative architectural approach is the so called static solution. In the static solution the UA server capability is added to OpenModelica so that it becomes a part of the compiled simulation model executable, hence disallowing changing the model while a UA server is running. With this kind of an approach, the structure of the implementation becomes rather straightforward.

As making changes to the model would yield to recompiling, the UA server included in the model would need to be recompiled along and thus restarted. However, coupling the UA server tightly to OpenModelica brings a couple of benefits: Firstly, tight couplings enable high efficiency. Secondly, the system is rather simple; hence the risks of different kinds of failures in developing such a system are quite low.

In the later of this section, two different implementation methods are presented: a straightforward implementation with no intermediate conversions and an implementation with the Adda interface layer used in between the UA interface and OpenModelica. Even when all of the goals set for this project cannot be met by using the straightforward approach alone, it is presented shortly as a baseline for comparisons of different solution details.

4.1.2. Straightforward Approach

The OPC UA interface can be implemented to OpenModelica in a straightforward manner. It means that the UA server is connected straight to the real-time data inside the OpenModelica simulation without an additional interface in between. However, there would still be separate address spaces in OpenModelica and the UA server. Alternative design choices can be utilized to manage the transfer of data between the two.

The UA server could act in a couple of different ways, as is shown in Figure 10. The first alternative is that when a certain value is to be obtained from the simulation, the UA server polls the data and makes a copy of the value for its own address space. The second alternative is that the address space of the UA server contains merely pointers to the values in the simulation instead of duplicates. In that case, when a value is requested by a client, no additional copying needs to be done as the UA server can read the requested value straight from the simulation and send it to the client. The third alternative is that when a simulation step has finished, OpenModelica could write the values that have changed straight to the address space of the UA server where they can be requested from by a UA client at any time.

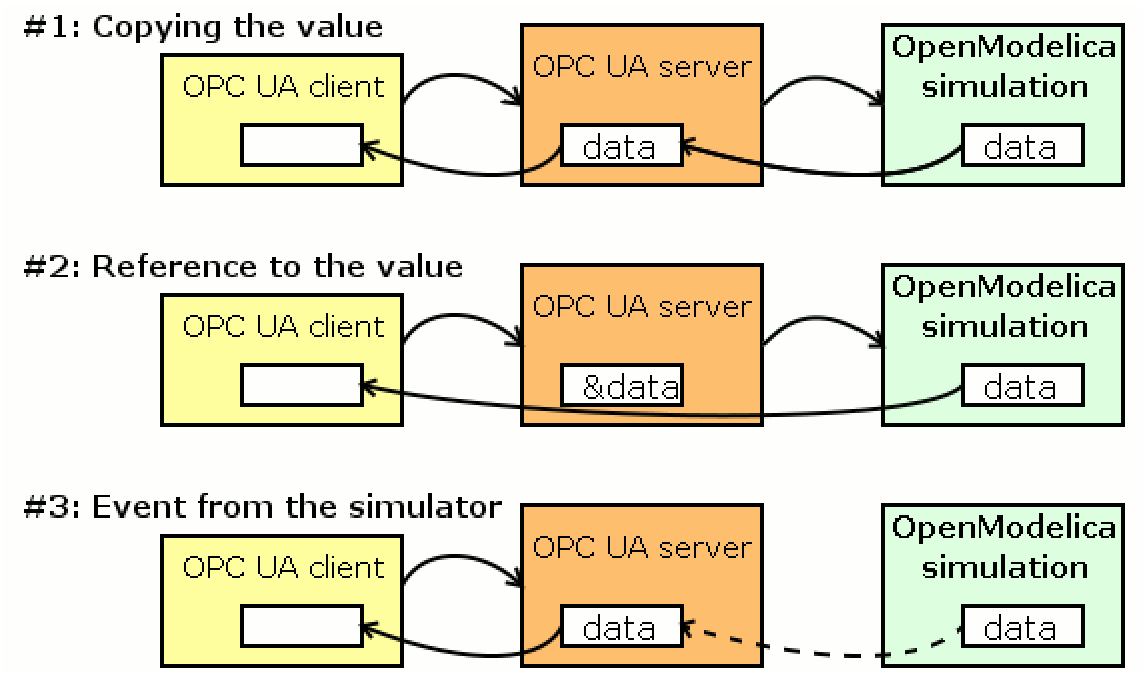

Figure 10. The three methods for passing data from OpenModelica simulation to the OPC UA interface

Using this straightforward approach would not give additional constraints neither to the implementation nor to the whole system. Thus a variety of implementation techniques can be considered and the most fitted opted for. In addition, if new features are desired in the future, only the OPC UA specification will set the limits. The performance of the implementation would also not be limited in any way since no extra work is executed by the application.

4.1.3. Implementation through Adda

As is depicted in Figure 11, a slightly more complex way than the straightforward implementation would be to connect the UA server to OpenModelica through the Adda interface. In this approach, to acquire the desired data out of the simulation, the OPC UA server is created on top of the OPC UA communication stack; it implements the services defined in the OPC UA specifications by transforming client requests to commands available in the Adda interface. The OpenModelica frontend is created to implement these Adda commands. The call-back and other functions which are called from the simulator are implemented in the OPC UA server. In Figure 11, likewise in the following pictures depicting the dynamic implementations, the boxes left uncoloured represent the parts that need to be implemented.

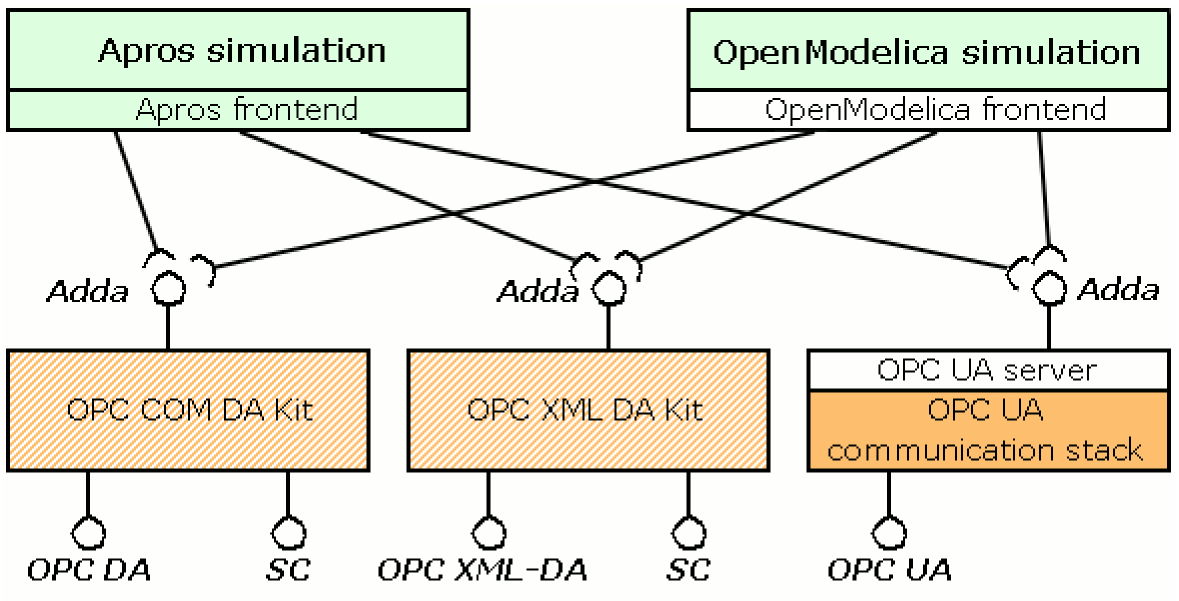

Figure 11. The static solution; both OpenModelica and Apros incorporate an OPC UA server as well as are connected to OPC DA, OPC XML-DA, and Simulation Control interfaces through the OPC Kits. [†]

The including of the Adda interface allows broader interoperability, as mentioned in Chapter 2: Besides through the OPC UA interface, OpenModelica can communicate through OPC DA and OPC XML-DA interfaces in a less computationally intensive manner than what would be possible by using wrapper software. Moreover, interconnecting OPC UA to Adda makes possible the Apros system to be controlled through the OPC UA interface. Naturally, using Adda has a couple of threats to the system performance as well as some clear downsides. The most noteworthy of these threats and downsides are discussed in the following.

First of all, the Adda interface is strongly based on the OPC DA interface which, when it comes to the technical details, differs quite significantly from OPC UA interfaces. Hence, when mapping the functionalities between these two interfaces, certain difficulties occur and rather meaningless functions from the UA server point of view need to be executed.

A layer of abstraction slows down the system a certain amount. The Adda is, however, a very light-weight interface. Since Adda enforces no additional copying of values or using of complex data structures, to use it is not too much more processor intensive compared with the aforementioned straightforward solution. Hence the influence that Adda has on the performance of the system is most likely a small.

Since the Adda interface provides little support to represent metadata, it gives limitations to utilizing the structural representation of OpenModelica. Thus, for instance, the class information about the objects in the model cannot be transferred via Adda.

[†] To be precise, the Adda interface provides no functions to be called from software applications connected to the simulator software. Instead, Adda provides registered functions that are called from the simulator at the start of the program execution. These registered functions are used to relay pointers to the simulator functions that are called from software application.

As a demo implementation using Adda or a comparable interface is not completely finished, all of the points above cannot be either proven or disproven. However, since Adda is not too large a specification, it can be analyzed in a fairly comprehensive manner and therefore there is little risk that any surprising obstacles in the development are to appear.

4.2. Dynamic Solution

4.2.1. Basis

The other implementation approach would be not to include the OPC UA server to the compiled OpenModelica simulation executable but to implement it as a stand-alone entity. With this kind of an approach, the UA server would have to be created only once and thus a new server would not have to be regenerated every time the model is changed. This would allow the model to be modified while the UA server exposing the simulation model is still connected to its clients. As there are quite a lot of unknown factors in the details of the implementation, to draw definite conclusions concerning the main characteristics of the solution is somewhat difficult. Hence, unlike in the case of analyzing the static solution, in here certain assumptions have to be made and different possible scenarios contemplated.

An example situation where the dynamic solution is needed is as follows: A designer is constructing a model needed in a model-based control system by using an OpenModelica environment of some sort. Simultaneously with the model construction phase, the model is wanted to be simulated while in connection with other systems, such as other models or simulations in other applications, other applications controlling or monitoring the simulation, or even a real-world system. With a dynamic solution, the other systems can access the data within the simulation in the model construction phase without having to establish the connection to the OPC UA server all over again each time the model is altered.

As the OPC UA server is not compiled along with the model, it will not have to be restarted whenever the model is altered. Instead, when a model change occurs, only the connection between the old model and the UA server is cut and the connection to the new model is established. The UA server is informed about the change by OpenModelica after which the UA server can in turn notify its clients.

Compared with the static solution, the dynamic implementation will introduce an additional step on the communication path from OpenModelica to the UA server: a piece of software, hereinafter called the OPC UA frontend, is to be created to handle the communication between the OPC UA server and OpenModelica simulations using communication methods of some sort. The main difference of the new OPC UA frontend compared with the frontend of the static solution is that it uses dynamic means of communication with OpenModelica simulations thus allowing model changes. In detail this means that when a change occurs in the model, that is a model is replaced by another one, the frontend replaces the connections to the deleted model by respective connections to the newly created one. As an example, if the model is replaced by an identical model but with one component removed, the frontend changes the existing connections to point to the corresponding components within the new simulation, excluding of course the connections to the deleted component. Once the connection change is done, the frontend shall inform the UA server about the model change, after which the UA server in turn informs its clients. A client connected to the UA server will only need to rebrowse the very point in the address space that was changed to be able to continue normal operation.

4.2.2. Implementation Approaches

One possible technique to implement the dynamic approach is depicted in Figure 12: A static implementation of the OPC UA interface is first embedded to OpenModelica simulations. This static UA connectivity is in turn used as the means of communication between the dynamic UA server and OpenModelica. The frontend is responsible for mapping the address space of the simulation to the address space of the dynamic UA server. As the dynamic UA server is started, it establishes a connection to the OpenModelica simulation 1 through the OPC UA interface. When the model is altered and thus the OpenModelica simulation 2 is generated, the connection to the first simulation is cut and a new one is created to the simulation 2.

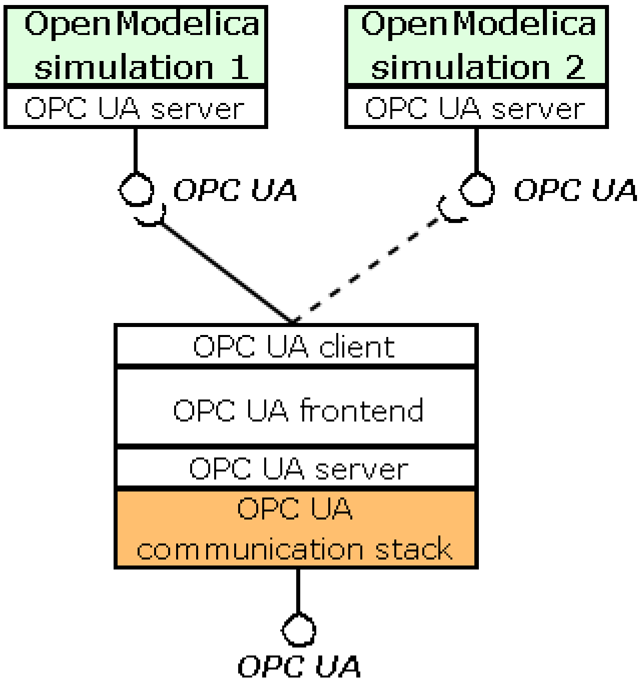

Figure 12. The dynamic solution that utilizes the static OPC UA interface in communication between the dynamic OPC UA server and OpenModelica simulations

A more probable choice for the implementation than what was presented above is to integrate the UA server as a part of the Simantics platform, as is shown in Figure 13. To connect to simulations, the frontend could utilize the *Variable* interface of Simantics. With such an approach, there would be no need to rely on a static implementation or to build an OPC UA client. Moreover, the Simantics platform would be responsible of switching the connections from the old model to the new one to a certain degree. Besides enabling the connection between OpenModelica and OPC UA, such an implementation could be reused to enable Apros, as well as other modelling environments within Simantics, to communicate through the UA interface, as well.

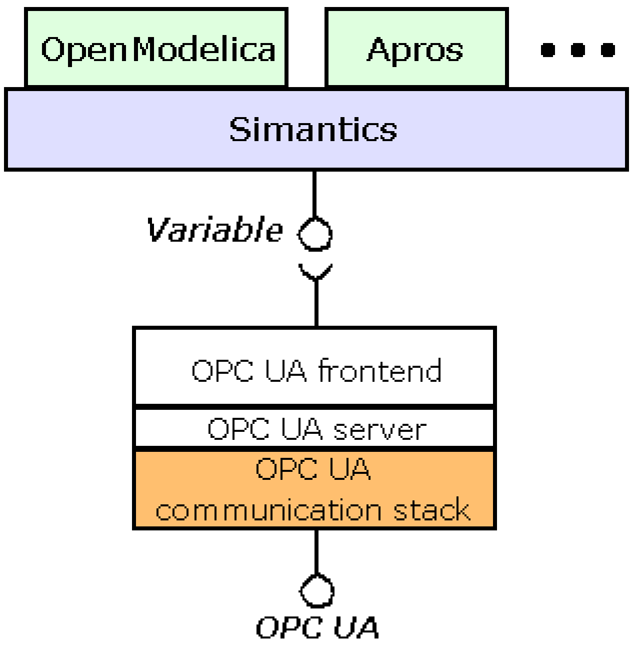

Figure 13. The dynamic solution that uses the Variable interface of the Simantics platform to handle the connection between the OPC UA server and OpenModelica simulations

A downside for both of the approaches presented above is that the dynamic server cannot be utilized via the regular OPC interface by using the OPC Kits; to allow OPC connectivity, the Adda interface would have to be utilized. If the frontend was coupled to the UA server through the Adda interface, the implementation of the UA server itself ought not to differ from the one proposed in the previous section. However, because reading and writing values in the Adda interface is handled by using pointers straight to the simulation data, one major difficulty will follow: the only feasible solution would be to maintain a copy of the original address space of the OpenModelica in the frontend, with duplicates of the variable values as well. The copy would be kept updated by using events from the simulator or by polling. Implementing the dynamic solution with the Adda interface included would thereby lead to many uncertainties in the development process.

As described above, introducing the dynamic solution makes the implementation more complex by bringing a new layer of abstraction to the communication system. Hence the time allocated for the implementation process needs to be evidently longer than with the static solution. The complexity also brings unpredictability to the estimation of the actual workload. A new layer of abstraction leads to a decrease in performance. However, if a computationally effective static solution is implemented alongside, the dynamic solution would not need to have high performance.

5. Chosen Approach

In this chapter, the chosen approach is selected and the choice is justified. The details of the implementation are then discussed more deeply as well as are the performance issues concerning the implementation. Lastly, different scenarios of how to continue the development after an initial version has been created are presented.

5.1. Comparison and Rationale

The two solutions described above have considerably dissimilar capacity demands as they emphasize the two mutually exclusive quality attributes: computational efficiency and flexibility. When used in model based control, there is no need to provide the ability of modifying the model, a feature which a designer of the system would need. Conversely, the designer does not need the computational efficiency which is needed in real-time control. Since efficiency limits flexibility and vice versa, two different architectural approaches should be utilized in order to achieve the different goals.

At this point of the project, the static solution using the Adda interface is selected to be the chosen architectural approach in incorporating the OPC UA interface into OpenModelica. By using either one of the solutions, the Adda interface could be utilized and the minor goals related to connecting UA to Apros and OPC to OpenModelica achieved. However, since the dynamic implementation with Adda may introduce significant problems and cannot guarantee high computational efficiency, the static solution is chosen over the dynamic one.

In addition to the higher computational efficiency of the static solution, there are other reasons as well to support the choice. The static solution is easier to implement and its properties can be analyzed more easily. Hence the implementation can be completed in a shorter time span and the succeeding of the project can be assured with a greater amount of certainty. Another reason for developing the static solution instead of the dynamic one is that once the static solution has been implemented, it is conceivable that parts of the implementation can be reused to develop the dynamic solution. Even if the components were not directly reused, the methodology employed for the static implementation could be used to help the developing of the dynamic implementation.

Some of the implementation problems can be solved more effortlessly by using the dynamic approach, though. An example of such a problem is accessing the simulation instantly after it has been created. Since the simulation starts running right away when it is started, a UA client implemented in the static manner has to deal with all the connecting procedures with the simulation already running before obtaining access to the simulation. With the dynamic solution, the connection can be established beforehand and thus the control is gained at the moment the simulation is started. The above example is rather a special case, though; the great majority of problems demand evidently more work when solved with the dynamic solution.

5.2. Implementation Details

In this section, the main details concerning the implementation are discussed further, yet still on a relatively high level. In addition, alternative methods for the implementation of certain functionalities are compared with each other as well as some optionally included functionalities are presented.

To facilitate building an OPC UA server, the OPC Foundation provides reference implementations for the communication stacks written in C, C#, and Java. Since the stack provides mechanisms for message passing only, to start building a UA server on top of a mere communication stack is very time-consuming: all the higher level application layer functionalities have to be written from scratch. In addition, as is stated by the OPC Foundation, such a way of developing a UA server is not encouraged because of, among others, interoperability reasons [29]. Thus a *software development kit*, SDK, can be used on top of the communication stack. In the scope of this report, an SDK is software which supports the application programmer to implement a UA server. It provides implementations for the service sets of UA, helper classes for often used functionalities, security handling, and so on, thus leaving nothing but the connection between the system and the UA server on the responsibility of the application programmer.

The software development kit chosen to be used in this project is *UA SDK C++* [30] from Unified Automation. The licence for the full version can be acquired for the price of €13,000. Even when there is a C# implementation offered for free by the OPC Foundation, using C++ ensures higher efficiency with no burden of a virtual machine or a need to include the extensive C# standard libraries. Furthermore, with the most parts of OpenModelica being written in C++ as well, to use C++ comes in a natural way. An evaluation version of the SDK is available for free and has been used in order to familiarize with the features of the software. The evaluation version has been used for early prototyping of the implementation, too.

The Unified Automation SDK offers a comprehensive set of functionalities to enable creating a UA server application: Everything from starting and running the server application to managing connections and sessions with clients can be achieved with no effort from the application programmer. Functionalities such as managing nodes in the address space and handling subscriptions are also readily implemented. The software package also contains a sample implementation of a UA server and sufficient documentations to ensure a gentle learning curve.

With a major part of the UA server readily implemented by the SDK, all that needs to be done to create a connection between OpenModelica and the OPC UA interface is as follows: Firstly, the address space of OpenModelica has to be mapped to the address space of the UA server. Secondly, functions of UA, such as read and write, have to be mapped to the corresponding functions in the Adda interface. Thirdly, functions provided by the Adda interface have to be implemented in OpenModelica. The same applies vice versa if such functions are needed. The three phases are described in the following.

The address space of OpenModelica is mapped to the one of OPC UA in a straightforward manner. The Adda interface can be used to handle only trees with branches and items. Hence the address space of the OPC UA interface is a tree as well. Modelica Objects and Components are shown as folder-like objects and variables, respectively, in the OPC UA interface.

The functionalities of the OPC UA interface are implemented only to a certain extent. With the restrictions caused by the Adda interface, it would not be even possible to implement a system supporting all the functionalities provided by the OPC UA interface. In the scope of this project, the OPC UA Data Access Specification is implemented to OpenModelica: the minimum target is to implement basic functionalities for browsing the address space, reading and writing values, and subscribing for variables. Since the SDK does most of the work on behalf of the application programmer, the main things that need to be done are to scan the whole address space of OpenModelica for starters and implement the read and write functions to acquire data from the simulation. However, in addition to mapping these functionalities to Adda functions, some extra work needs to be done. For instance, since the Adda interface introduces a grouping ideology similar to that of OPC, additional function calls need to be made before an access to a variable value can be gained. In addition to implementing the Data Access specification, since the execution of the simulation is wanted to be controlled as well, the Programs Specification [31] has to be implemented to some extent: the simulation control functions of Adda responsible for starting and interrupting the simulation are mapped to UA method calls in a straightforward manner while the rest of the functions are omitted.

The Adda interface ought to be implemented completely in OpenModelica in order to enable full interoperability with the OPC Kits. However, certain prioritization is performed and thus not all functionalities, such as event handling, are implemented. In addition, because of the differences between Adda and OPC UA, it remains to be seen whether all of the functionalities used by the simulator need to be implemented in the UA server at all as they might not be of any use.

The SDK used does not support variables in its address space to have pointers to values in the underlying simulator as such. Instead, when a client requests an access to a certain variable through the UA interface, the server copies the value from the simulator and sends the copy to the client. Since the SDK cannot be completely inspected using the mere evaluation version, it is hard to determine whether applying such a property would be trivial or time-consuming. Hence the optimization of the process is left for further development and the SDK is used as such.

By using an OMI-like interface, integrating OPC UA to OpenModelica would be eminently simple. This is due to that when OpenModelica has finished calculating one step in the simulation, it sends all the values of the specified variables that have changed. If OpenModelica used this kind of a methodology, but with better efficiency than OMI, of course, the third option for accessing the simulator data presented in subsection 4.1.2 could be used: OpenModelica could send all the values which have changed to the UA server. However, as the Adda interface has no support for behaviour of that kind, this implementation option could not be chosen. Another thing would be, though, whether this kind of an approach was beneficial since by using this method one additional copy operation would be mandatory in any case.

5.3. Performance Issues

The main performance goal of the OPC UA interface is that it can be used in systems with a great amount of data to be processed and transferred. Since the SDK is provided with the means of using the UA Native binary protocol to transfer data, only the processing of data needs to be focused on. With a great amount of data to be processed, the key question in optimizing the performance is the computational efficiency of the read and write operations.

A major performance issue is security. The practice has shown that encrypting data can lead to even two orders of magnitude slower performance. For certain systems, however, there is no practical demand for security. Thus the security can be switched off in those situations.

As mentioned earlier, the read and write operations utilize pointers in the Adda interface whereas in the SDK one copy operation has to be made for either operation. If no excessive copying is made in between Adda and OpenModelica, the total amount of additional copies can thus be limited to one. Whether or not this is a sufficient enough result can be determined only by measuring the performance of the implementation and reflecting the results against the requirements.

Before data can be accessed through the Adda interface, a couple of additional function calls have to be made. These function calls are quite meaningless from the viewpoint of the OPC UA. However, as they are not related straight to reading and writing values, they merely bring some latency in the initialization phase instead of slowing down data processing in normal operating conditions. Thereby, these aspects have not much importance.

Another issue is whether the SDK used is otherwise computationally heavy. Only a minor superficial inspection has been made in the field of this subject; no accurate measurements have been taken. However, as the SDK is written in C++, no additional computational overhead is caused by a virtual computer like with Java or C#.

For applications with high performance requirements and with no need for security, most of the importance is laid on having as fast an implementation as possible. Once an initial implementation is completed, estimating the true performance of the system and how big an influence each operation has on it will become a lot easier. Moreover, not until then, there is little reason to optimize the performance to a higher degree. Let the performance be optimized or not, the structure of the software shall be constructed in such a way that optimization can be carried out afterwards if necessary.

5.4. Future Work

In this section, different ideas and visions are viewed concerning how the project could be continued after an initial implementation is completed. New features as well as methods to improve performance are discussed, as well.

The most likely way to begin further development is to start implementing the dynamic solution. Since neither the static nor the dynamic solution guarantees both performance and flexibility, in order to achieve the both properties, the dynamic solution is to be implemented beside the static one. Thus one of the solutions could be used in the model building phase and the other when the model is used as a part of a control system. Moreover, if no strict efficiency requirements exist for the dynamic solution, to facilitate the implementation, a higher level programming language, namely Java, could be utilized in the dynamic solution.

Concerning the static solution, a vision where a parallel implementation not using the Adda interface could be contemplated as well. Omitting the Adda interface would lead to increased performance and allow more evolved functionalities to be implemented. Even when a great deal of such functionalities can be implemented with Adda as well, some cannot. For instance, a comprehensive use of metadata, one of the key advantages of OPC UA, cannot be utilized with Adda involved.

Even without creating a new implementation based on a different architecture, certain features could be added to the existing implementation. One proposed feature related to the simulation control is that the simulation could be controlled by a finite state machine representing the relations between the different states of the simulation and controlling the simulation. The initial implementation has two “states”: running and suspended. However, their relationships are not described in the UA server in any way; instead, the simulation is controlled using start and pause commands which have no knowledge about the state of the simulation. A state machine would allow a UA client to be informed about the state of the simulation as well as provide proper interface of controlling the simulation. If the state machine consisted of more than just two states, more commands to control the simulation might need to be implemented additionally.

The chosen static approach provides no support whatsoever for events. To enable efficient use in model based control, the latencies of the system would need to be as small as possible, a target achievable by using events. Events could be used to inform clients of when a simulation step has finished, about which values have changed, and so on. If a dynamic solution was to be implemented, one solution for the communication between OpenModelica and the frontend would be an OMI-like approach where the simulation values that have changed were sent to the frontend.

Besides using events provided by the UA and Adda interfaces, the abovementioned properties could be achieved by equipping the UA server with a UA client. In such a server–server approach, both OpenModelica and the application in a connection with it would incorporate OPC UA servers with client capabilities embedded. Even when such an architecture is neither proposed nor designed here, it seems that the workload of the migration to such an architecture seems not to be very large.

Besides creating the implementation without Adda, there is another solution to improve computational efficiency. Since the source code of SDK can be acquired as well, it can be altered in order to remove undesired layers of abstraction. Even if the structure of the software would not be touched, using of pointers to the variables of the simulation instead of duplicate values would decrease computational overhead and thus improve performance.

One proposed solution, instead of continuing the development with the Unified Automation SDK, is to implement the whole OPC UA server from scratch. Unlike a commercial SDK, a self-made implementation would not need to be implemented to the same extent; only simple data accessing and simulation control would need to be implemented. Besides enabling to design the implementation to emphasize the features important for this very project, this approach has another benefit, too: By using the SDK, the software created has to be published as a closed source precompiled dynamic link library. On the contrary, when creating the whole software independently with no dependencies to a commercial product, the whole UA server implementation can be published as open source, an approach fitting better to the idea of openness in OpenModelica.

6. Conclusions

Equipping OpenModelica with the OPC UA server capability will enable simulations to be used in controlling complex technical systems in real time. Additionally, OpenModelica simulations themselves can be supervised, controlled, and connected to practically any kind of application capable of acting as an OPC UA client; any software, not just control and automation applications, benefitting from running OpenModelica simulations can do so through the OPC UA interface.

The architectural approach chosen for the implementation enables OpenModelica to be used through the OPC UA connection within systems requiring high performance in both transferring and processing of data. As the chosen approach allows the regular OPC interface to be utilized in an efficient manner as well, both already existing and new solutions whose communication methods rely on OPC can be handled. In addition, to enable the Apros software to be capable of being connected with state-of-the-art process control solutions in future as well, parts of the implementation are reused in order to embed the OPC UA interface to Apros, too.

To achieve the abovementioned qualities, the implementation of the OPC UA interface based on the suggested static solution can be initiated. Additionally, in order to realize the potential of OPC UA that cannot be attained with the static solution, the development can be continued further; once the initial implementation is in a stable condition, a closer study concerning the dynamic solution implementation can be launched. Optionally, other means to improve and extend the implementation can be considered in order to achieve new properties and better performance.

The need for a common open interface through which different software and hardware applications from different vendors can communicate is self-evident. Equally crystal clear is that simulation tools are needed as a part of modern control systems. Once the OPC interfaces are supported in OpenModelica, the broad adopter base of systems already using OPC is provided with connectivity to an open simulation environment. Moreover, OPC UA is seen as a forward-looking solution to work out the I/O problem of OpenModelica; in the future, assuming OPC UA breaks through as a commonly used protocol in communication between applications of both software and hardware, there is a strong likeliness that OpenModelica can be tightly integrated to almost any kind of technical system, including, but not limited to, process control and manufacturing automation systems.

References

[1] OPENPROD: OPENPROD – Research Project. Available at: http://www.ida.liu.se/~pelab/OpenProd [Accessed 12 July 2010].

[2] ITEA Office: ITEA2. Available at: http://www.itea2.org [Accessed 12 July 2010].

[3] Modelica: Modelica Association. Available at: http://www.modelica.org/association [Accessed 28 June 2010].

[4] Fritzson, P.: Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. IEEE Press. 2004. ISBN 0-471-47163-1.

[5] Fritzson, P.; Engelson, V.: Modelica — A Unified Object-Oriented Language for System Modeling and Simulation. Lecture Notes in Computer Science, Volume 1445/1998. Pages 67 – 90. Springer Berlin / Heidelberg. ISBN 978-3-540-64737-9. DOI 10.1007/BFb0054087. Available at: http://www.springerlink.com/content/2h82h371600781t4/fulltext.pdf [Accessed 28 June 2010].

[6] Fritzson, P.; Pop, A.; Aronsson, P.; et al.: OpenModelica Users Guide, Version 2009-11-09 for OpenModelica 1.5. 2009.

[7] OpenModelica: Introduction. Available at: http://www.openmodelica.org [Accessed 28 June 2010].

[8] OpenModelica: Consortium. Available at: http://www.openmodelica.org/index.php/home/consortium [Accessed 28 June 2010].

[9] OPC Foundation: What is OPC? Available at: http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp [Accessed 28 June 2010].

[10] OPC foundation: What is the OPC Foundation? Available at: http://www.opcfoundation.org/Default.aspx/01_about/01_history.asp [Accessed 28 June 2010].

[11] Hannelius, T.; Salmenpera, M.; Kuikka, S.: Roadmap to adopting OPC UA. 6th IEEE International Conference on Industrial Informatics, 2008. Pages: 756 – 761. DOI: 10.1109/INDIN.2008.4618203. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4618203 [Accessed 28 June 2010].

[12] OPC Foundation: OPC DA 3.00 Specification. 2003. Available at: http://www.opcfoundation.org/DownloadFile.aspx?CM=3&RI=67&CN=KEY&CI=283&CU=13 [Accessed 28 June 2010].

[13] Kontron Czech: What is OPC? Available at: http://shop.kontron-czech.com/InfoPage.asp?TP=FT&ID=89 [Accessed 28 June 2010].

[14] Leitner, S.-H.; Mahnke, W.: OPC UA – Service-oriented Architecture for Industrial Applications. ABB Corporate Research Center. Available at: http://cimug.ucaiug.mobi/KB/Knowledge%20Base/SOA%20for%20Industrial%20Applications.pdf [Accessed 28 June 2010].

[15] Mattila, M.: OPC:n uudet tuulet. Automaatioväylä, Volume 4, 2005. Available at: http://www.automaatioseura.fi/index/tiedostot/OPC_Lisatieto.pdf (in Finnish) [Accessed 28 June 2010].

[16] Schleipen, M.; OPC UA supporting the automated engineering of production monitoring and control systems. *Emerging Technologies and Factory Automation, IEEE International Conference on*, 15-18 Sept. 2008, Pages 640-647. DOI: 10.1109/ETFA.2008.4638464. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4638464&isnumber=4638343 [Accessed 12 July 2010].

[17] Mahnke, W; Leitner, S.-H.: OPC Unified Architecture - The future standard for communication and information modeling in automation. ABB Review 3/2009, Pages 56-61. Available at: http://www.abb.com/abbreview [Accessed 12 July 2010].

[18] OPC Foundation: OPC Unified Architecture. Available at: http://www.opcfoundation.org/Default.aspx/01_about/UA.asp [Accessed 28 June 2010].

[19] OPC Foundation: OPC Unified Architecture Specification Part 2: Security, Release 1.01. 2009. Available at: http://www.opcfoundation.org/DownloadFile.aspx?CM=3&RI=415&CN=KEY&CI=283&CU=1 [Accessed 12 July 2010].

[20] Burke, T.: Data Exchange. Prime Magazine, Autumn 2008, Page 18. Available at: http://www.onwindows.com/Portals/0/images/prime-issue-14.pdf [Accessed 12 July 2010].

[21] Mahnke, W.; Leitner, S.-H.; Damm, M.: OPC Unified Architecture. Springer-Verlag Berlin Heidelberg. 2009. ISBN 978-3-540-68899-0. Available at: http://books.google.fi/books?id=de9uLdXKj1IC [Accessed 12 July 2010].

[22] Aro, J.: OPC Unified Architecture – uuden sukupolven tiedonsiirtoprotokolla automaatiojärjestelmille ja vähän muuhunkin. Automaatioväylä, Volume 4, 2006, Pages 7-8. Available at: https://www.simulationsite.net/opcfinland/tiedostot/OPCUA-Jouni.pdf [Accessed 12 July 2010].

[23] Simantics: Simantics Platform. Available at: https://www.simantics.org/simantics/about-simantics/simantics-platform [Accessed 12 July 2010].

[24] Apros: Apros in Brief. Available at: http://www.apros.fi/en/apros_in_brief_2 [Accessed 12 July 2010].

[25] Peltoniemi, J.: Adda (Advanced data access) -interface documentation for OPC COM DA Kit or XML DA Kit users. VTT. 2009.

[26] OPC Foundation: OPC Unified Architecture Specification Part 3: Address Space Model, Release 1.01. 2009. Available at: http://www.opcfoundation.org/DownloadFile.aspx?CM=3&RI=338&CN=KEY&CI=283&CU=1 [Accessed 12 July 2010].

[27] OPC Foundation: OPC Unified Architecture Specification Part 4: Services, Release 1.01. 2009. Available at: http://www.opcfoundation.org/DownloadFile.aspx?CM=3&RI=416&CN=KEY&CI=283&CU=1 [Accessed 12 July 2010].

[28] Fritzson, P.; Pop, A.; Aronsson, P.; et al.: OpenModelica System Documentation, Version 2010-05-03 DRAFT for OpenModelica 1.5. 2010.

[29] OPC Foundation: The Benefits of OPC UA. Available at: http://www.opcfoundation.org/Default.aspx/UASDK/Files/The%20Benefits%20of%20UA.htm [Accessed 28 July 2010].

[30] Unified Automation: C++ based OPC UA Server SDK. Available at: http://www.unified-automation.com/c++-based-opc-ua-server-sdk.htm [Accessed 12 July 2010].

[31] OPC Foundation: OPC Unified Architecture Specification Part 10: Programs, Version 1.00. 2009. Available at: http://www.opcfoundation.org/DownloadFile.aspx?CM=3&RI=335&CN=KEY&CI=290&CU=43 [Accessed 19 July 2010].