DOI: 10.15439/2015F160
ACSIS, Vol. 5

Object Oriented Internet

Mariusz Postol
Technical University of Lodz
Institute of Computer Science
ul. Stefanowskiego 18/22, budynek
A14, 90-924 Łodź, Poland
Email: postol@zsk.p.lodz.pl

***Abstract*—The widespread use of the HTTP and hypertext makes it possible to freely publish new information and expose it in the context of its description. Unfortunately, this is a human-centric environment that cannot easily be adapted to an application-centric approach, which is required to provide distributed enterprise management and real-time process control. In this article new architecture is presented that can provide a generic solution for publishing and updating information in the context that can be used to describe and discover it. It is proposed to distribute the publisher (server) tasks to three classes: (a) information context management using the object oriented programming paradigm, (b) a predefined fixed set of services to access data and meta-data, and (c) a pluggable custom process data binding mechanism. It is also proposed to implement this architecture using the OPC Unified Architecture - a new emerging industrial integration standard.**

I. Introduction

BEFORE commencing discussion about how to use the Internet the question "What is it?" must be addressed. Usually we can hear a definition like that: "the public worldwide computer network system that carries a vast array of information resources and services". It seems to be too broad for further discussion. From the definition, it is networking technology providing access to information resources and services. "To get access", the information transfer must be carried out between the Internet users, i.e. a resource or service provider and its consumers. To enable this, the following assertions must be made:

- Communicating parties must use the Internet protocol (IP) [1];
- All access points to the Internet communication infrastructure must have globally unique addresses;
- Users must be attached to those access points.

As long as the above rules are obeyed the communicating parties use Internet communication – colloquially, they are connected to the Internet. Reversing this sentence, we can say that the Internet is an infrastructure connecting any entities following the above principles. From the user point of view that is all, but to traverse information over the network the Internet infrastructure must be smart enough to locate the access points.

From any ICT solution we would expect information processing rather than having only interconnection between the Internet access points. To meet this requirement the Internet users must be processing engines rather than hosting applications responsible for this work. Application to application connectivity is, therefore, required that can be provided by an additional transport protocol. Examples of such protocols are TCP (connection oriented) and UDP (connectionless). It is worth stressing that we can use a variety of protocols as the transport protocol and the above assertions still hold true.

On the other hand, any user expecting information processing from the communicating party is interested in selecting appropriate functionality, but not a particular application instance. Therefore, we can distinguish three meanings of the transport protocol address, called a port:

- Functionality selector – for a consumer interested in utilizing a particulate information resource or service;
- Functionality publication end point – for servers offering resources or services;
- Address to identify sending and receiving application end-points – for the protocol stack.

The TCP and UDP protocols share the same address space with a capacity of 64k end points. Even today, having so many applications hosted on any network node is impractical and hard to manage. Unfortunately, mapping between functionalities and their identifiers is static, which means that the majority of available port numbers are globally unique functionality identifiers governed by the Internet Assigned Numbers Authority (IANA). The others called dynamic/private ones are not assigned and used to identify sending and receiving application end-points only.

Using a global dictionary instead of a description, discovery and integration mechanism results in a fully exhaustion of transport protocol address space and the registration of new functionality becomes very difficult. Therefore, to communicate over the Internet, users need to select one from the 49152 existing options.

The selection should be based on well-defined requirements, but how to define the requirements having only the general assumption that we expect access to the information resources or services. Many aspects may be taken into consideration. For the above assumptions a solution that allows server to freely publish new resources and services is needed. A globally acceptable discovery and integration mechanism is a possible option. Alternatively, another protocol must be selected on the following assumptions:

- The publishing server is responsible for managing the address space;
- The protocol provides an infinite address space capacity;
- The protocol is transparent for the payload transported.

The above assumption make the Internet a publication platform containing countless resources, but, to be useful, consumers must be allowed to find the appropriate ones using a description and discovery mechanism. It requires that publisher must provide additional information (meta-data), which describes the resources to allow the selection. Additionally, the descriptions must be coupled with addresses to selectively access them. For human-centric solutions a graphical interface is an appropriate mechanism. HTTP [1] as a protocol and HTML [2] (more general a hypertext) as a description language are the big winners selected by millions of people and they have led to the establishment of the World Wide Web.

Unfortunately, for application to application connectivity a programming interface (API) is required. Because community acceptance and reuse of the existing solutions is so important for the Internet evolution, a new solution – called web services - atop HTTP has been developed by the World Wide Web Consortium (W3C) [2]. This specifications suit is commonly referred to as WS-* and contains:

- Simple Object Access Protocol (SOAP) to use the services;
- Web Services Description Language (WSDL) to describe the services;
- Universal Description, Discovery and Integration (UDDI) to get access to the services description.

To obtain applications interoperability all clients consuming services provided by a server offering them must conform to a WSDL specification prepared in advance that defines a contract between them. Hence, this process requires software development and, because WSDL cannot provide complete semantics of the service, the process is usually manual and requires conformance testing. There are no good global scope solutions of this issue. Today solution is for the server publisher organization to provide complimentary compliant client applications. In this approach, typical problems like operating system dependence, software updating and versioning must be solved. Finally, it leads to a static solution where functionality is exposed as a fixed set of services.

It seems that the next level of abstraction is needed to meet the above mentioned goal and allow the server to freely publish resources and services. Generally speaking, all ICT systems are expected to provide information processing capabilities. Information is an abstract knowledge; it cannot, therefore, be directly processed by physical machines. To make information capable of being processed, it must be represented as computer-centric binary data. To propose a solution that meets those requirements two questions should be addressed:

- How to get access to (transport) the process data?
- How to represent (model) the information?

To answer the first question we need a globally accepted, platform-neutral (assuring that the above stated assertions hold true) communication standard that allows also addressing the second question, i.e. designing of an appropriate Information Model.

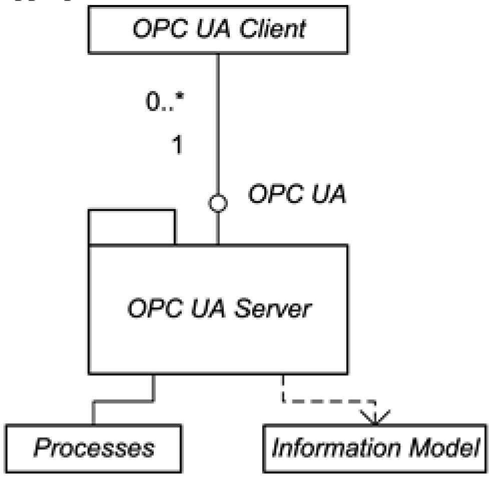

Fig. 1 OPC Unified Architecture archetype

The OPC Unified Architecture (UA) (Fig. 1) technology [3], [4], [5], [6] meets all the requirements, because:

- It is Internet based technology;
- It is a platform neutral standard allowing easy implementation on any system including embedded systems;
- It is designed to support complex data types and object models;
- It is designed to achieve high speed data transfers using efficient binary protocols;
- It is scalable from embedded applications up to the process control and enterprise management/operation systems;
- It has broad industry support and is being used in support of other industry standards such as ISA S95, ISA S88, EDDL, MIMOSA, OAGiS, etc. [7].

It is a broad class of application domains where business IT and control systems are converged in a global scope to make a large whole with the aim to improve performance as the result of the macro optimization and synergy effect. One of the main requirements of the Industrial ICT is to provide a consistent mechanism for the integration of the vast varieties of systems. This requirement can be met as the result of employing the OPC Unified Architecture (UA) as the mechanism for the integration. It is assumed that it should be robust and the implementation should be platform independent. Fig. 1 illustrates the architecture of the proposed solution. In this approach three elements are distinguishable from the typical client server archetype:

- *OPC UA*: an interface representing invariable Service Model [8] responsible for providing client/server connectivity;
- *Information Model*: application domain unique description of a context the process data is made accessible in to the clients.
- *Processes*: source of exposed information resources and services hereinafter referred to as process data.

To make systems interoperable, the data transfer mechanism must be associated with a consistent information representation model. OPC UA uses an object as a fundamental notion to represent data and activity of underlying processes (see Sec. II.B). The objects are placeholders of variables, properties, events and methods and are interconnected by references. This concept is similar to well-known object oriented programming (OOP) that is a programming paradigm using "objects" – data structures consisting of fields, events and methods – and their interactions to design computer programs [9]. The OPC UA Information Model [10], [11] provides features such as data abstraction, encapsulation, polymorphism, and inheritance.

The OPC UA object model allows servers to provide type definitions for objects and their components. Having defined types in advance, clients may provide dedicated functionality, for example: displaying the information in the context of specific graphics.

The OPC UA information modeling concept (Sect. III) is based on layers, which step by step expand the basic model provided by the OPC UA Specification [19]. The Information Model is abstract and hence, in a real environment, it must be implemented in terms of bit streams (to make information transferable) and addresses (to make the data selectively available). To meet this requirement, OPC UA introduces a node notion as an atomic addressable entity that consists of attributes (value-holders) and references (address-holders of coupled nodes). The set of nodes that an OPC UA server makes available to clients is referred to as its Address Space [4], [5], [12], which enables representation of both underlying processes environment and its behavior. The Address Space exposed by the server makes up a context the process data is made available in to the clients (Fig. 1). Creation of this context (Sec. IV) depends on an application domain unique Information Model.

Processes in Fig. 1 represents a class of functions responsible for getting access to business or industrial processes source of exposed information resources and services hereinafter referred to as process data.

Basing on the defined typical enterprise systems structure and requirements [25], a new architecture is proposed (Sect. IV) where an intermediate component called Process Observer is proposed. The model allows for significant reduction of the solution complexity, but the implementation of this model proves that the architecture additionally could increase robustness by adding redundancy. The example (sect. V.B), where the presented model has been used, shows that the approach can be a platform for multi-enterprise collaboration to benefit from synergy effect and macro optimization.

Sect. III, III.C and IV describe novel architectural proposal and corresponding communication algorithms allowing building robust real time distributed systems. A case study where the presented solutions have successfully been implemented is in Sect. V.

II. OPC Unified Architecture Key Features

A. Service Oriented Architecture

At the very beginning of a new solution development the question about its fundamental paradigms and architecture must be addressed. Observing continuous evolution of the ICT domain, it seems that finding a solution that will guarantee an unlimited lifetime is a real challenge. However, decoupling the solution from any base technology increases the chance of its surviving the disappearance of the base technology from the market. Fortunately, as mentioned above, there are many options on how to get applications interconnected over the Internet. Developing services and deploying them using Service Oriented Architecture (SOA) is the best way to utilize ICT systems to meet this challenge. A service differs from an object or a procedure because it is defined by messages that it exchanges with other services. SOA defines the way in which services are deployed and managed. Adopting of the SOA approach increases reuse, lowers overall cost, and improves the ability to rapidly change and evolve systems, whether old or new.

To make systems interoperable, any even brilliant idea is not enough - a data transfer technology is needed, however – when defining data exchange in context of messages – we do not need to bother about different technologies used by the participants as long as they can absorb the messages.

Today, an ideal platform for the SOA concept implementation is Web Service technologies. Web Services are a set of standards based on XML (eXtensible Markup Language) and developed by W3C (World Wide Web Consortium) [2] marked with a WS-* symbol. Because the WS-* standards are developed without any initial assumption concerning the underlying system platform they are implemented on, they therefore must precisely define what must be on the "wire".

The WS-* standards are the basic foundation for OPC UA but, using them alone, would not be enough to reach the expected data throughput performance in industrial applications. To promote scalability, the OPC UA suite of protocols, therefore, expands the WS-* standards by defining a few proprietary ones that can be used alternatively. OPC

UA messages may be encoded as an XML text or in binary format for efficiency purposes.

B. Object Oriented Information Model

OPC UA uses an object as a fundamental notion to represent data and activity of an underlying processes system. The objects are placeholders of variables, events and methods and are interconnected by references. This concept is similar to well-known object oriented programming (OOP) paradigm [9]. The OPC UA Information Model [4], [5], [10], [11] provides features such as data abstraction, encapsulation, polymorphism, and inheritance.

The OPC UA object model allows servers to provide type definitions for objects and their components. Type definitions may be abstract, and may be inherited by new types to reflect polymorphism. They may also be common or they may be system-specific. Object types may be defined by standardization organizations, vendors or end users. Each type must have a globally unique identifier that can be used to provide description of the information meaning, i.e. semantics from the defining body or organization. Using the type definitions to describe the information exposed by the server allows:

- Development against type definition;
- Unambiguous assignment of the semantics to the data expected by the client.

C. Abstraction and Mapping

Interoperability of applications can be achieved if communicating parties are able to interchange streams of bits and assign to these streams the same meaning without any ambiguity. Unfortunately, the representation of information on the wire, and communication protocols are subject to continuous evolution, if not revolution nowadays. This could be dangerous for any long term initiatives. Because it is impossible to stop the progress of technology changes, some other precautions must be taken to keep the specification alive within a long term horizon. It is achieved by clear separation of definitions provided by the specification from their actual implementation. It makes OPC UA seamlessly portable from one technology to another. Mappings defined in the specification [13] set forth how to implement an OPC UA feature using a specific technology.

D. Security

Security is the fundamental aspect of computer systems, in particular those dedicated to enterprise and process management. Especially in this kind of applications, security must be robust and effective. Security infrastructure should also be flexible enough to support a variety of security policies required by different organizations. OPC UA may be deployed in diverse environments; from clients and servers residing on the same hosts, throughout hosts located on the same operation network protected by the security boundary protections that separate the operation network from external connections, up to applications running in global environments using public network infrastructure. Depending on the environment and application requirements, the communication services must provide different protections to make the solution secure [14].

OPC UA Security is concerned with the authentication of clients and servers, the authorization of users, the integrity and confidentiality of their communications and the auditing of client server interactions. To meet this goal, security is integrated into all aspects of the design and implementation of OPC UA servers and clients.

OPC UA relies upon the site cyber security management system to protect confidentiality on the network and system infrastructure, and utilizes the public key infrastructure to manage keys used for symmetric and asymmetric encryption [15]. OPC UA uses symmetric and asymmetric encryption to protect confidentiality as a security objective, as well symmetric and asymmetric signatures to address integrity as a security objective.

E. Profiles

OPC UA is designed to support integration of wide range of servers, from plant-floor control devices to enterprise management and operation systems. All of them are characterized by a variety of performances, execution platforms and functional capabilities. Therefore, OPC UA defines a comprehensive set of capabilities servers may implement a subset of. These subsets are referred to as *Profiles*, and servers may claim conformance to them.

F. Robustness

Because it is to be used in the production environment including real-time process control applications, OPC UA is designed especially to provide robustness of the remote access to the underlying process data. OPC UA provides mechanisms for clients to quickly detect and recover from communication failures associated with transfers without having to wait for long timeouts provided by the underlying protocols [16].

III. Information Model

A. Concept

The primary objective of the OPC UA server is to expose information resources and services, which then can be used by clients to manage an underlying real-time process or the entire enterprise as a large whole with the main challenge of integrating systems and management resources into one homogenous environment. Information describes the state and behavior of the processes and the server must be able to transfer it in both directions. The main challenge of the OPC UA Information Model is to support this transfer by a unique and transparent means in spite of the process complexity and roles of clients in the enterprise management hierarchy.

Information is an abstract knowledge; therefore it cannot be directly processed by physical machines. To make information capable of being processed, it must be represented as the binary data. To define the relationship between information and binary data on the one-to-one basis, syntax and semantics are needed. Syntax defines rules of the vocabulary usage, and semantics maps valid bits pattern to the associated piece of information.

An Information Model for OPC Unified Architecture is such a collection of vocabulary, syntax and semantics. This collection plays a role similar to high level programming languages that describe data structures and an algorithm to be executed by the processor.

Information exposed by the OPC UA server may be complex. Clients may, therefore, want to obtain the information definition. Generally speaking, to select a particular target piece of information we have two options: random access or browsing. Random access requires that the target entity must have been assigned a globally unique address and the clients must know it in advance. We call them well-known addresses. The browsing approach means that the clients walk down available paths that build up the structure of information. For example hypertext document containing URL’s locating recursively hypertext documents and other resources.

It seems that, in spite of the access method, we have to assign an address to all of the accessible items in the representation of the information structure. Therefore we call the collection of these items the Address Space [4], [5], [12]. This atomic addressable item is called a node. Each node is a collection of predefined set of attributes that have values accessible locally in context of the node. To represent information about the internal structure, nodes are interconnected by references.

Accessing information by clients is the first aspect of controlling the data stream between the clients and the underlying process environment of the server. Another one is creating and maintaining the Address Space in real-time.

To create the Address Space, we need to instantiate nodes and interconnect them by references. Instantiating nodes operation requires assigning appropriate values to attributes and adding references. To make information internally consistent as a large whole, we need rules governing the creation and modification processes. The Information Model implies these rules using the following two concepts:

- ***NodeClass*** – as a formal description of the node defining the allowed attributes and references;
- ***Type*** – as a formal description of the node defining the allowed attributes and references values.

For the client to understand the Information Model, it must be predefined or exposed.

Available NodeClasses are predefined, i.e. the specification provides a strictly defined non-extensible set of NodeClasses. Each one is assigned a dedicated function, e.g. *Variable* NodeClass defines nodes that provide a value, and *Method* NodeClass represents a function.

Like the NodeClass concept, the specification provides a set of predefined types, which is extensible. According to the above rule, all not predefined types must be exposed in the Address Space. To expose predefined and proprietary type definitions in the Address Space, there are dedicated NodeClasses, namely *ObjectType*, *VariableType* and *ReferenceType*. For example, nodes of the *VariableType* NodeClass provide clients with definitions of types derived from the *BaseVariableType* that is a base type for all variables. The main role of the types represented by the above NodeClasses is to provide a description of the Address Space structure and to allow clients to use this knowledge to navigate to desired information resources (represented by the *Variable* nodes) and services (represented by the *Method* nodes) in the Address Space exposed by the OPC UA server.

DataType NodeClass is also dedicated to describe types. In this case, the represented types have a special mission, because they describe underlying process data that client has access to using a connection to the OPC UA server. For example, a node of the DataType can provide information to clients that the data has a numeric value and the clients reading it can use this knowledge to interpret and process the obtained value.

Types are called metadata since they describe the data structure (context) not the actual data values.

Even though the OPC UA specification contains a rich set of predefined types, the type concept allows designers to freely define types according to the application needs. New types are derived from the existing ones. The derived types inherit all features of the base types but can include modifications to make the new types more appropriate for information that is to be represented.

The Address Space concept based on types can be a foundation for exposing any information that is required. Clients understand the Address Space concept and have a browse service to navigate through the Address Space. Since browsing is based on the incremental and relative passage among nodes it is apparent that each path must have a defined entry point, so the question as to “where to start" must be addressed. To meet this requirement, the Address Space must have a predefined template containing well defined nodes that can be used as anchors from which a client can start browsing the Address Space content. Thus to design an Address Space and define new types, they must be derived from the existing ones. At the very beginning the only existing types are the standard ones defined by the specification. The available standard types are briefly described in the Section III.B.

B. Standard Information Model

The primary objective of the OPC UA Address Space is to provide a standard way for servers to represent objects to the clients. The Object NodeClass is used to define objects. Each object in the Address Space has an assigned *ObjectType*. The specification has provided a

BaseObjectType from which all other *ObjectTypes* shall either directly or indirectly inherit.

Variable NodeClass is dedicated to provide a value to the clients. To define a *Variable* two types must be provided:

• *VariableType*: describes the type of a variable. Each Variable node has the *HasTypeDefinition* reference to its type definition.

• *DataType*: describes the type of the value of the variable. It is assigned to the *DataType* attribute.

The type of data provided by the *Variable* Value attribute is defined by the associated *DataType*. *DataType* is pointed out by the *DataType* attribute of the *Variable* and *VariableType* nodes. In many cases, the value of the *DataType* attribute will be well-known to clients and servers. Well-known data types allow clients to use random addressing and interpret values without having to read the type description from the server.

To some standard data types – called built-in types - special rules apply. Built-in data types are a fixed set that should be known to all OPC UA products. Examples of built-in data types are *Int32* and *Double*. Most of the built-in data types are similar to those in programming languages.

Process data could be complex. *Structure* is an abstract data type defined as the base for all structured types. All complex data, if not defined in the specification explicitly as primitive, are created by defining of new types derived from the *Structure*.

Reference types are used to create interconnections between nodes. They are not instantiated, i.e. a NodeClass representing a reference is not defined. Instead of instantiating the references, they are added to a collection associated with each node. NodeClass of the node and its type decide what references are allowed to be added to this collection.

The base of all references is an abstract *References* type. There is no semantics associated with it. There are two disjoint sets of standard references:

• *HierarchicalReferences*

• *NonHierarchicalReferences*

This distinction reflects two fundamental relationship categories that can be generally distinguished: the association and the dependency. Associations are used to build information architecture – nodes hierarchy - that can be discovered by the clients using the browsing mechanism. An example of the association is the “parent/child” relationship. In this case it can be said that the target belongs to the source. A dependency of a source element (called the client) on a target element (called the supplier) indicates that the source element uses or depends on the target element. An example of dependency is the variable and the variable type relationship. In this case the target describes the source.

C. Extending OPC UA Information Model

The standard OPC UA Information Model is expandable. For example, in 2008 the OPC Foundation announced support for Analyzer Devices Integration into the OPC Unified Architecture and created a working group composed of end users and vendors with its main goal to develop a common method for data exchange and an analyzer data model for process and laboratory analyzers. In 2009 the OPC Unified Architecture Companion Specification for Analyzer Devices was released [17]. To prove the concept a reference implementation has been developed containing ADI compliant server and simple client using the Software Development Kid released by the OPC Foundation [17].

It is an example of how OPC UA standard Information Model can be expanded by a selected domain application. Standardized expandability of the metadata used to provide a context of underling process data is key requirements of the presented Object Oriented Internet concept.

In this example, the model described in the specification is intended to provide a unified view of analyzers irrespective of the underlying device. This Information Model is also referred to as the ADI Information Model. As it was mentioned, analyzers can be further refined into various groups, but the specification defines an Information Model that can be applied to all the groups of analyzers.

The ADI Information Model is located above the DI Information Model [18] [19]. It means that the ADI model refers to definitions provided by the DI model, but the reverse is not true. To expand the ADI Information Model, the additional layers shall be provided.

IV. Information Model Deployment

The OPC UA is a standard that allows clients to get access to the server underling processes. To meet this objective, each server instantiates and maintains an Address Space that is a collection of data to be exposed to clients. The OPC UA Address Space consists of nodes and references. The main role of the nodes is to expose the underlying processes state and behavior as a selectable, well-defined piece of information.

To create the Address Space the OPC UA servers must instantiate all nodes and interconnect them by means of references.

As it was stated previously, typical implementation architecture consists of OPC UA Clients, which are connected to an OPC UA server (Fig. 1). To get access to underlying *Processes* data a generic client does not need to have any awareness of the Information Model used to create the Address Space exposed by the sever in advance. However, in the production environment, the Information Model (types) knowledge may be useful to offer additional functions, like dedicated data processing, customized control panels or predefined structure of the database tables. Types knowledge also simplifies configuration of the clients, because all of the items composing the complex process information can be accessed simultaneously – they can have one single address – identifier.

To implement the Address Space two questions must be addressed [20]:

- How to couple the nodes bi-directionally with the underling process data sources?
- How to create and maintain it?

Using the instantiated nodes by means of a well-defined set of services [8] (*OPC UA interface*), clients get access to data representing a selected part of the underlying processes environment. Nodes are divided into classes. The *Variable* class is used to represent the values – has the Value attribute. To be used as the process state representation, the value of the Value attribute must be bound with a real data source, e.g. an analog signal or a database item. The *Method* class represents a function that can be called by the clients connected to the server. In this case the real-time process bindings are responsible for conveying the parameter values, invoking the represented series of operations and returning the execution result. In Fig. 1 both classes are the main building blocks of the architecture that allow the server to couple the exposed Address Space with the current state and behavior of the underlying *Processes*.

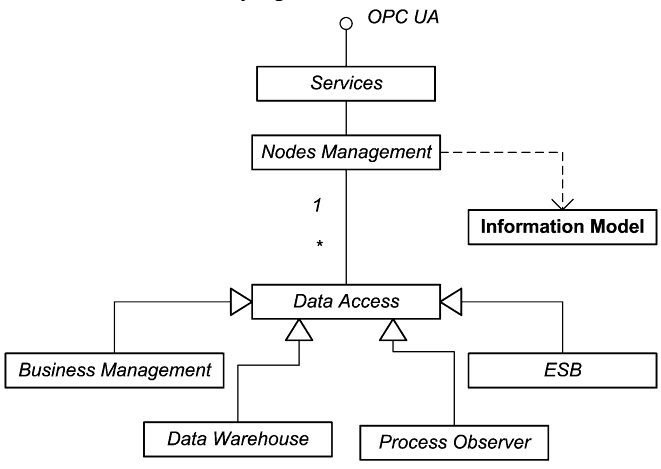

Fig. 2 Process Observer archetype diagram

The technique of binding the nodes with process data is vendor specific, but it must be transparent to the *Clients*. Nodes management functionality on the *Client* part is standardized by the OPC UA Service Model [8] (*OPC UA* interface - as a set of services depicted in Fig. 2). Access to the values representing the current process state is provided by the Read/Write functions. The client can also be informed about changes of the process state using "data change" notifications. Invoke and event notification functionalities allow clients to use the *Methods*.

In Fig. 2 the proposed internal diagram of the *OPC UA Server* package is shown. To implement the functionality presented above, three coupled function classes shall be distinguished:

- *Services*
- *Nodes Management*
- *Data Access*

The diagram in Figure 2 shows the associations between the above function classes. In this architecture the *Data Access* is responsible for transferring process data up and down. The *Nodes Management* function class couples the *Processes* data with appropriate nodes instances representing underling process metadata and provides a homogenous picture to *Services* that finally exposes it to all connected clients.

Real-time process data can be obtained from any underling process, i.e. file system, database, device or even large scale highly distributed automation system. For embedded applications it may directly use internal controller registers of the device. The *Data Access* function class is able to obtain data using the random access or underlying communication infrastructure and vendor-specific protocols.

To create the Address Space - i.e. to instantiate all nodes and interconnect them by means of references - the *Nodes Management* function class uses a predesigned static *Information Context* (dependent on the *Information Model* – not shown in Fig. 2) providing a detailed description of all the nodes, including their attributes and references. Static means that the model is predefined for the selected environment, but it does not mean that the exposed Address Space is static. In this approach, nodes can be instantiated and linked dynamically, however this operations must conform to the model definition. Dynamic behavior of the Address Space can be controlled by the connected clients using services or by the current state of the process.

Before nodes making up the Address Space can be instantiated by the server, this Address Space must be designed first. Model designing is a process aimed at designing Information Model as a set of nodes and their associations and, next, creating the *Information Context* as its representation in a format appropriate for the implementation of the *Nodes Management* function class. Depending on the OPC UA server implementation, the Information Model representation and support for the modeling process varies. The main challenge that must be faced up is how to prepare *Information Context* seamlessly without programming. The designing process can be supported by the Address Space Model Designer tool [19], [21], [22] that is intended to help architects, engineers and developers accomplish *Information Context* preparation. Using the tool it could be similar to preparation of a hypertext document.

The tool developed by a team leaded by the author is very useful to make the publication of the process data in the context of metadata straightforward and without programming, but it is only proof of the solution concept. To promote the Object Oriented Internet concept in a wider scope more research is required with the goal to define a formal, widely accepted representation of Information Model, semantic validation methods, generation of the Address Space and custom complex data serialization to leverage the deliverables to the designers, developers, end user, etc. and to integrate them into other applications. It is proposed to carry on this research work as a common effort using an open source project [23] as the research workspace, which offers basic work framework and very convenient project management utilities available on the well-known GitHub platform. In other words, applications interoperability is yet granted by the OPC UA standard, the next step is to work out unification of the designing/deploying methods and supporting tools to make people cooperation possible and finally the Object Oriented Internet a real option.

V. Accessing Information Resources

A. Architecture

According to the definition the Internet is expected to provide access to information resources and services hereinafter referred to as data sources. For the architecture proposed in Fig. 2 the *Data Access* functional class is responsible for fulfilling this job. Because the underling information resources and services are to be exposed in the context of the Address Space the functional class *Nodes Management* is responsible for binding the underlining data sources with appropriate variable and method nodes embedded in the Address Space. These variables and methods are accessible by the remote clients using the standard *OPC UA* interface provided by the *Services* functional class

In the proposed approach there are no limits regarding possible data sources that can be coupled by the *Data Access* with *Nodes Management*. Generally, three classes of data sources can be distinguished:

- Data representing the current state and behavior of the underling real-time industrial processes;
- Archival data representing the behavior of the underling processes in time;
- Current processed data obtained from business supporting applications and other connectivity standards.

A typical example of the real-time physical processes is the industrial automation process control system. The process control contains digital plant floor devices responsible for measurements, controlling and condition monitoring of the real-time process locally. Usually, the predominant function in this case is accomplished using PLC (Programmable Logic Controller) or DCS (Distributed Control System) class products. In a distributed process, one can distinguish autonomous islands of automation, whose cooperation has to be harmonized by a supervisory system that is responsible for controlling the process as a larger whole.

To get access to the plant floor devices and couple them to the *Nodes Management* functional class underlying proprietary communication links must be instantiated. Although from the design point of view this communication can be considered transparent, its availability and reliability is crucial for the final result. Assuming transparency, it simplifies the problem to a great extent, provided that the assumption is valid.

To instantiate a link we need a medium. To transfer data over the medium, we have to use selected protocols controlling access to the medium and responsible for robust data transfer. Additionally, the protocol and medium often limit the bandwidth and medium access. Any of these requirements can cause that the above assumption and, in consequence, this approach becomes unreal. Therefore, we need to look for a compromise between an unacceptable complexity and unreal assumption.

To make the plant floor device interoperable with the *Data Access* functional class, both have to use the same vendor-specific or standard-compliant protocol. Relying on vendor-specific solutions limits future solution expandability. Generally, it is, therefore, not recommended and vendors usually offer a standard protocol for plant floor devices. Unfortunately, there are hundreds of “open standards” defined in the automation marketplace.

For the highly distributed process control systems (like smart grid, smart heat distribution networks, etc.) assuming that the whole system uses one common communication medium is not feasible [24].

Lack of common medium coverage of the whole area that the controlled process is dispersed over requires engaging simultaneously many communication infrastructures, and dealing with a multidimensional communication network. The main advantage of using many infrastructures is the possibility of improving robustness of the system by providing communication redundancy [24] in overlapping areas provided that it is possible to utilize them alternatively.

To transfer the data, we need a medium, but to use the medium, we need to engage an infrastructure: a technology (Internet, GSM, satellite, ISDN, etc.) governed by technical standards and an organization governed by regulations, procedures, practice, etc. A platform optimal today may be useless for future because technology is progressing rapidly and economical standing of organizations may fluctuate.

To address all issues described above the *Data Access* functional class has to be expanded to employ appropriate communication functionality. It is proposed to implement Process Observer architecture presented in next section as an extension to manage the underling communication infrastructure and transfer process data in real-time in a systematic manner.

The next example of the underlying data source is a repository containing manufacturing process information, like data base or even a data warehouse. Data warehouses are designed to facilitate reporting and analysis. This kind of application focuses on data retrieving and analysis, to extract, transform and load data.

To interconnect with an archival processes data repository the *Data Warehouse* extension of the *Data Access* functional class has been added to the proposed architecture in Fig. 2.

Data without context has no meaning, hence metadata is critical to a data strategy. Designing a data binding mechanism both data and metadata must be considered. The *Data Warehouse* extension is responsible for providing an appropriate translation (according to the OPC UA Information Model) between metadata of the underling process data and the context of the server Address Space where the data is made available to the clients. Simplicity of this relationship is crucial to the business, because metadata exposed by the OPC UA server and metadata describing the underling repository content must be designed on the basis of the same semantics rules. In the design process, where the metadata originates and how to synchronize it should be addressed first.

Usually, apart from the historical data access mechanism OPC UA clients use real-time data access subscribing to current data changes. Therefore, the *Data Warehouse* must be smart enough to provide updates by following the repository modifications.

Business Intelligent (BI) applications are a keystone for macro optimization at the enterprise level because they provide an insight into data, which allows analysts and executives to easily uncover patterns and abnormalities in the business [26]. In the late 90s organizations also implemented enterprise resource planning (ERP) and customer relationship management (CRM) software that can be candidates for the next data source. There are many other business level applications (BLA) processing information and providing results that can be published by the OPC UA server in the Internet using the Object Oriented approach.

Usually a data warehouse (DW) is a central part of today's BLA and real-time process control deployment and hence the archival data may be available also indirectly via the *Business Management* or *Process Observer* data bindings.

The cornerstone of a successful BI application is its capability to provide business users fast and easy access to data for analysis. Online analytical processing (OLAP) tools are a foundation of BI application. In the discussed architecture, another option is to distribute BI application over the Internet and couple the OPC server exposing OLAP functionality to the remote applications.

The implementation of the above described functional classes requires a dedicated link used to manage data transfer. Data transfer for the most popular database management systems are governed using Structured Query Language [27]. It is a language rather than connectivity, but can be used together with widely used vendor services to standardize the data access and simplify the *Data Warehouse* implementation.

To make the enterprise more and more beneficial, the applications supporting automation and business processes have to be integrated. From integration, we should expect additional performance improvement as a result of synergy and real-time macro optimization effects. Enterprise Service Bus (ESB) [28] is a standard-based concept and hence it is well suited for integration projects. The ESB provides a highly distributed, event-driven Service Oriented Architecture (SOA) that combines Message Oriented Middleware (MOM), web services, XML data transformation and intelligent routing based on content.

Using ESB as a foundation for the applications integration allows for implementation of the OPC UA server data bindings by interconnecting of the *Data Access* with this bus. In the architecture presented in Figure 2 this role is fulfilled by the ESB extension of the *Data Access*.

B. Process Observer Architecture

The Process Observer architecture described in [24], [25] is proposed to be used as a consistent sole representation of a distributed real-time process (Fig. 3). It is an extension of the *Data Access* class (Fig. 2).

In the presented architecture the following classes are distinguished:

- *Cache* is a collection of the latest values of the process data.
- *Controller* holds the plant-floor device data description.
- *Channel* is used to represent independent communication threads conducted simultaneously to each other.
- *Segment* represents a single communication path and is responsible for managing communication resources and data transfer from a group of devices that is to be accessed using the same transport connection.
- *DataProvider* is responsible for providing a stream of data to the *Segment*.
- The *Pipe* is a collection of *Ports*, where only one of them is active at any time.
- The *Port* represents a bidirectional device data streaming functionality.

The description represented by the *Controller* is used to schedule in time all read operations to update the data in the *Cache*.

Usually, lower layer communication requires multidimensional networks. The *Channel* class allows creation as many simultaneous communication paths as it is necessary. To assure mutually exclusive access to common resources, the *Channel* activates only one *Segment* at any time.

To provide a consistent process data from multidimensional network environment and using custom protocols the proposed solution enables to create many *DataProviders* instances by a *Channel* and use them by a data transfer algorithm realized by the *Segment*. Each *Segment* can use only one *DataProvider*, but one *DataProvider* can be used by many *Segments* associated with the same channel.

To provide polymorphism for the environment specific needs, the *DataProvider* is located outside the main software package and inherits an interface ensuring flexible management of the communication medium and transfer of the process data. This solution makes it possible to keep the core software unchanged and adapt a Software Development Kit to the specific needs. In this scenario, the late binding approach is supported. Late binding is useful if it is required to replace a part of software package without recompiling of the code base. In this case, a variety of protocols might be supported with a separate module for each protocol specification. A declarative configuration can be used to tell the application to use a specific module at runtime. Another scenario where late binding can be useful is to enable users of the system to provide their own customization through a plug-in. Again, the system can be instructed to use a specific customization by using a configuration setting.

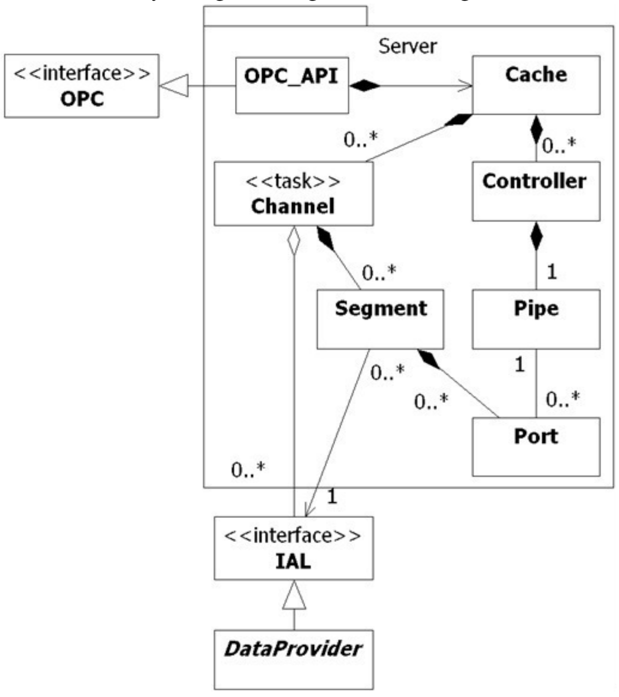

Fig. 3 Process Observer Architecture

In a real environment, apart from accessing underlying process data, monitoring and management of the recourses and communication infrastructure are often of the same importance. Monitor class (Fig. 4) represent this functionality. To commence factory tests or provide a state observer a simulation environment is required. Simulator class is responsible to provide the simulated data and can be used in place of the Protocol class for testing purpose. This concept makes it possible to publish all of the mentioned types of information in the same way using the defined interface and late binding approach.

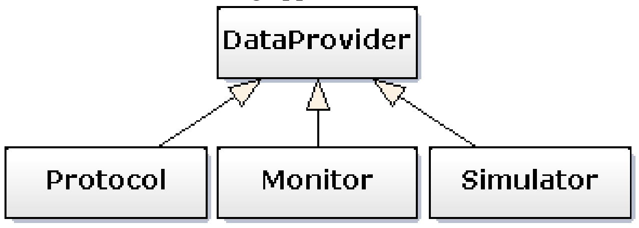

Fig. 4 DataProvider functions

In the proposed model (Fig. 3), the *Pipe* concept is used to assure redundancy. After detecting a failure of the active path, another *Port* belonging to the same pipe is activated immediately. *Segment* uses only active *Ports* and, therefore, the data is transferred over the network once only. The *Pipe* checks availability of non-active paths periodically. Using paths redundancy additional spare plant-floor devices can be used seamlessly as the next level of redundancy.

The main job of the communication software is to make `best effort' to keep the process data fresh and allow clients to access the data randomly. From the communication point of view, two independent communication environments can be distinguished (Fig. 1):

- *Processes* connecting plant-floor devices to an intermediate component (server);
- OPC UA Interface connecting the intermediate component (server) to *OPC UA Clients*.

Because both are used to transfer the process data, it is vital how these data transfer processes are related to one another. To finally design an appropriate sampling scheduling mechanism on the process side, we need to take into consideration:

- Needs of the *OPC UA Clients*.
- Current real-time process state;
- Current communication path load and its throughput;

All of them can change in time and, therefore, it is proposed to implement the following two unique closely coupled costs saving algorithms providing process data just in time and preserving communication bottlenecks:

- *Adaptive Sampling Algorithm* (ASA): responsible for adjusting the plant-floor devices sampling rate according to the current process state.
- *Optimal Transfer Algorithm* (OTA): responsible for minimizing the difference between requirements of client individual process data update rate and current sampling rate of a process control devices;

To minimize the data transfer costs, the sampling rate is adapted to the current process needs.

The Process Observer architecture is widely used as a communication engine in highly distributed systems. The supervisory control of a metropolitan heating system located in the city of Lodz – Poland [24],[25] is an example. The heat distribution network of Lodz (750k citizens) is supplied from heat and power plants with total thermal output of 2.5GW. It consists of:

- 3 heat and power plants,
- 2 backbone pumping stations,
- Hundreds of backbone heat chambers
- Thousands of local distribution points.

Their optimal utilization requires a control system to allow working on common supplying areas. As the system is distributed geographically (about 800km of pipes), safe communication between nodes (automation islands) is very important. An implementation [25] of Process Observer

Architecture proves the concept in highly distributed application.

The architecture presented in this section has been already integrated with the OPC UA services, but further research is required to integrate it with Information Model designing methodology consistently. The main challenge is how to support custom complex data. The complex data must:

- be factored using components gathered from the underling process,
- follow the DataType declarations in the Information Model,
- be transparently serialized over the wire.

VI. Conclusion

Nowadays, in such a fiercely competitive environment, modern manufacturing and transportation automation systems have to be involved. Such systems usually consist of numerous different ICT systems located at business and process management levels. They are frequently dispersed geographically in multi-division organizations.

The Internet is a globally available communication infrastructure that makes it the first and practically the only candidate to be used as a platform to build a universal solution for the above objectives and even to integrate systems belonging to cooperating organization groups to benefit from the synergy effect and global optimization.

The freely expandable Object Oriented archetype and its practical implementation presented in the article prove that the above goal can be achieved and the final solution offers the following features:

- It provides application to application robust interoperability over the Internet;
- On the server side, it makes it possible to freely publish and update information and services in a contextual (semantics aware) environment;
- On the client side, it makes it possible to get a description, discover and finally get access to the requested information and services;
- Information resources and services exposed by the server that represent the state and behavior of the underlying processes allow clients to manage and control them over the Internet/Intranet;
- Client and server software can be offered by independent vendors as generic off-the-shelf products;
- The products can be tested for interoperability independently of each other.

To accomplish this it is proposed to distribute publisher (server) main tasks to three functional classes:

- A predefined fixed set of services based on the SOA concept conforming to the OPC Unified Architecture specification;
- Information context management using the object oriented programming paradigm;
- A pluggable proprietary data binding mechanism.

Development of generic communication software that can be interoperable requires specification compliance testing. It is proposed that OPC Unified Architecture, a new emerging industrial standard that fulfils requirements derived from this architecture should be used because it provides a definition of an appropriate: set of services and Information Model concept dedicated to formally describe the Address Space – context for the exposed information resources and services. It has wide industrial support and a well defined compliance test procedure governed by the OPC Foundation.

Available reference applications and commercial products pointed out in the article prove that the data binding concept can be successfully implemented as dedicated application-dependent pluggable components. The components must be able to couple the proprietary underling data access mechanism with the server mechanism managing the context where the data is embedded and made available to connected clients.

The approach to represent the underlying data processing environment as presented in the paper can be used for countless applications, from exposing the representation of measurement devices to building multi-enterprise management and remote process control systems. Smart networks, i.e. smart grid, smart district heat distribution networks, utility distribution, oil distribution, railways, etc. are an example of applications like that.

It is worth nothing that to promote the Object Oriented Internet concept in a wider scope more research is required with the goal to define a formal, widely accepted representation of Information Model, semantic validation methods, generation of the Address Space and custom complex data serialization to leverage the deliverables to the designers, developers, end user, etc. and to integrate them into other applications. In other words, applications interoperability is yet granted by the OPC UA standard, the next step is to work out unification of the designing/deploying methods and supporting tools to make people cooperation in this respect possible and finally the Object Oriented Interned a real option.

It is proposed to carry on this research work as a community effort using the open source project [23] as the research workspace on the well-known GitHub platform.

References

[1] Network Protocols Handbook, Javvin Press, 2007;

[2] http://www.w3.org/ The World Wide Web Consortium (W3C), 2015.

[3] M. Postol, UA Specifications, in J. Lange, F. Iwanitz, T. J. Burke, OPC – from Data Access to Unified Architecture, Hüthig Fachverlag, 2010.

[4] http://www.commsvr.com/UAModelDesigner/ OPC Unified Architecture e-book, 2015.

[5] W. Mahnke, S. Helmut L., M. Damm. OPC Unified Architecture. Berlin: Springer, 2009.

[6] OPC UA Specification: Part 1 – Concepts, Version 1.0 or later. OPC Foundation, 2009.

[7] https://opcfoundation.org/, The OPC Foundation - The Interoperability Standard for Industrial Automation, 2015.

[8] OPC UA Specification: Part 4 – Services, Version 1.0 or later. OPC Foundation, 2009.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley 1995

[10] OPC UA Specification: Part 5 – Information Model, Version 1.0 or later. OPC Foundation, 2009.

[11] M. Postol, Information model, in J. Lange, F. Iwanitz, T. J. Burke, OPC – from Data Access to Unified Architecture, Hüthig Fachverlag, 2010.

[12] OPC UA Specification: Part 3 – Address Space Model, Version 1.0 or later. OPC Foundation, 2009.

[13] OPC UA Specification: Part 6 – Mappings, Version 1.0 or later. OPC Foundation, 2009.

[14] OPC UA Specification: Part 2 – Security, Version 1.0 or later. OPC Foundation, 2009.

[15] C. Adams, S. Lloyd: Understanding PKI: Concepts, Standards, and Deployment Considerations, Second Edition, Addison-Wesley Professional, 2002;

[16] http://www.commsvr.com/Products/OPCUA/CommServerUA.aspx- CommServerUA: Redundant, Multi-protocol, Multi-channel OPC UA Server For Highly Distributed Systems, 2015.

[17] OPC Unified Architecture Companion Specification for Analyser Devices. OPC Foundation, 2009.

[18] OPC Unified Architecture Companion Specification for Devices. OPC Foundation, 2009.

[19] M. Postol, OPC UA Information Model Deployment, CAS, 2015, http://goo.gl/HqYjvy

[20] M. Postol, Design and Modelling of the Address Space, in J. Lange, F. Iwanitz, T. J. Burke, OPC – from Data Access to Unified Architecture, Hüthig Fachverlag, 2010.

[21] http://www.commsvr.com/Products/UAModelDesigner.aspx – OPC UA Address Space Model Designer software, 2010

[22] M. Postol, UA Address Space Model Designer, in J. Lange, F. Iwanitz, T. J. Burke, OPC – from Data Access to Unified Architecture, Hüthig Fachverlag, 2010.

[23] OPC UA Object Oriented Internet, Opc-ua-ooi open source project on GitHub, http://mpostol.github.io/OPC-UA-OOI/, 2015

[24] M. Postol, Real-Time Communication for Large Scale Distributed Control Systems; International Multiconference on Computer Science and Information Technology; Wisła (2007) PIPS, pp. 849–859 ISSN 1896-7094

[25] M. Postol, Large scale distributed process and business management integration; 14th International Congress of Cybernetics and Systems of World Organization of Systems and Cybernetics, Wroclaw (2008), pp. 632-642, ISBN 978-83-7493-400-8

[26] W. A. Giovinazzo: Internet-Enabled Business Intelligence, Prentice Hall; 2002

[27] T. Connolly, Database Systems (2nd ed.). Addison-Wesley, 1999.

[28] David A Chappell: Enterprise Service Bus, O'Reilly Media, Inc., 2004;

[29] J. Lange, F. Iwanitz, T. J. Burke, OPC – from Data Access to Unified Architecture, Hüthig Fachverlag, 2010.