Web Services in Building Automation: Mapping KNX to oBIX

Matthias Neugschwandtner, Georg Neugschwandtner, and Wolfgang Kastner

Abstract— **Web services are a key technology for enabling interoperable machine-to-machine interaction over a network. They also lend themselves excellently to the integration of automation and IT systems. This paper discusses oBIX, a new standard for representing and accessing building automation (BA) data via Web services. It is shown how access to a BA system that follows the KNX protocol standard can be faithfully represented by way of oBIX entities. A prototype implementation of such a gateway is presented. The extensible nature of the oBIX data model is leveraged by using it to express the required KNX–oBIX mapping information as well. This approach allows a particularly clear and efficient gateway design.**

I. Introduction

In a typical present-day building automation system, building services are provided by domain specific autonomous subsystems (e.g., heating, ventilation, and air conditioning; lighting and blinds; security; and fire safety). A building management system (BMS) provides global management functions such as integrated visualization, high level control, or unified handling of technical alarms. For this purpose, the subsystems must expose their functionality in a way suitable for interaction with the BMS.

Obviously it is desirable that subsystems expose datapoints in a common, preferably open format. At present, OPC [9] is very popular for this purpose in both process and building automation. However, OPC is restricted to Windows platforms in its classic design incarnation. Web Services (WS) technology allows a solution without this drawback.

Using this technology, self-contained software applications running on disparate platforms and operating systems can collaborate over the Internet as well as corporate intranets. Web Services are totally platform independent – they can be implemented using any programming language and run on any hardware platform or operating system. This maximizes interoperability and greatly improves the reusability of the services they provide. Central to Web services are the use of XML for data representation and a standardized resource access protocol (typically SOAP or plain HTTP).

Web Services follow a modular concept. This allows the use of off-the-shelf standards for transmission, eventing, discovery, security and many more. Web Services are also the most popular way to implement service oriented architectures (SOA). Web Services in such an architecture are built to be self-contained and loosely coupled. This means that they can be flexibly arranged into complex applications. So, WS that provide interoperable access to datapoints as a first step lay an excellent foundation for high-level business services (e.g., load management) to be created. The application of SOA and WS to automation systems is discussed in detail in [5].

A drawback of Web Services is the additional overhead introduced by the use of XML encoded messages, especially if small amounts of data are transmitted frequently. In addition, receiving events from the server requires the client to either poll periodically or implement an entire WS server itself (when following the WS-Eventing standard). At the field level, where fast response and resource efficiency are crucial, this would be a considerable disadvantage. However, at the management level this is less an issue.

M. Neugschwandtner, G. Neugschwandtner, and W. Kastner ({mneug, gn, k}@auto.tuwien.ac.at) are with the Automation Systems Group, Institute of Computer Aided Automation, Vienna University of Technology; Treitlstrasse 1-3, 1040 Vienna, Austria.

II. The oBIX standard

The commitment to use WS technology determines low-level aspects of data representation and transmission. Still, the application domain remains to be modelled. Today, three standards hold particular promises for bringing the world of WS to BA. First, the forthcoming OPC UA (Unified Architecture) is based on WS. Second, BACnet was extended in this regard (BACnet/WS: Building Automation and Control Network – Web Services) [1]. Third, the oBIX (Open Building Information Exchange) initiative (hosted by OASIS) added a challenger to these established standards [7]. All three provide historical data access and event and alarm management besides a point abstraction. Both the BACnet/WS and oBIX standards, which have been developed with special regard to building automation, are freely available to the general public. oBIX is of particular interest due to its extensible data model.

A. Data model

oBIX provides a flexible object model to describe the data and operations available on the server. In oBIX everything is an object: objects are also used to describe data types (classes) and operations (method signatures). The flexibility of oBIX is based on the possibility to custom define any kind of object. Both subtyping (*is-a*) and composition (*has-a*) are supported.

oBIX follows what is called a RESTful approach ("Representational State Transfer"), a resource centric architectural style for Web Services. Central concepts of the RESTful approach are resources that share a uniform interface and a highly restricted set of operations on these resources. This approach mimics the World Wide Web where only the commands GET, PUT and POST are used to access countless resources.

The same is true for oBIX services: only three network request types are defined at the WS level. The first two of them are `read` (applicable to any object) and `write` (for writable objects). For operations beyond basic "get" and "set," custom operations can be defined on the oBIX level, just as any other custom object. For invoking these operations a third base network request type `invoke` is provided. In addition

to SOAP, its RESTful approach also allows oBIX to easily support a plain HTTP protocol binding, needing nothing more than HTTP GET, PUT and POST.

Naming in oBIX is realized via two different, complementary concepts. First, every object can have a "name." It is used to identify a sub object within a composite object (for example, consider an object with two string members or two operations – their name tells them from each other). Second, every object can be assigned a URI (an "href"). This is necessary whenever an object is to be referenced from the outside. To apply any network request (read, write, or invoke) to an object, this reference is required. No higher-level semantics are associated with the URI namespace.

B. Predefined objects

The root object specifies a number of mandatory attributes like an object's name, which are inherited by all other objects. Like in every data model, various object types to hold primitive values are defined. These are Booleans, integer and floating point numbers, enumerations, strings, points in time and time spans. oBIX base object types also cover a number of universally applicable concepts: lists and feeds (containers with either static content or event queue semantics), errors, references to other objects, and operations. Custom classes can be derived from any object type. Multiple inheritance is supported. Every custom-built class can be handled by any oBIX client. oBIX provides a very flexible SI-based system for describing engineering units that is also based on objects. Furthermore, the oBIX standard library defines special purpose classes encapsulating server functionality.

The Lobby object has a well-known location on the server and serves as its entry point. Besides providing information about the server through the About object, it offers services to reduce protocol overhead. First, it provides a special batch operation which accepts an entire batch of requests at once. Second, clients can register objects with the watch service. Every time the client later polls this watch, the server will return a feed of events (value changes) which have happened since the last poll operation. This also ensures that no value changes are lost, independent of the polling cycle.

Points are classes which are used to flag primitive values as coming from the automation system. Read only points are effectively an empty class (acting merely as a semantic marker); writable points additionally specify an operation for altering the corresponding value.

Historical trends can be represented via the oBIX History Record, which groups a point value and a time stamp. The History object consists of a list of history records and methods to query them. Query filters can be specified and extended to a rollup calculation for, e.g., average values.

Eventually, oBIX defines a normalized model to query, watch and acknowledge alarms. An oBIX server supplies feeds of alarm objects. Every time the server detects that the value of an object meets a predefined alarm condition, an alarm object is added to the feed. This object contains a timestamp and points to its source. Alarms can be stateful (i.e., the point in time when the source returned to normal is recorded) and they can record if (and by whom) the alarm was acknowledged.

C. XML representation

Base object types are directly mapped to individual XML elements. For example, if an object is an integer or is derived from an integer, it is rendered as `<int/>`. Any other objects, which are not derived from a base object type, are all rendered using the standard `<obj/>` element. In this case, the "is"-attribute specifies the class of an object.

A similar distinction exists regarding how the attributes of an object are rendered. Base attributes, e.g., the name, are mapped to XML attributes – called "facets." Any individually added attributes are represented as sub objects nested within the opening and closing tags of their parent object.

Methods are, on the one hand, the normal network request types on the WS level – any object can be read, written, or in case of an operation, invoked. On the other hand, operations added on the oBIX level are again represented as sub objects.

Sub objects can be included in their full XML representation or via a reference. Whether composition by containment or reference is to be used has to be determined when designing the class hierarchy. For example, an oBIX alarm object is encoded as:

```
<obj name="somealarm" is="obix:Alarm">
   <ref name="source" href="/myhouse/somewhere"/>
   <abstime name="timestamp"
         val="2006-10-12_12:11:02"/>
</obj>
```

The object referenced by the "is" attribute is called a "contract." It acts like a template for the referencing object. All sub objects of this reference object are inherited by default. Still, the referencing object can override these values (sub-objects). So this object referenced by the "is" attribute defines a class (in the sense of object orientation).

Objects can fulfill multiple contracts (resulting in multiple inheritance). Finally, contracts can also be empty and merely describe semantics of an object – a prominent example being the (read-only) oBIX Point. In many aspects, contracts are similar to Java Interfaces.

An example contract describing a generic model of a furnace (a furnace template or "class"):

```
<obj href="def:furnace">
   <bool name="burnerOn"/>
   <real name="curTemp" is="obix:Point"/>
   <real name="setTemp" val="50.0"
      is="obix:WriteablePoint"/>
</obj>
```

Note the use of point contracts (the WriteablePoint contract adds a write operation) and that a default value is specified for the second. An object following this contract:

```
<obj name="furnace" href="myhouse/heating/furnace"
     is="def:furnace">
   <bool name="burnerOn" val="true"/>
   <real name="curTemp" val="45.3"/>
</obj>
```

This would be an instance of the furnace class (an actual furnace). It inherits the default value of 50 for the set temperature. By specifying the curTemp sub object itself, any possibly inherited default value is overridden. However, the point contract for this attribute is still inherited. To allow access by an oBIX client, an href is specified.

III. Representing KNX in oBIX

Our mapping of the KNX system to oBIX data model and services shall allow a plain oBIX client to pull data from a KNX installation (e.g. monitor a room temperature) and influence the process (e.g. switch off all lights). The mapping shall provide the means to build up a remote management station to control, monitor and change parameters of the KNX installation. Thus, it shall fulfill the first (access to operating information by enterprise applications software) and, to some extent, second (enabling unified management workstations regardless of the control system in use) of the application classes described for XML and WS in [4].

To this end, a mapping of datapoint types and the services for process and management communication is required. Also, a suitable form of discovery of the mapped datapoints must be supported. Generally, the mapping should leverage native oBIX language element semantics. For example, incoming events from the KNX system should be represented by an oBIX feed rather than a simple value object.

A. KNX interworking overview

KNX [6] is an established home and building automation system (building on the legacy of its predecessor EIB). Like most distributed automation systems, it uses functional blocks to model system functionality.

Functional blocks (FBs) are logical parts of a device, representing a function of this device. For example, an FB "push button" could describe part of a switch and an FB "light switching actuator" part of an eight fold power relay. FBs do never span more than one device.

FBs are associated with a set of datapoints (DP), which act as communication endpoints providing access to the functions of a block. Their syntax and semantics are well-defined for a particular FB type. For example, a "dimming actuator standard" FP provides the DP "switch on off" that can be used to turn the light on and off.

In KNX, DPs can either be realized as group objects (GOs) or interface object properties (IOPs). GOs are endpoints for KNX group communication relationships, which use a content-addressing scheme to provide producer-consumer style multicasts. Process data, i.e., communication between sensors, actuators, and controllers, are exclusively exchanged via group communication. With KNX group communication, data can either be transmitted in a spontaneous push-style using only write-request messages, or in a request-based pull-style using read-request and read-response messages. The mode of communication is determined at system setup time depending on whether the semantics of the GO are stateful or stateless. IOPs are solely used for management data transmission (parameters, configuration and diagnostic data). They are individually addressed via the physical address of the device, object index and property key. Reading and writing IOPs follows a simple client-server (pull-style) communication pattern.

The syntax and partial semantics of a datapoint value (GO as well as IOP) are specified in the datapoint type (DPT). For example, the DP "switch on off" accepts values conforming to the DPT "boolean switch." A DPT describes the bit-level encoding as well as aspects such as valid range, state labels (for Boolean values) or units. It does not describe how changes of the DP's value relate to other DPs or physical inputs and outputs (this is specified in the FB definition).

B. DPT mapping

As a necessary prerequisite to interact with the KNX installation, the means for representing data in a KNX system – the DPs – have to be made available on the server. First, the information carrying part of the DP has to be mapped, that is, the object types. The data types used within the KNX system are defined in the DPTs.

A KNX DPT specifies the data type of a value (format and encoding, e.g. two's complement, or IEEE float) as well as the dimension (range and engineering unit). The data type is mapped to oBIX value object types – e.g., a DPT_B1 is directly mapped to a `<bool/>` element. For complex data types, object containment is used. Table I shows some correspondences:

TABLE I
MAPPING OF THE DATAPOINT TYPES

Datapoint type	oBIX object
Boolean value (DPT_B1)	`<bool/>`
unsigned value	`<int/>`
float value	`<real/>`
bit	`<int max="1"/>`
(single) ASCII/ISO 8859-1 char	`<str/>`
ASCII/ISO 8859-1 string	`<str/>`

Control_Dimming (DPT_B1U3, complex type)	`<obj href="knx:DPT_B1U3">`
	`  <bool/>`
	`  <int max="7"/>`
	`</obj>`

The dimension is represented via an additional contract that adds range and unit information by using the means available in oBIX (e.g., the min/max facets or the oBIX unit system). In most cases, this amounts to adding application semantics and human readable names (the `displayName` facet):

```
<obj href="knx:DPT_Control_Dimming" is="knx:DPT_B1U3">
   <bool name="B1" displayName="brightness"
      range="knx:range/incDec"/>
   <int name="U3" displayName="stepcode"/>
</obj>

<list href="knx:range/incDec" is="obix:Range">
   <obj name="true" displayName="increase"/>
   <obj name="false" displayName="decrease"/>
</list>
```

C. Communication services mapping

Having found a suitable oBIX representation of the DPTs, the ways and means to interact with a DP remain to be mapped. In KNX, process communication happens via group communication only. A process data DP can never be addressed individually. Therefore, a Group Communication Endpoint (GCE) class is introduced to enable communication with GOs. A GCE represents a server-side facility for data exchange via one particular group address (roughly comparable to a

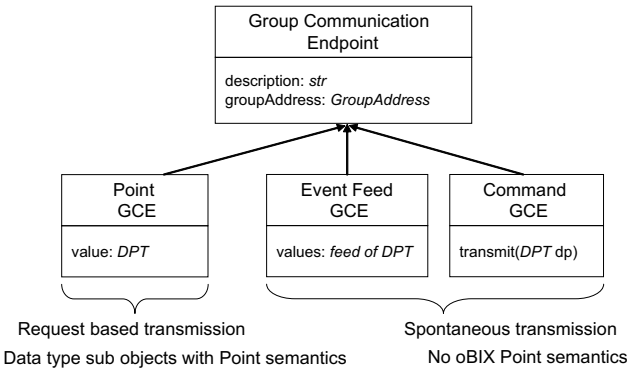

Fig. 1. Group communication endpoint mapping

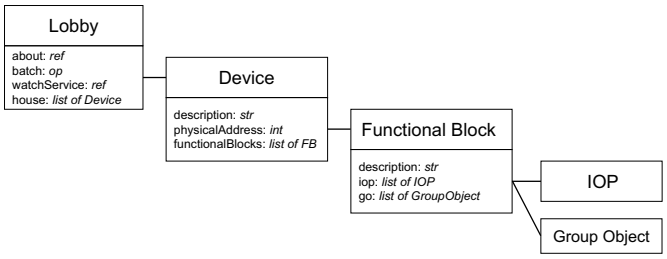

Fig. 2. Discovery: management view

UDP socket).[1] An interaction with a GCE always effects the exchange of a single DP value via the KNX network. But, depending on the mode of communication associated with that group address, different types of interaction make sense. Therefore, the GCE is subclassed to represent the proper operations (Fig. 1).

The first case to support is request based transmission with the oBIX server as data sink. In this case, the associated GO has state semantics. This means the server can always retrieve the current value from the network. This maps perfectly to the semantics associated with oBIX Points, hence the name Point GCE was chosen. An example could be a current lux value obtained from a sensor. Request based transmission with the server as data source makes no sense for a management station and thus does not require a representation in our mapping.

Since the oBIX Point contract can only be assigned to primitive value types, the server must care to promote it to the sub objects of a Point GCE instance in case this instance references a complex DPT. When the oBIX client chooses to retrieve one of the sub objects of such a Point, the server retrieves the entire complex DP value from the KNX side. Since state semantics are implied, this is not a problem.

The second case is spontaneous transmission. In this case, the associated GOs have event semantics. Two subclasses exist corresponding to the role of the server. If the server acts as data sink, the Event Feed GCE buffers incoming events using an oBIX feed object. Since the server cannot actively retrieve the current value of anything from the network, the Event Feed GCE does not have Point semantics. An example would be a motion sensor sending a trigger event. If the server acts as the data source, the Command GCE contract provides a transmit operation (e.g., to send an "increase brightness" command to a dimmer).

One case remains to be discussed. KNX allows every combination of communication modes for a group object, placing the decision on what is appropriate in this respect in the hands of the project engineer. This includes group objects that can act both as data source with state semantics and sink, corresponding to the oBIX Writeable Point. Their use is discouraged by the KNX standard due to protocol related side effects which are complex to handle. Also, an oBIX client can choose to update only part of a complex DPT, which further obscures semantics. As a matter of fact, however, KNX device manufacturers do not always follow this recommendation. Thus, the Writeable Point GCE is optional in our solution to be available if the setup of the KNX system requires it.

Management communication is effected via interface object properties and point-to-point ("physical") addressing. Mapping this communication mode is straightforward as long as read and write access to single properties is all that is required (in contrast to, e.g., program downloads, which require a complex access sequence). For this purpose, an IOP object is introduced which holds the required addressing information. The access types of an IOP are simply read-only and read-write. This perfectly corresponds with the oBIX Point and Writable Point concept, resulting in the two subclasses "Point IOP" and "Writeable Point IOP".

D. Discovery

With DPTs, GCEs and IOPs specified, the means to interact with KNX devices are available. Still, a client needs to discover which entities are present on the oBIX server. The server could easily provide a flat list. However, this is obviously not an optimal solution; a hierarchical structure is desirable. Since oBIX does not provide predefined structure elements, these shall be custom defined. Discovery builds on the fact that in oBIX objects are composed from other objects. All objects, which can be reached from the Lobby, either directly or through other objects, by way of containment or reference, are discoverable. The structure of KNX systems leads to two approaches for discovery: device or group address centric.

The probably most obvious possibility is to map the KNX system structure to the oBIX object model in a device oriented way. In such an approach, the system functionality is perceived as a collection of devices and functional blocks. The FBs are necessarily organized per network device because a FB does not span more than one device. An oBIX client enters via the Lobby and receives a list of devices with their associated functional blocks. These functional blocks in turn consist of IOPs and group objects (Fig. 2).

This view is useful to access parameters and diagnostic data, which is why we call it the "management view." However, it is not useful for controlling the process, because process communication exclusively uses group communication in KNX.

[1]This is different from a KNX GO, which can be associated with one group address for outgoing plus multiple group addresses for incoming messages. The name "GCE" was deliberately chosen over "GO" to make this difference visible.

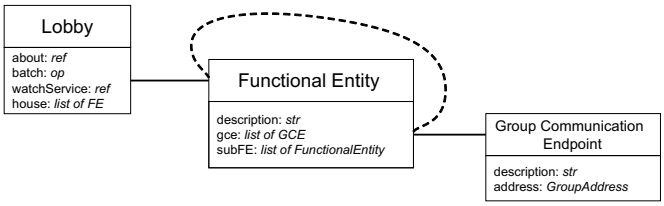

Fig. 3. Discovery: process view

Group communication can affect any number of devices, and due to the producer-consumer style of communication these cannot be trivially enumerated, either. Thus, the "group objects" in Fig. 2 can be used for information only, showing which group addresses are associated with a particular DP. The oBIX server could of course easily provide these group addresses in the form of GCEs. However, this is useless without information about which function will be effected by interacting with such a GCE.

Therefore, a discovery scheme for process communication must focus on functions rather than individual devices. Someone who wants to switch on the light in a room typically enters the room and flips the switch. Being able to use the same procedure in the oBIX representation would be desirable: entering the house via the lobby, selecting the room (a functional entity, FE) on a list and flipping a virtual switch in this room (by invoking a function). Focusing on functions in KNX means focusing on group addresses. In our mapping, the means for interacting with groups of KNX devices are GCEs. To bring them in some order, we use a sort of directory tree. From the Lobby, the client sees FEs such as "west_wing" or the "HVAC" system. These FEs can be composed of GCEs – or again FEs, such as rooms for example (Fig. 3). A roughly similar concept is found in BACnet/WS, where a similar hierarchy is used to organize system data, as well as in the FGAG extension to KNX [3].

Note that, keeping with the oBIX specification, we do not regulate the namespace of the href URIs in order to structure the GCEs (and rather leave this to the individual server implementations). What is the benefit of introducing FE objects? At first glance, it would probably appear more convenient if one were able to access the light in the cafeteria via "mybuilding.org/obix/cafeteria/ceiling_light" or similar. However, to obtain a list of GCEs related to the cafeteria, one cannot simply retrieve the object with the URI "mybuilding.org/obix/cafeteria" as one would do to obtain a directory listing in a file system. The server will not return anything unless an object with this particular href has been registered. Obviously, some kind of directory object is needed. Actually, the FE object is precisely that: it contains a list of GCEs (the "files") and a list of FEs (the "subdirectories"). Also, it should be kept in mind that we are dealing with machine-to-machine communication and human users will seldom be exposed to the URI string.

IV. Server implementation

The server implementation is intended as a proof of concept and thus has several limitations. Currently, only the process

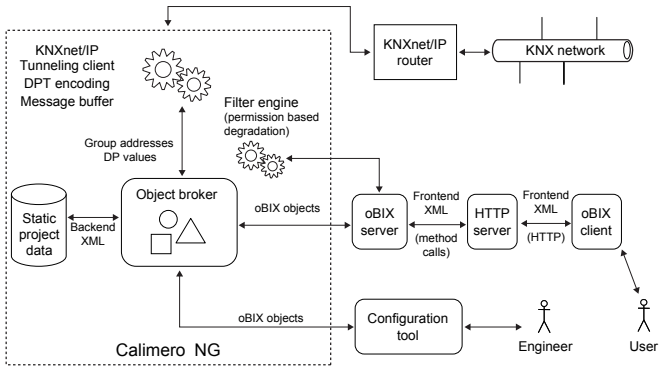

Fig. 4. oBIX server implementation

view and GCEs (i.e., the necessary means for group communication) are implemented. The mapping of DPTs is limited to the most popular types, e.g., DPT_B1 to turn on a light via a switching actuator and DPT_U8 to set the power level of a dimming actuator. oBIX object access is restricted to basic services. Batching, alarms and histories are not available. Configuration data has to be supplied manually.

The server uses the plain HTTP oBIX protocol binding. It is operational, and its full source code is available.[2] Although the code is not optimized for performance, first evaluation results show that the server (running on a standard office-class PC) can easily deliver and accept messages with the maximum rate available on the KNX twisted pair medium.

A. Design

The implementation uses Java and is based on existing open source software projects – the oBIX toolkit, version 0.12.0 [8] and the Calimero library, version 1.3 [2]. Calimero is used for network access via KNXnet/IP Tunnelling and DPT transcoding. Since incoming frames from the KNX network cannot be pushed to the oBIX client, they must be buffered until the client polls. Calimero was extended to provide the necessary functionality ("Calimero Next Generation"). Currently, the buffer only holds a single message. The server design is shown in Fig. 4.

The **data base** holds static information about the KNX installation. It uses the same object model as is used for communication with the oBIX client. This is possible thanks to the extensible oBIX data model and allows a very clean and straightforward server design. Of course, attributes such as KNX group addresses contained in a GCE are of no interest for oBIX clients and thus are masked out using the filter engine when answering a client request.

The data base is implemented as text file holding the XML representation of the oBIX objects. Information about the datapoints in the KNX system, their communication modes and addressing information as well as friendly names have to be provided during the configuration phase. While this has to be done by manually editing the XML file for the time being, the design intends to provide a (possibly graphical) tool, which interacts with the data base by way of the object broker.

[2] http://www.auto.tuwien.ac.at/~mneug/knx2obix_src.tar.gz

The database also holds some server configuration data. Up to now, this includes queue depth and caching information for the event feed and read only GCEs. Eventually, access control lists may also be stored in the objects.

The **HTTP server** implements the oBIX HTTP protocol binding. It is responsible for handling HTTP requests (PUT, GET, POST + data) issued by the oBIX client. It invokes the appropriate methods (getObject, writeObject and invokeOperation) on the oBIX server component, passing the URI (href) and XML stream it received from the client. It also handles user authentication and informs the oBIX server module about the user name associated with every request. For GET and POST operations, the XML stream returned by the oBIX server is passed back to the client.

The **oBIX server module** (not to be confused with the entire oBIX server application) operates on the level of oBIX network requests. It translates between oBIX XML streams and their Java object representations. To actually execute the requests (e.g., retrieve a GCE with a particular URI), it invokes the appropriate methods of the object broker.

Any XML streams returned to the caller (the HTTP server) are filtered according to the authorization level associated with the user name specified. Also, unauthorized requests are denied. For this purpose, the oBIX server component maintains a user database. The actual XML filtering is done by a separate component, the filter engine.

The **object broker** provides a front end to the data base. It allows to store, retrieve and delete objects by way of their URI (href), ensuring the persistence of changes via the data base. For purely static objects (such as FEs for discovery), nothing more is required. For GCE (and, in future, IOP) objects, however, "smart" operations are provided that enrich the static project data with dynamic process data. For example, when a read operation on a Point GCE object is requested, the object broker not only retrieves the object corresponding to the URI from the data base, but accesses the KNX network using the addressing information contained within the object. The retrieved current value is stored in the value property of the GCE object before returning it to the caller.

The **filter engine** removes information from oBIX objects which the client is not expected to need (e.g., KNX group addresses) or provide. This is done to provide a clean interface as well as for security reasons. oBIX suggests permission based degradation, which basically means that users can only see and manipulate the objects they have the permission for. The filter engine thus removes (sub-)objects which are below the specified authorization threshold. For all users except administrators, it will remove the KNX specific data (such as network addresses). In the current implementation, the filter engine does not remove anything.

B. KNX message buffering

Calimero NG provides a buffer for process data messages. The type of buffering required depends on the mode of communication.

Queueing: Event feed GCEs are polled regularly by the oBIX client. Every time a message addressed to the group address associated with the GCE appears on the KNX network, Calimero NG internally queues this message. When the event feed GCE is polled by the client, the server fills it with the values stored in this queue. This type of buffering is not optional, since the server cannot push spontaneously incoming messages to the client.

Caching: If the client issues a "read" command on a Point GCE, the server will by default pass the read request to the KNX network. Calimero NG can cache the received value instead and service future client requests from the cache. After a configurable timeout (or proactively in a certain interval), the value is again retrieved from the KNX network to refresh the cache. In addition, it is updated when push-style notifications are received. For example, consider a sensor that sends its value both periodically and additionally whenever it changes. In this case, the refresh timeout can be set to zero (never). This type of buffering is optional to enhance performance and reduce KNX network load.

V. Outlook

oBIX is a powerful and flexible tool. In our case, it did not only allow to represent the constructs on the control network side seamlessly, but also easily doubled as the back end server data structure. This allowed an unusually simple server design and low development time. As another benefit of the XML format, configuration of the server, both for the management and the process view, may become a matter of a single XML transformation in the future. This will be the case once the ETS (KNX Engineering Tool Software) XML export format is stable and makes broad use of the standard functional blocks already defined in the KNX specification.

However, this flexibility necessarily comes at the price that oBIX is by far not trivial to understand. A thorough comparison of oBIX with BACnet/WS and OPC UA would be desirable.

Apart from these implementation specific aspects, the development of enterprise-level abstractions (whether using Web services or not) has to be kept in view. If one was to judge by the speed of oBIX development up to now, high-level oBIX abstractions (working title "V2") could reach draft status soon. However, such extrapolation is hardly justified, as this is a far more complex matter than datapoint abstraction.

References

[1] *ANSI/ASHRAE Addendum c to ANSI/ASHRAE Standard 135-2004,* American Society of Heating, Refrigerating and Air-Conditioning Engineers, Atlanta, 2006.

[2] Calimero — EIBnet/IP Tunnelling (and more) for Java, http://calimero.sourceforge.net (accessed Jan. 25, 2007).

[3] Cezary Szczegielniak and Markus A. Wischy, "Using FGAG to export ETS data for Visualization," *Proc. KNX Scientific Conference 2005,* 2005.

[4] David Fisher, "XML, Web Services, and the Problems of Enterprise-Level Data Exchange," *HPAC Engineering,* vol. 76, no. 4, pp. 13–14, April 2004.

[5] Francois Jammes and Harm Smit, "Service-oriented paradigms in industrial automation," *IEEE Transactions on Industrial Informatics,* vol. 1, no. 1, pp. 62–70, Feb. 2005.

[6] *KNX Specification, Version 1.1,* Konnex Association, Diegem, 2004.

[7] *oBIX 1.0 Committee Specification 01*, OASIS, 5 Dec. 2006.

[8] oBIX Java Toolkit, http://sf.net/projects/obix (acc. Jan. 25, 2007).

[9] The OPC Foundation, http://www.opcfoundation.org (acc. Apr. 12, 2007).