2012 IEEE International Conference on Green Computing and Communications, Conference on Internet of Things, and Conference on Cyber, Physical and Social Computing

# A transparent IPv6 multi-protocol gateway to integrate Building Automation Systems in the Internet of Things

Markus Jung, Jürgen Weidinger,
Christian Reinisch, Wolfgang Kastner
Institute of Computer Aided Automation
Vienna University of Technology
Vienna, Austria
e-mail: {mjung,jweidinger,cr,k}@auto.tuwien.ac.at

Cedric Crettaz
Mandat International
Geneva, Switzerland
e-mail: iot6@mandint.org

Alex Olivieri, Yann Bocchi
Institute Informatique de Gestion
Haute Ecole Spécialisée
de Suisse occidentale
Sierre, Switzerland
e-mail: {yann.bocchi,alex.olivieri}@hevs.ch

***Abstract*—The future Internet of Things (IoT) should enable machine-to-machine interaction for devices out of numerous domains. Recent developments and standards focus on how to deploy IP directly on devices and investigate application protocols that fit the constrained environments, whereas research on the integration of widely deployed legacy devices of technologies like BACnet, LonWorks and KNX is still neglected. For a success of the ambitions towards an IoT we identify it of highest importance to research various integration styles for non-IP based devices already deployed in home and building automation. Therefore, this paper contributes an overview of various possible integration styles, provides a concrete multi-protocol integration architecture and presents evaluation results of a proof of concept implementation.**

***Index Terms*—Internet of Things, IPv6, Building Automation, Multi-protocol integration, Web services**

## I. Introduction

The Internet of Things (IoT) promises to interconnect billions of devices out of different domains, ranging from supply chains, telecommunications, home and building automation, entertainment and more recently also smart metering and smart grid infrastructures. For the desired interconnection, a trend towards Web service based communication protocols resting on HTTP and XML can be seen. Also, resource oriented so called RESTful Web services seem to be favored over SOAP based approaches which tend to be better suited for the integration of enterprise IT systems.

Standards for a so called constrained restful environment have been identified by an IETF working group[1]. Core technologies to be supported by future native IoT devices include IPv6, IPv6 on low power wireless personal area networks (6LoWPAN) and the constrained application protocol (CoAP). They allow equipping every device with a unique (IPv6) address and even deploy the required protocol stacks on resource limited (embedded) devices with only a fraction of memory capacity compared to an (embedded) PC.

As a next step, the interconnection to building services has to be taken into consideration. Within this domain, several standards and technologies exists, like BACnet, LonWorks, KNX, DALI, EnOcean and ZigBee to name just a few of them. They are tailored to the very specific needs of building automation and the respective building services. It is unlikely that they will be replaced by a new system based on an IoT protocol stack with IPv6 at its network layer. Instead, it is necessary that they are integrated into the IoT bridging the benefits of both worlds. For the integration of building automation existing technologies like OPC UA [1], BACnet/WS [2] and oBIX [3] provide the required capabilities to map building automation systems(BAS) technologies to Web service based protocols. These technologies provide standardized information models and protocol definitions that are not provided by CoAP. In [4] oBIX is mentioned as possible application layer protocol for integrating BAS:

> "Instead of running a control network specific building automation protocol such as BACnet/IP or KNX over 6LoWPAN, oBIX together with compression and UDP/IP binding may be a solution."

Therefore this paper presents an IoT protocol stack that uses oBIX, tackles the compression of XML messages through EXI and offers a UDP protocol binding based on CoAP with IPv6 basis. Furthermore, different state of the art integration styles of BAS are discussed that outline alternatives to the presented approach. Finally, the architecture of a multi-protocol gateway is presented that allows a seamless integration of existing BAS technologies into the identified IoT protocol stack. Further, a proof of concept implementation is used to evaluate the results.

The paper is structured as follows. Section II provides an overview of related research work. Next, different state of the art integration styles for BAS are discussed in Section III. A possible target IoT system and its protocol stack are outlined in Section IV, followed by an integration approach based on a transparent multi-protocol gateway in Section V. Finally, Section VI summarizes the results and contributions and provides an outlook on further work.

[1] https://datatracker.ietf.org/wg/core/charter/, Accessed: 15.05.2012

978-0-7695-4865-4/12 $26.00 © 2012 IEEE
DOI 10.1109/GreenCom.2012.42

## II. Related Work

In [5], Shelby presents how RESTful Web services can be adopted for a deployment on constrained devices connected through wireless networks. The paper introduces the standardization work done by the IETF Constrained Restful Enviroments (CoRE) working group, with the Constrained Application Protocol (CoAP) as main contribution. The interaction through CoAP with native IoT devices is sketched and the possible architecture is outlined. [4] deals with the implications of constrained data links that operate IPv6 on the upper application layer protocols. Furthermore, this book provides first ideas how to integrate various application layer protocols in the IoT.

[6] presents how CoAP and EXI can be used to deploy Web services on constrained devices. The experimental evaluation gives first insights in the footprint of the CoAP stack implementation and packet size improvements that can be gained by the use of EXI. In [7], it is shown how CoAP can be used as transport binding for SOAP Web services using DPWS on top of the implementation. Both papers present possible protocol stacks for native IoT devices. In contrast, we present how existing BAS devices can be integrated through a transparent gateway in such novel systems.

State of the art integration of BAS using Web service technologies like OPC UA, BACnet/WS and oBIX are presented in [8], [9] and [10]. The implications of using IPv6 as network layer for the integration of BAS in the IoT are analyzed in [11].

The advantages of using Web technologies for the IoT and a first realization of a Web of Things are presented in [12] and in [13].

Semantic problems that arise when different application layer protocols are integrated, and a solution to these problems based on ontologies and semantic Web technologies are addressed by e.g. [14] and [15].

In 2008, the Universal Device Gateway [16] research project explored and tested the potentiality of integrating heterogeneous communication protocols through IPv6. The UDG architecture demonstrates the possible integration and interoperability among various protocols used in building environment, such as KNX, X10, and ZigBee. The protocol piles are integrated from physical to application layer, with an abstraction of the various application layers into a unified semantics. This architecture is used as a multi-protocol gateway as well as an interoperability enabler for cross-protocol interactions. UDG builds on IPv6 as the core network protocol and hence uses IPv6 addresses to communicate with devices and the newly developed UDG multi protocol boards.

## III. Integration Styles

Typical building automation systems (BAS) define several layers of the ISO/OSI reference model and care for a specific application model. The application models usually follow a *data point* approach, meaning that every device is expressed as a collection of input and output data points of well defined data types. Data points are often grouped to functional blocks which define a desired behavior (e.g. light switch actuator). The meta data and semantics of functional blocks are provided in a human readable way. Furthermore the application layer protocols are strongly aligned to the custom network and data link layers of the underlying BAS technology, which typically require a variety of interaction styles like client/server, producer/consumer, or publish/subscribe communication. The network layer of a BAS is usually kept quite simple. However, reliable and non-reliable data transfer, point-to-point, multicast or broadcast based communication should be supported.

For BAS integration in an IoT it is not possible to simply put application layers of building automation technologies on top of a common network layer, since these technologies usually define custom protocols for the transport and network layer. Instead, gateway devices are necessary. As shown in Fig. 1, these gateways may operate either following an *N-to-N* protocol mapping approach or may map all technologies to one protocol following an *N-to-1** approach. In this case, *1** stands for a new IoT protocol stack (IoTsys) that allows native interaction with a device in the IoT.

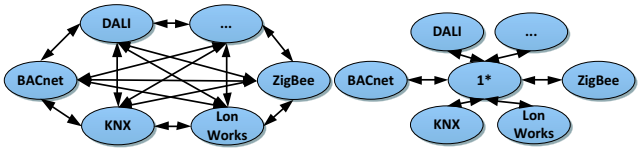


**Fig. 1:** N-to-N or N-to-1* BAS integration

For an N-to-N integration two BAS systems are made compatible with each other through gateways featuring different network options. In the best case, the gateway is not visible for the other BAS system. If two BAS technologies rest upon the same network layer, multi-protocol devices may come into play as presented in [17]. Nevertheless, for this kind of integration in the worst case $\frac{n*(n-1)}{2}$ mappings are required.

The N-to-1* integration approach refers to the integration of different building automation systems to a common target system. The mapping effort is reduced to $n$ mappings, since all technologies have to be integrated only into one new technology. The problem remains to identify the commonly accepted 1* technology that is ready for the Internet of Things. Our work focuses on a common technology based on Web services to provide open and interoperable interfaces. In this respect, technologies like OPC UA, BACnet/WS and oBIX are candidates. Their Web services interfaces can even be used for enterprise application integration. Figure 2 shows an *N-to-1** mapping of two BAS representatives (i.e., BACnet and KNX) to the IoT, where IPv6 acts as a common IoT network layer. Though different options for layers above IPv6 exists, this paper concentrates on UDP, CoAP and oBIX.

When it comes to the integration of BAS another decision to be made is at which point the integration happens. It is either possible to provide each device with a Web service interface or to offer centralized interface types. A centralized server has the advantage of providing the required computational resources for a Web service interface for all devices behind

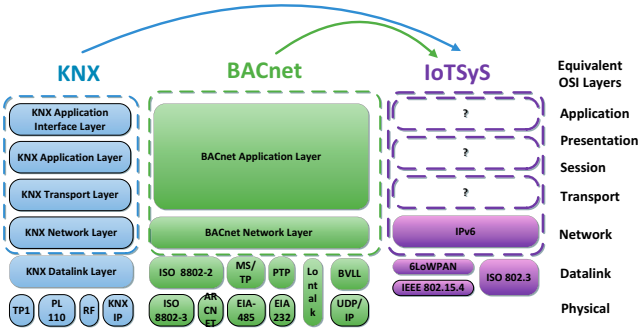


**Fig. 2:** Integration challenges

it. A decentralized approach requires more computational resources on field devices like sensors and actuators but has the advantage that Web services can natively be used to interact with a device and the devices by themselves may even use Web services to interact with each other (if IP is supported as network option). Fig. 3 identifies four possible integration approaches.

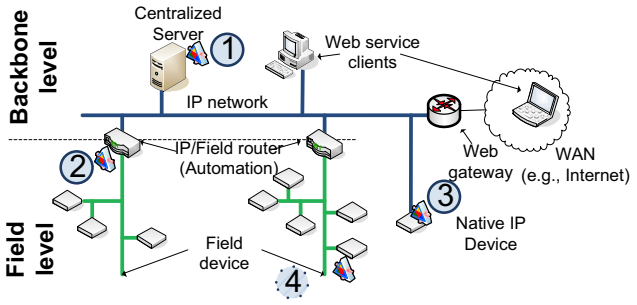


**Fig. 3:** Centralized and decentralized integration

1) A centralized server at the IP backbone of a BAS is the state of the art approach and allows the integration into enterprise systems and remote access.
2) Providing the Web service interface at the automation layer (which bridges the backbone and the field layer) via an IP/field router is a more decentralized approach but from the application layer aspects equivalent to the central deployment.
3) IP field devices using IP not only as data link but also as network layer protocol can be equipped with Web services and may offer this interface directly to other devices and Web service clients.
4) Finally, it is even possible to emulate Web service interfaces with a separate IPv6 address either at a centralized server or at an IP/field router acting as transparent gateway. Equipping this gateway with multiple protocol stacks and physical interfaces to different media allows to build a transparent multi-protocol gateway mentioned above.

## IV. The IoT System

For integration of existing BAS technologies into an IoT system standardization should not be neglected. An important work towards common IoT standards is handled by the IETF working group on Constrained RESTful Environments addressing the Constrained Application Protocol. CoAP is designed to work on a protocol stack based on IPv6 and UDP, which perfectly fits wireless devices running, for instance, 6LoWPAN. CoAP is an application protocol that allows interacting with resources on the devices (sensors, actuators or controllers) in a RESTful interaction style.

IPv6 is a main building block of the IoT, since the 32 Bit address space provided by IPv4 is already exhausted and not capable of providing each device with a unique address. The 128 Bit address space of IPv6 offers the required means to host billion of devices which will be part of an IoT. With IPv6, each device gets an IP address for identification. It enables globally valid IP addresses for endpoints and allows direct end-to-end communication and security facilities without the need for the definition of an overlay network as required for peer-to-peer networks with peers residing in local IPv4 networks. More information on how IPv6 supports the integration of building automation in the IoT is presented in [11].

IP provides the facilities to send one packet from one communication partner to another through connected IP networks. On top of IP either the connection-oriented TCP can be used or one may opt for the packet-oriented UDP. Both protocols have their particular advantages and drawbacks. When it comes to the definition of new native IP devices that probably operate wirelessly and on constrained resources, UDP is a superior candidate. If one considers 6LoWPAN as the most promising option for operating IPv6 directly on devices, UDP is the best choice since IEEE 802.15.4 provides a frame size of 127 Bytes with a layer two payload size below 72 Bytes [4]. In order to avoid fragmentation, the application layer protocol data units should not exceed this limit. Now the question arises which application layer protocol should be used and how the application layer interfaces and services should be defined. In the Internet, Web services are a sound way to offer services to a service consumer and to realize machine-to-machine communication. Where in the enterprise IT context SOAP based Web services are prevalent due to the large number of features of the WS-* stack, for resource limited devices or in the mobile area RESTful services are a preferable choice. RESTful services follow a resource oriented service design and rely on proven technologies like DNS, HTTP and XML that once made the WWW successful.

For building automation integration, the OASIS standard oBIX is one of the most promising choices. oBIX provides an object model to represent devices in the domain of building automation as well as a Web service based protocol to interact with the oBIX objects. It follows a RESTful design and is based on a simple protocol that can be mapped to HTTP or SOAP. XML is used for the encoding of the objects. By using HTTP and XML, oBIX servers provide an interoperable

interface that is not restricted to any specific platform. A client-server model is used for the interaction between oBIX clients and servers defining three request types on objects, which are i) `read`, ii) `write` and iii) `invoke`. These request types are mapped straight forward to the HTTP protocol verbs `get`, `put`, and `post`. The object model provides the required meta model for information modeling of resources. It defines 17 standard object types ranging from basic data items like `bool`, `int`, `real` and `str` over to more complex object types. Figure 4 provides an overview of the oBIX object model. Each object is addressable through a uniform resource identifier (URI). Extensibility is provided, allowing to define custom objects and contracts based on the existing XML syntax defined by the standard. Object types are defined through so called oBIX contracts that provide a template and common object definition that represents the semantics of the "types" used in oBIX. These contracts can be used to build clients and servers, probably by different vendors, that interwork out of the box. The object model, the network protocol and the contract methodology are enriched with a support for `watches`, `histories` and `alarms`. For clients being notified about changes of objects, oBIX provides a model for client polled eventing called `watches`. A `watch` represents a per-client state object that is created by a client using the `watch service`, exposed at a well-known URI. The client can register objects hosted by the server using their respective URIs and then poll for changes using the `pollChanges` operation defined by the `watch` object. However, due to the usage of HTTP, a client still has to poll for changes. The only improvement by this mechanism is that the full object is not transmitted twice if no change occured since the last poll for change.

The `history` object allows to query for the history of an arbitrary data point or oBIX object that implements the `obix:history` contract. For querying a history object, a filter with a number of records limit and a data range can be provided. Furthermore, a roll up can be done on a time series of elements and the results can be provided in a normalized interval.

The alarming feature of oBIX allows querying, watching and acknowledging alarms using a normalized interaction model. An alarm is used to indicate a certain event that requires an action or a notification of a user, e.g., a system operator. This requires an acknowledgement if a compensating action is executed and furthermore a method is required to resolve the alarm. As for watches, the alarming of oBIX still requires the client to poll for alarms.

Regarding security, oBIX does not specify any mechanism and shifts this responsibility to the transport layer of the protocol.

What remains between oBIX and IPv6 is the protocol used for message exchange and the transport layer. For this reason, oBIX provides a protocol binding for HTTP and SOAP. Both HTTP and SOAP rely on TCP for reliable message exchange for a connection oriented communication between two parties. This is where CoAP comes into consideration.

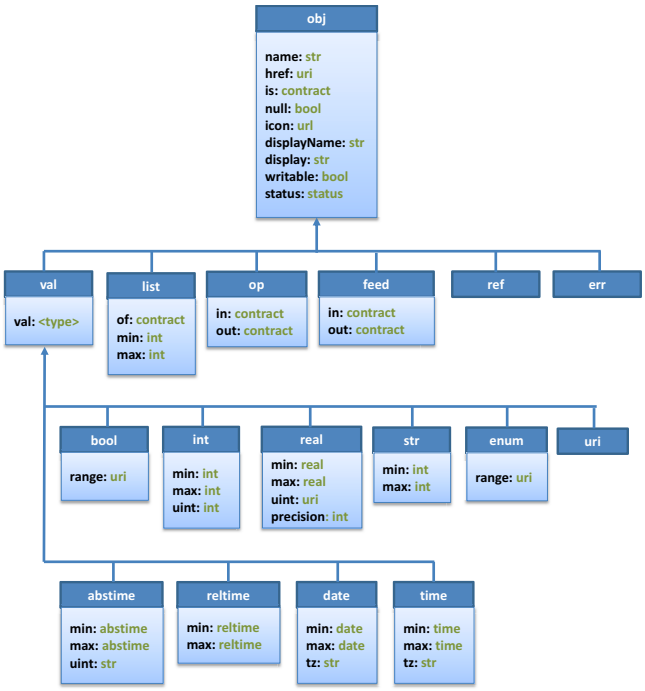


**Fig. 4:** oBIX object model [3]

CoAP is a mapping of the TCP based HTTP protocol to UDP, but due to the connectionless message exchange defines facilities for acknowledged message exchange (cf. Figure 5) and furthermore allows multicast based and asynchronous communication. Figure 6 illustrates the abstract layering of the CoAP protocol stack, which maps the asynchronous packet oriented communication of UDP to a request/response interaction aligned to HTTP.

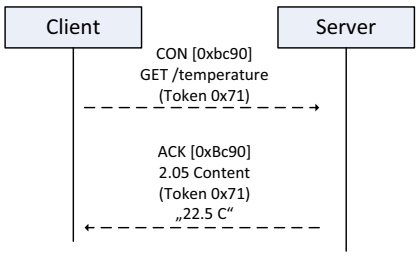


**Fig. 5:** CoAP message exchange [18]

Still, for carrying oBIX messages, a protocol binding for CoAP needs to be defined. Furthermore, the concepts of oBIX alarms and event feeds can take benefit of the asynchronous communication of CoAP. Using XML based messages is heavyweight for constrained devices given that IEEE 802.15.4 as data link layer limits the application payload within a single message to less than 72 Bytes in order to be transmitted without fragmentation. oBIX contracts need to be carefully designed to run over such links. A solution to this problem is

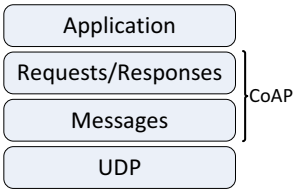


**Fig. 6:** Abstract layering of CoAP [18]

the binary protocol binding of oBIX. Another way to solve this problem in a more standardized way is to use efficient XML interchange (EXI) for encoding the XML artefacts. EXI is a W3C recommendation[2], therefore making this optimization technology the more preferable way.

Figure 7 provides an overview of the possible protocol alternatives of an IoT protocol stack identifying the most promising candidates.

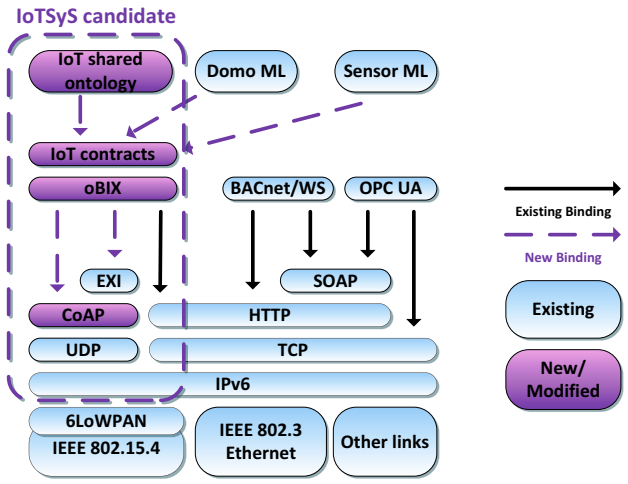


**Fig. 7:** IoT protocol stack

### A. Required CoRE extensions

CoAP in the current working draft[3] provides a request/response interaction model leaving the potential of asynchronous communication of UDP unused. The following message types are supported: confirmed (CON), non-confirmed (NON), acknowledged (ACK) and reset (RST) together with the methods `get`, `put`, `post` and `delete`. The advantages of asynchronous communication regarding observing changes of a resource are taken into account by the separate specification for observing resources in CoAP[4]. There the `observe` option is defined for `get` requests and allows to receive multiple response messages without the requirement of permanent resource polling.

[2]http://www.w3.org/TR/2011/REC-exi-20110310/, Accessed: 30.07.2012
[3]http://tools.ietf.org/html/draft-ietf-core-coap-11, Accessed: 30.07.2012
[4]http://tools.ietf.org/html/draft-ietf-core-observe-05, Accessed: 30.07.2012

oBIX provides the concept of `Watches`, `Feeds` and `Alarms` for resources, but is tailored to a polling interaction model. It is possible to create stateful Watch objects that can be polled by a client using the `pollChanges` operation defined by the oBIX object through a `post` request. This polling interaction model can be kept when using oBIX on top of CoAP but the advantages of asynchronous communication are not used. So for watching resources it is possible to simply rely on the CoAP `observe` option for `get` requests and to define this new semantics in a oBIX protocol binding to CoAP. However, it would be desireable to define the `observe` option also for the other request types. This would allow to perform a CoAP `post` call on an oBIX `pollChanges` operation and to receive continuous change notifications. Furthermore, for alarming it is desirable to provide certain conditions as payload. These conditions depend strongly on the information model of the application layer protocol above CoAP, so it is the most reasonable way to provide conditions as payload within either a `post` or `put` request.

### B. From oBIX to IoTSyS

Using oBIX as application layer protocol for the IoT requires several enhancements. Firstly, a protocol binding to CoAP needs to be defined. This works straightforward and a reference implementation has been performed for the evaluation of the presented concept in this paper. Secondly, instead of defining a custom binary application protocol it is desirable to use EXI. Promising results can be achieved using EXI even without having a schema. Thirdly, generic IoT contracts for the various device types need to be specified as oBIX contracts comparable to the use of functional blocks in BAS. This provides interworking additionally to the interoperability and further allows to define XML schema documents leading to a fixed EXI grammar and optimal binary representation of exchanged messages. IoT application protocol data should fit in the best case into a single packet of a constrained wireless data link. A careful design of the oBIX contracts is required and an EXI based compression allows to further optimize the packet size. A shortcoming of oBIX is that the contracts provide semantics only to human beings. This can be addressed by generating oBIX contracts from ontologies. Ontologies define a common vocabulary and provide the means to make semantics available for processing through machines. Generic ontologies that represent the common concept of different BAS application layers [14] can act as a solid foundation for a future IoT ontology. This shared IoT ontology can be used to generate oBIX contracts. oBIX objects can then be annotated with semantic attributes that refer to the according concept of the ontology. Existing standard models defined by SensorML [19] or DomoML [20] can also be used as references for an IoT ontology.

Regarding discovery of resouces, the CoRE link format uses the well known interface `/.well-known/core` to discover the resources hosted by a device. The oBIX lobby provides the same functionality and the representation can easily be mapped to the CoRE representation. Furthermore

the `rt` attribute reflecting the resource type can by assigned with the unique name of an oBIX IoT contract.

These mentioned adjustments of oBIX combined with a protocol binding to CoAP, a message encoding based on EXI and standardized oBIX contracts provide the protocol stack for an Internet of Things system. It is therefore named IoTSyS.

## V. Integrating building automation in the IoT system

The previous section outlined a possible protocol stack of a target IoT system. New IoT devices might work natively on that stack using IPv6 for addressing and a UDP based application layer protocol for message exchange, which allows to use this application layer protocol directly on constrained devices like 6LoWPAN nodes. The oBIX based approach in the previous section is one possible realization of an IoT system. With the extension to oTIX, a suitable API to devices is provided through a uniform interface and object contracts. The question still to be answered is how existing (legacy) systems can be integrated. For oBIX, BACnet/WS or OPC UA a centralized server that provides access to all devices of a BAS system is a common way. Devices directly interacting with each other using a RESTful approach are not the typical usage scenario for the existing BAS integration technologies. There is no direct interworking and for each specific enterprize system the integration has to be done manually.

In the centralized approach a single Web service endpoint is offered with a single IPv6 address. Thus, the addressing is moved to the information model of the specific integration technology. A gateway following such an integration approach from KNX to oBIX is presented in [10]. For the realization of the IoT, this integration approach is not satisfactory. The Web service interface acts as a single visible application level gateway for the client and direct device interaction is not possbile with such an approach.

In contrast the CoRE architecture presented by [5] uses Web services on a device level and a more convenient way for integrated existing devices is to build a transparent gateway, that provides an CoAP interface bound to an IPv6 address for each client that resides behind the gateway. This allows to extend the CoRE architecture transparently (see Figure 8) with devices using existing BAS technologies. Transparency is guaranteed because an application acting as a CoAP client cannot determine whether it is communicating with a native CoAP device or a legacy device residing behind the gateway. Furthermore, it is also possible that two legacy devices interact with each other using the IoT API.

### A. Multi-protocol gateway architecture

Figure 9 provides a closer view on the required components of the IPv6 multi-protocol gateway.

The protocol adapters (e.g., *KNX Adapter*) are key components of the gateway architecture. They provide the interface to the BAS specific application layer protocol. Depending on the BAS, the connections to different physical and data link layers need to be provided. Furthermore, the mapping of BAS

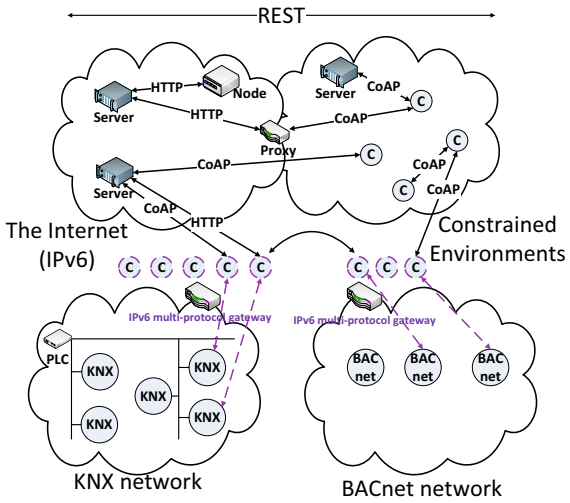


**Fig. 8:** Extending the CoRE architecture by Shelby [5] transparently with BAS

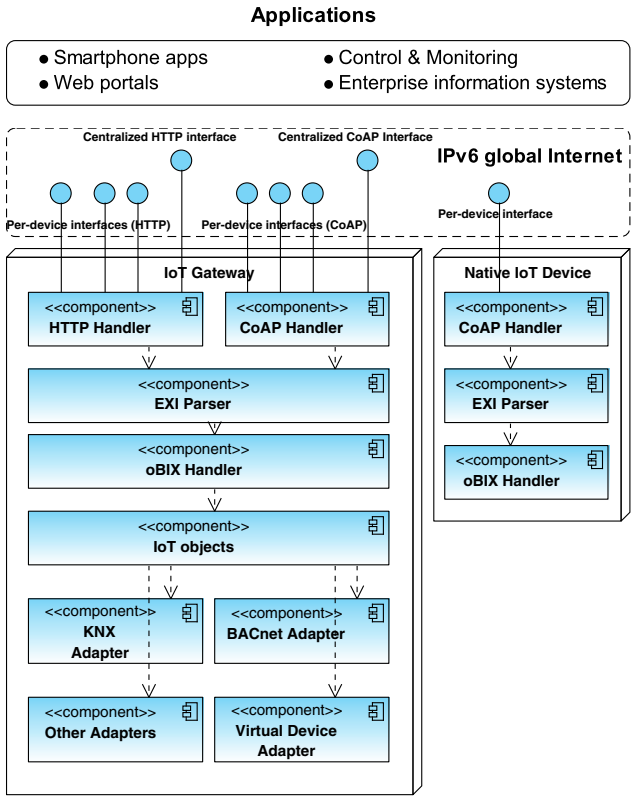


**Fig. 9:** Multi-protocol gateway architecture

specific concepts to the generic objects adhering to the IoT oBIX contracts has to happen there. These contracts allow to map various technologies into a common object oriented representation, which can act as a connecting element to an ontology based on semantic technologies.

The oBIX handler takes care of read, write and invoke

requests to the oBIX objects. In general it is independent of the protocol that is used to interact with the objects. It also manages oBIX watches that are used to monitor changes of objects. It uses the observer pattern to realize this functionality. In case of an CoAP `get` request with an observe option, it also takes care to send asynchronous responses if changes occur. For clients both ways are possible to observe a resource, either the traditional oBIX polling or the CoAP observe approach or both.

Furthermore the oBIX handler publishes the represented devices through the traditional oBIX lobby which can be easily mapped to the CoAP `/.well-known` listing of available resources. Beside this centralized device access approach, the oBIX handler also publishes each device using a separate HTTP and CoAP handler on a per-device IPv6 address. In this way it is possible to have a centralized access and per-device access in parallel, since requests to all endpoints will finally be handled by the same oBIX objects.

The EXI compression can happen transparently between the HTTP and CoAP handler and the oBIX handler if no schema information is used for compression. The oBIX handler only works on XML representation.

Finally, multiple HTTP and CoAP handlers are provided and bound to virtual and physical network interfaces. An HTTP and a CoAP handler can be provided offering a centralized interface conforming to the traditional oBIX approach to interact with devices. In the case of HTTP this is fully compliant to the oBIX standard, at the same time per-device interfaces can be offered with HTTP and CoAP. The HTTP interfaces remain oBIX compliant and the CoAP interfaces are compliant to the CoRE standards. An exception is the `observe` support for CoAP `get` requests and the CoAP binding as well as the use of EXI for oBIX. However, it is argued that these modifications do not violate the initial intentions of the standard, but rather enhance it. This is, for example, evidenced by standard-compliant clients still being able access all devices.

### *B. Interaction example*

For the illustration of the interaction, a simple scenario featuring a sensor and an actuator is taken. For the sensor, a push button is used as example. It can be turned on or off and controls an actuator which is represented by a light switching actuator. The latter device can be used to switch an electric circuit, possible powering one ore multiple lights.

The IoT contracts define the data points offered by these devices and are provided in Listing 1 and 2. The contracts are quite similar, since both offer a single boolean datapoint, but in case of the actuator it is also writeable. For more sophisticated devices there would be more data points, e.g. a dimming push button or a dimming actuator would also have an integer property with the current setting.

**Listing 1:** Light switching actuator contract

```
<obj href="iot:LightSwitchActuator" is="iot:Actuator">
      <bool name="value" href="value" val="false" writable="
          true"/>
</obj>
```

**Listing 2:** Push button contract

```
<obj href="iot:PushButton" is="iot:Sensor">
      <bool name="value" href="value" val="false"/>
</obj>
```

Assume a real push button and light switch actuator are bound to the context paths `[Central IPv6 address]/lightSwitch` and `[Central IPv6 address]/pushButton` at the centralized IPv6 interface as well as directly to the context root for the virtual per-device IPv6 `[IPv6 address light switch]/` and `[IPv6 address push button]/`. For these devices multiple interactions are possible with application clients. First of all the objects can be read through a HTTP `get` or CoAP `get` resulting in a response equivalent to the above representation. In a centralized approach the `get` can be sent to `[Central IPv6 address]/lightSwitch`, otherwise the request can be directly performed using the per-device IPv6 address.

A response to such a request returns the full extent of the object, which is in most cases not required. Since the boolean data point is also a oBIX object, it is possible to add the name of the data point to the request path. This results in either a `lightSwitch/value` or just a `/value` request.

Writing on an object, e.g. switching the actuator, is achieved using `put` requests in oBIX. This can be done either using the full object or just a data point object as given in Listing 3 and 4, respectively. As it can be seen, using a `put` on a data point is characterized by a much simplified payload.

**Listing 3:** Write on full object

```
PUT http://[Central IPv6 address]/lightSwitch

<obj href="http://[Central IPv6 address]/lightSwitch" is="
     iot:LightSwitchActuator">
  <bool name="value" href="value" val="true" writable="true"
       />
</obj>
```

**Listing 4:** Write on data point

```
PUT http://[Central IPv6 address]/lightSwitch/value

<bool val="true"/>
```

Now if an application is interested in the change of the light switching actuator or in the action of pressing the button there are two possibilities. Firstly, a watch object can be created providing a per-client state of changes to the object. Conforming to oBIX, the requests and responses listed in Listing 5 and 6 are required.

**Listing 5:** Create oBIX watch object

```
POST http://[Central IPv6 address]/watchService/make

Response:
<obj href="http://[Central IPv6 address]/watch0" is="obix:
     Watch">
  <op name="add" in="obix:WatchIn" out="obix:WatchOut"/>
  <op name="remove" in="obix:WatchIn" out="obix:Nil"/>
  <reltime name="lease" val="PT60S"/>
  <op name="pollChanges" in="obix:WatchIn" out="obix:
       WatchOut"/>
```

```
  <op name="pollRefresh" in="obix:WatchIn" out="obix:
      WatchOut"/>
  <op name="delete" in="obix:Nil" out="obix:Nil"/>
</obj>
```

**Listing 6:** Add object to watch

```
POST http://[Central IPv6 address]/watch0/add
Payload:
<obj is="obix:WatchIn">
  <list name="hrefs">
    <uri val="/lightSwitch" />
  </list>
</obj>

Response:
<obj is="obix:WatchOut">
  <list>
    <obj href="http://[Central IPv6 address]/lightSwitch" is=
        "iot:LightSwitchActuator">
      <bool name="value" href="value" val="false" writable="
          true"/>
    </obj>
  </list>
</obj>
```

The client has to poll for changes using the `pollChanges` operation provided by oBIX, which provides the same response if some data point or sub object of the object has changed. The same procedure can be done only on the data point object itself in order to be more efficient regarding message size. This interaction can be bound easily to CoAP using the same method calls and a request/response interaction. However, since polling is not a satisfactory way of being notified about changes, a CoAP `get` request using the `observe` option is more convenient. This request can bypass the creation of watches and directly subscribe to oBIX objects observing them as shown in Listing 7.

**Listing 7:** CoAP observe to oBIX object

```
GET [Centralized IPv6 address]/lightSwitch/value
Observe: 0
Token: 0x44

Response:
2.05 Content
Observe: 10
Token: 0x44
Payload:
<bool name="value" href="value" val="true" writable="true"/>

2.05 Content
Observe: 20
Token: 0x44
Payload:
<bool name="value" href="value" val="false" writable="true"/
    >
```

Control applications can use the observe functionality to link various IoT objects. E.g. a push button can be linked to a light switching actuator independent of the underlying technology and there is no difference between legacy devices and native IoT technologies.

### C. Implementation

The implementation is based on an existing open source oBIX server for KNX[5]. Several modifications to this server have been performed in order to implement the described changes to oBIX as well as to realize virtual IPv6 interfaces on a per-device level. The oBIX implementation is based on the oBIX toolkit[6]. As HTTP server the NanoHTTP server[7] version 1.25 is used. As CoAP server the open source implementation Californum[8] is taken. OpenEXI[9] is used between the oBIX handler and the server components to encode and decode the payloads. The current proof of concept implementation supports KNX and virtual devices. For KNX the Java library Calimero[10] is used. For the simple interaction scenario a small testbed based on a KNX push button and switching actuator as shown in Figure 10 is used. For the creation of virtual IPv6 interfaces a simple Linux shell script is triggered from the Java application.

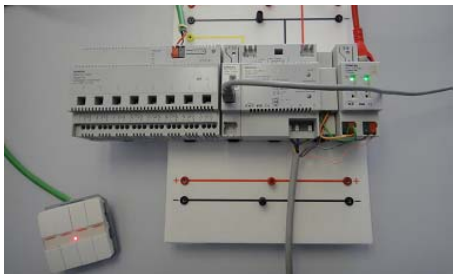

**Fig. 10:** KNX Testbed

### D. Evaluation

For the evaluation the above mentioned interaction scenarios are tested using the Firefox plugins Copper[11] and HttpRequester[12] to interface the multi-protocol gateway. By using Californum as CoAP server and Copper as CoAP client the conformance of the multi-protocol gateway to the ETSI CoAP Plugtest[13] can be verified. The resulting message size of the interactions is analyzed using Wireshark[14] and the results are shown in Table I.

It can be seen that the extension to oBIX using a CoAP protocol binding and EXI are significant and the payload can be reduced below the important limit of 127 Bytes, which is optimal for 6LoWPAN data links. Furthermore, using CoAP and the *observe* service heavily reduces the required network traffic and ensures real-time updates of changed information.

## VI. Conclusion & Further Work

The integration of legacy systems in the Internet of Things is a challenging task. This paper presents a concept and evaluation results of a seamless integration approach using a transparent multi-protocol gateway that provides a traditional

[5]https://www.auto.tuwien.ac.at/a-lab/knx2obix.html, Accessed: 30.07.2012

[6]http://sourceforge.net/projects/obix/, Accessed: 30.07.2012
[7]http://elonen.iki.fi/code/nanohttpd/, Accessed: 30.07.2012
[8]http://people.inf.ethz.ch/mkovatsc/californium.php, Accessed: 30.07.2012
[9]http://openexi.sourceforge.net/, Accessed: 30.07.2012
[10]http://calimero.sourceforge.net/, Accessed: 30.07.2012
[11]http://people.inf.ethz.ch/mkovatsc/copper.php, Accessed: 30.07.2012
[12]https://addons.mozilla.org/de/firefox/addon/httprequester/, Accessed: 30.07.2012
[13]http://www.etsi.org/plugtests/coap/coap.htm, Accessed: 30.07.2012
[14]http://www.wireshark.org/

| Technology | | | oBIX(HTTP) | | oBIX(CoAP) | |
|---|---|---|---|---|---|---|
| Encoding | | | XML | EXI | XML | EXI |
| GET object | Conversation | | **1125** | **1017** | **315** | **250** |
| | Request | Protocol | 401 | 353 | 10 | 12 |
| | | Payload | 0 | 0 | 0 | 0 |
| | Response | Protocol | 81 | 88 | 6 | 6 |
| | | Payload | 175 | 108 | 175 | 108 |
| GET data point | Conversation | | **983** | **892** | **247** | **199** |
| | Request | Protocol | 333 | 285 | 16 | 18 |
| | | Payload | 0 | 0 | 0 | 0 |
| | Response | Protocol | 81 | 88 | 6 | 6 |
| | | Payload | 101 | 51 | 101 | 51 |
| PUT object | Conversation | | **1096** | **991** | **448** | **352** |
| | Request | Protocol | 372 | 331 | 8 | 12 |
| | | Payload | 136 | 104 | 134 | 104 |
| | Response | Protocol | 81 | 88 | 6 | 6 |
| | | Payload | 175 | 104 | 175 | 104 |
| PUT data point | Conversation | | **1042** | **938** | **264** | **213** |
| | Request | Protocol | 392 | 336 | 15 | 22 |
| | | Payload | 19 | 17 | 19 | 17 |
| | Response | Protocol | 81 | 88 | 6 | 6 |
| | | Payload | 101 | 46 | 101 | 46 |
| POST polling | Conversation | | **1205** | **1038** | **308** | **231** |
| | Request | Protocol | 501 | 453 | 23 | 27 |
| | | Payload | 0 | 0 | 0 | 0 |
| | Response | Protocol | 81 | 29 | 6 | 6 |
| | | Payload | 155 | 88 | 155 | 88 |
| CoAP observe | Conversation | | **n.a.** | **n.a.** | **255** | **205** |
| | Request | Protocol | n.a. | n.a. | 20 | 24 |
| | | Payload | n.a. | n.a. | 0 | 0 |
| | Response | Protocol | n.a. | n.a. | 10 | 9 |
| | | Payload | n.a. | n.a. | 101 | 46 |

All numbers provided in Bytes.
Conversation refers to data link frame size including request and response and in the case of TCP the connection open and close.
EXI uses Bit-aligned encoding without schema information.

**TABLE I:** oBIX protocol binding and encoding evaluation

oBIX interface and a per-device IPv6 CoAP interface conforming to the CoRE standards. The feasibility of a CoAP and EXI binding for oBIX is proven and the use of oBIX features on top of CoAP is identified as a performant solution. The integration approach may conform to the existing standards. However, to provide an optimal solution, the standards of oBIX and CoRE need to be extended. The presented concepts thus go beyond simply using oBIX on top of CoAP and rather represent a possible IoT protocol stack.

The main contributions of this paper may be summarized as follows:

- An overview of different integration styles that can be used to integrate BAS into the IoT.
- A CoAP and EXI protocol binding for oBIX.
- A mapping from oBIX to CoRE and modification towards a per-device IPv6 interface for each oBIX object.
- The requirements to define a standardized IoT ontology comparable to traditional standardized functional blocks of existing BAS in order to achieve interworking and improved EXI compression.

As further work we consider to publish the multi-protocol gateway as open source project and to support further BAS like BACnet, ZigBee, EnOcean and Wireless M-Bus. Furthermore we want to investigate service discovery and commissioning facilities that fit for a large scale system like the IoT, to address the security and privacy challenges and to close the semantic gap between IoT oBIX contracts and ontologies.

## ACKNOWLEDGEMENTS

Authors express their acknowledgement to the consortium of the project IoT6 (www.iot6.org). The IoT6 project is supported by funding under the Seventh Research Framework Program of the European Union, with the grant agreement FP7-ICT-2011-7-288445.

## REFERENCES

[1] "OPC unified architecture specification," OPC Foundation, 2009.
[2] "BACnet – a data communication protocol for building automation and control networks," ANSI/ASHRAE 135, 2010.
[3] "oBIX 1.1 draft committe specification," OASIS, 2011.
[4] Z. Shelby and C. Bormann, *6LoWPAN: The Wireless Embedded Internet*. Wiley, 2011.
[5] Z. Shelby, "Embedded Web Services," *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 52–57, 2010.
[6] A. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, "Web Services for the Internet of Things through CoAP and EXI," in *IEEE International Conference on Communications (ICC)*, 2011.
[7] G. Moritz, F. Golatowski, and D. Timmermann, "A Lightweight SOAP over CoAP Transport Binding for Resource Constraint Networks," in *Proc. of the 8th IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, 2011.
[8] D. van der Linden, W. Granzer, and W. Kastner, "OPC Unified Architecture (OPC UA) new opportunities of system integration and information modelling in automation systems," in *Proc. of the 9th IEEE International Conference on Industrial Informatics (INDIN)*, 2011.
[9] W. Kastner and S. Szucsich, "Accessing KNX networks via BACnet/WS," in *Proc. of the IEEE International Symposium on Industrial Electronics (ISIE '11)*, 2011.
[10] M. Neugschwandtner, G. Neugschwandtner, and W. Kastner, "Web Services in Building Automation: Mapping KNX to oBIX," in *Proc. of the 5th IEEE International Conference on Industrial Informatics (INDIN '07)*, 2007.
[11] M. Jung, C. Reinisch, and W. Kastner, "Integrating Building Automation Systems and IPv6 in the Internet of Things," in *Proc. of the 6th IEEE International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS' 12)*, 2012, in Press.
[12] D. Guinard and V. Trifa, "Towards the Web of things: Web mashups for embedded devices," in *Proc. of the International World Wide Web Conferences, Madrid, Spain*, 2009.
[13] D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards physical mashups in the Web of things," in *Proc. of the 6th International Conference on Networked Sensing Systems (INSS)*. IEEE, 2009.
[14] C. Reinisch, W. Granzer, F. Praus, and W. Kastner, "Integration of Heterogeneous Building Automation Systems using Ontologies," in *Proc. of the 34th IEEE Conference Industrial Electronics Society (IECON '08)*, 2008.
[15] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, M. Pagel, M. Hauswirth, M. Karnstedt *et al.*, "SPITFIRE: toward a semantic Web of things," *Communications Magazine*, vol. 49, no. 11, pp. 40–48, 2011.
[16] (2012, Jul.) The Universal Device Gateway. [Online]. Available: http://www.devicegateway.com/
[17] W. Granzer, W. Kastner, and C. Reinisch, "Gateway-free integration of BACnet and KNX using multi-protocol devices," in *Proc. of the 6th IEEE International Conference on Industrial Informatics (INDIN '08)*, 2008.
[18] "Constrained Application Protocol (CoAP) draft-ietf-core-coap-12," IETF Internet Draft, Oct. 2012.
[19] "SensorML version 1.0.1," OCG, 2007.
[20] F. Furfari, L. Sommaruga, C. Soria, and R. Fresco, "DomoML: the definition of a standard markup for interoperability of human home interactions," in *Proc. of the 2nd European Union Symposium on Ambient Intelligence (EUSAI '04)*. ACM, 2004.