Universal Integration of the Internet of Things through an IPv6-based Service Oriented Architecture enabling heterogeneous components interoperability

Grant agreement for: Collaborative project
Grant agreement no.: 288445

Start date of project: October 1st, 2011 (36 months duration)

Deliverable D4.3
Multi-protocol integration report

<table>
<thead>
<tr>
<th>Contract Due Date</th>
<th>28/02/2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submission Date</td>
<td>07/04/2014</td>
</tr>
<tr>
<td>Version</td>
<td>v1.0</td>
</tr>
<tr>
<td>Responsible Partner</td>
<td>Vienna University of Technology, Austria</td>
</tr>
<tr>
<td>Author List</td>
<td>Markus Jung</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>PU</td>
</tr>
<tr>
<td>Keywords</td>
<td>Internet of Things, IPv6</td>
</tr>
</tbody>
</table>

Project Coordinator: Sébastien Ziegler
Mandat International (MI)
sziegler@mandint.org
# Table of Contents

[1] Introduction ............................................................................................................... 6

  2.1 Integration approach ................................................................................................... 6
  2.2 Communication principles of integrated technologies ............................................... 6
      2.2.1 Messaging Paradigms ........................................................................................... 7
      2.2.2 Data-point centric vs. command oriented communication ..................................... 7
  2.3 Multi-protocol integration – protocol and integration alternatives ............................ 8
  2.4 Multi-protocol integration architecture ...................................................................... 9
  2.5 IoT6 stack overview ................................................................................................... 11
      2.5.1 Application Layer ................................................................................................ 11
      2.5.2 Message Exchange ............................................................................................. 13
      2.5.3 Message Encoding .............................................................................................. 17
      2.5.4 Network Layer and Addressing .......................................................................... 17
      2.5.5 Data Link Layer ................................................................................................... 19
  2.6 Service discovery ....................................................................................................... 19

  3.1 KNX mapping to IoT6 stack ........................................................................................ 20
      3.1.1 Addressing .......................................................................................................... 20
      3.1.2 Transport layer .................................................................................................... 21
      3.1.3 Application Layer ................................................................................................ 22
      3.1.4 Mapping KNX group communication to IoT6 stack RESTful interaction .......... 22
      3.1.5 Mapping KNX data point types ........................................................................... 24
      3.1.6 Automated mapping of KNX devices .................................................................. 25
  3.2 BACnet ....................................................................................................................... 26
      3.2.1 Addressing .......................................................................................................... 26
      3.2.2 Transport Layer ................................................................................................... 27
      3.2.3 Application Layer services .................................................................................. 27
      3.2.4 Mapping BACnet objects .................................................................................... 28
      3.2.5 Automated mapping of BACnet devices ............................................................ 32
  3.3 ZigBee ........................................................................................................................ 33
      3.3.1 Addressing .......................................................................................................... 33
      3.3.2 Application layer ................................................................................................. 34
  3.4 EnOcean ..................................................................................................................... 35
      3.4.1 Addressing .......................................................................................................... 35
      3.4.2 Application layer services ................................................................................... 35

[4] Smart grid technologies ................................................................................................. 37
  4.1 Wired M-Bus .............................................................................................................. 37
  4.2 Wireless M-Bus .......................................................................................................... 37
      4.2.1 Addressing .......................................................................................................... 37
      4.2.2 Messaging .......................................................................................................... 39
      4.2.3 Application Layer ............................................................................................... 39
  4.3 Mapping Wireless and Wired M-Bus to the IoT6 stack ............................................. 41

[5] Identification technologies and other information sources .......................................... 42
5.1 RFID ............................................................................................................................ 42
  5.1.1 RFID address format ........................................................................................... 42
  5.1.2 Mapping RFID tag reader to IoT6 stack .............................................................. 42
5.2 Weather data ............................................................................................................. 42
  5.2.1 Weather data source .......................................................................................... 42
  5.2.2 Mapping weather data to the IoT6 stack ........................................................... 43
[6] IoT6 stack on constrained devices ............................................................................. 44
[7] IoT6 gateway implementation .................................................................................. 44
  7.1 IoT6 gateway architecture ......................................................................................... 44
  7.2 Integration with local control and monitoring system .............................................. 46
  7.3 Implementation ......................................................................................................... 48
  7.4 Gateway configuration and device discovery ........................................................... 48
  7.5 Protocol bundles ........................................................................................................ 49
    7.5.1 KNX protocol bundle .......................................................................................... 49
    7.5.2 BACnet protocol bundle ..................................................................................... 51
    7.5.3 Wireless M-Bus protocol bundle ........................................................................ 52
    7.5.4 Other protocol bundles ...................................................................................... 56
  7.6 Local HTML5 control interface ................................................................................... 56
[8] Deployment ............................................................................................................... 57
  9.1 Multi-protocol interaction tests ................................................................................ 59
    9.1.1 Lighting control ................................................................................................... 59
    9.1.2 HVAC control ...................................................................................................... 60
    9.1.3 Alarming ............................................................................................................. 60
    9.1.4 Access control and room adjustment ................................................................. 60
  9.2 Test environment ....................................................................................................... 60
    9.2.1 KNX equipment .................................................................................................. 61
    9.2.2 BACnet HVAC process model and Wireless M-Bus smart meter ....................... 63
    9.2.3 Wired M-Bus floor heating model ...................................................................... 65
    9.2.4 EnOcean wireless room automation .................................................................. 65
    9.2.5 6LoWPAN test bed .............................................................................................. 66
    9.2.6 Mobile IoT6 test bed .......................................................................................... 67
  9.3 Manual gateway interaction test ............................................................................... 68
    9.3.1 Device Discovery ................................................................................................. 69
    9.3.2 Query sensor/actuator ......................................................................................... 69
    9.3.3 Modify actuator ................................................................................................... 69
    9.3.4 Further interaction scenarios ............................................................................. 69
  9.4 Multi-protocol interaction use case scenarios tests .................................................. 70
    9.4.1 Lighting control scenario .................................................................................... 70
    9.4.2 HVAC control scenario ........................................................................................ 70
    9.4.3 Alarming scenario ............................................................................................... 71
    9.4.4 Access control and room environment settings scenario .................................... 72
[10] Conclusion .................................................................................................................. 73
[11] References ................................................................................................................. 74
Figures

Figure 1 Multi-protocol integration approach alternatives ......................................................... 6
Figure 2 Integration challenge .................................................................................................... 8
Figure 3 Relationship of common IP protocols [2] ..................................................................... 9
Figure 4 Multi-protocol integration architecture ...................................................................... 10
Figure 5 Multi-protocol stack .................................................................................................. 11
Figure 6 oBIX object model ...................................................................................................... 12
Figure 7 IoT6 generic object abstraction .................................................................................. 13
Figure 8 Group communication interaction scenarios ............................................................. 15
Figure 9 Individual address ....................................................................................................... 20
Figure 10 Group address .......................................................................................................... 20
Figure 11 KNX interworking model example [5] .................................................................... 23
Figure 12 Mapping KNX function blocks to oBIX contracts [14] ............................................. 24
Figure 13 KNX data type structure [5] ..................................................................................... 24
Figure 14 KNX Main Data Point Type 1 [5] .............................................................................. 24
Figure 15 BACnet collapsed architecture .............................................................................. 26
Figure 16 BACnet frame format .............................................................................................. 26
Figure 17 BACnet address length depending on media type ..................................................... 26
Figure 18 BACnet object identifier ......................................................................................... 27
Figure 19 OBIX object hierarchy of mapped BACnet types ..................................................... 29
Figure 20 ZigBee stack protocol stack [16] ............................................................................. 33
Figure 21 Clusters support by On/Off switch of home automation profile ............................... 34
Figure 22 ISO/IEC 14543-3-10 protocol architecture (WSP stack) [8] .................................... 35
Figure 23 Wireless M-Bus communication ............................................................................. 38
Figure 24 Wireless M-Bus example telegram ......................................................................... 39
Figure 25 COSEM object oriented model .............................................................................. 40
Figure 26 M-Bus data structure container .............................................................................. 41
Figure 27 oBIX contract for representing a smart meter ......................................................... 41
Figure 28 oBIX contract for representing RFID reader .......................................................... 42
Figure 29 oBIX Mapping of Weather Data ............................................................................ 43
Figure 30 Gateway architecture ............................................................................................. 45
Figure 31 Deployment of protocol bundles within the control and monitoring system (CMS) .... 46
Figure 32 IoT application module for the control and monitoring system .............................. 47
Figure 33 KNX protocol bundle internals ............................................................................. 50
Figure 34 CoAP based GET on KNX temperature sensor ....................................................... 51
Figure 35 Observe relationships between a CoAP client and the KNX bus device ................. 51
Figure 36 Package diagram illustrating the schematic overview of API .................................. 52
Figure 37 Class overview for parsing W-MBus telegramms .................................................... 54
Figure 38 Telegram manager class diagram .......................................................................... 55
Figure 39 W-MBus integration ............................................................................................... 56
Figure 40 HTML5 control logic editor .................................................................................... 57
Tables

Table 1 Group communication payload examples ................................................................. 15
Table 2 IPv6 multicast address format .................................................................................. 18
Table 3 IPv6 multicast address format for 6LoWPAN .......................................................... 18
Table 4 Mapping KNX application services to the IoT6 stack ............................................. 22
Table 5 Mapping KNX data point types to oBIX value object types [14] ................................. 25
Table 6 ZigBee addressing overview ..................................................................................... 34
Table 7 EnOcean Equipment Profiles telegram types ........................................................ 36
Table 8 M-Bus protocol stack ................................................................................................. 37
Table 9 oBIS code structure .................................................................................................. 40
[1] Introduction

This report will present the result of the integration efforts within work package 4 of the IoT6 project. It will provide insights into the taken integration approach sketch the software implementation and show test results of device communication across selected protocol borders.


This section will provide an overview of the taken integration approach, the communication principles of the integrated technologies, the overall multi-protocol integration architecture, the layers and features of the IoT6 stack and the mapping of the integrated technologies to this novel communication stack.

2.1 Integration approach

As stated in deliverable D4.1, for the integration of the heterogeneous protocols two different approaches can be taken: either technologies are mapped in an n-to-n manner or technologies are mapped to a well-defined target system stack (n-to-1 mapping). Within the IoT6 project, a target system stack has been specified which is further named as IoT6 stack. Also defined in deliverable D4.1, the integration challenge is to find an appropriate mapping of existing technologies to the novel communication stack.

2.2 Communication principles of integrated technologies

This subsection provides an overview of the communication principles of the heterogeneous automation technologies that need to be integrated. For the technologies to be integrated, numerous physical media can be found. Historically standardized wired media are used adhering to certain quality requirements regarding bandwidth, range, transmission quality and costs where in particular the latter can be seen as major influence that affects most of the design choices for the upper layers of a communication system. More recently, also wireless media became more important for automation technologies. Since bus systems follow a multi-tiered approach where the automation and field level usually cover a local control segment, the network layer of these technologies is kept rather simple or sometimes does not exist. The rationale behind this is to keep the frame size of the packets small by having short protocol headers and, thus, to reduce the overhead. On the other hand, it is aimed at keeping the memory requirements of a stack implementation small in order to optimize the device costs. Addressing in such systems happens on multiple layers but is mainly focused on the data link layer due to the lack of sophisticated network layer capabilities. Typically, the application layer services directly reside on data link layer concepts. This is required due to the optimization but sometimes breaks the OSI layer model principle of having dependencies to lower layers of the protocol. Due to these design flaws
and the different network layer protocols it is not easily possible to map the application layers of these technologies to an IPv6 based network. For example, BACnet defines a virtual link layer for IPv6 that degrades an IPv6 network to the functionality of a data link layer. Therefore, the only way to integrate these technologies and to provide true interoperability is to use a full application layer gateway that maps all layers of a certain system to a new protocol stack.

### 2.2.1 Messaging Paradigms

For the application layer within existing technologies various communication forms can be found. On the one hand there is unicast based message exchange following a client/server based communication, on the other hand group communication based on multicasting or broadcasting following a publish/subscribe and producer/consumer pattern can be detected. For unicast based communication, a message is exchanged between only two interacting devices in which one device acts as a client and sending the message and its counterpart acts as server receiving the message. Unicast based communication can be further divided into synchronous and asynchronous communication. Within synchronous communication the client process blocks until it receives an answer of the server. For asynchronous communication, the client request is non-blocking and the answer needs to be expected in a later response message. A further communication principle is request/response centric interaction in which a server might only send a response message if requested by the client. Some systems rely on group communication principles in which data is exchanged based on group addresses that link together different I/O signals and virtual data variables of devices. Group communication relies either on broadcast mechanisms or more efficient multicasting mechanisms. Group communication can be used to realize the concept of a network shared variable in which a certain data variable is linked to a group address. Devices can synchronize the state of this variable by relying on the group communication mechanism. This can be seen closely related to a publish/subscribe communication paradigm in which a message broker is used to exchange messages for a certain topic, following a more data-centric message exchange. However, the publish/subscribe communication principle can be realized with unicast based client/server communication having a centralized message broker or based on group communication without the need of a centralized message broker.

### 2.2.2 Data-point centric vs. command oriented communication

A very important concept in the domain of home and building automation technologies or wireless sensor and actuator networks is the concept of data-point centric information modeling combined with simple read and write communication services. In this way the system is modeled using basic data types and the interaction between devices is reduced to a simple set of basic standard communication services that need to be supported by all devices to ensure interoperability. A different approach is to use a command centered modeling in which an object-oriented approach is taken.
2.3 Multi-protocol integration – protocol and integration alternatives

For the definition of the layers above IPv6 multiple alternatives are available. Application layer protocols deal with the message exchange and methods in the context of inter-process communication. The application layer relies upon the transport layer which provides host-to-host communication and port multiplexing allowing multiple processes to communicate simultaneously. For the transport layer above IPv6, UDP and TCP are the main available alternatives. While UDP provides best-effort communication and TCP supports reliable connection-oriented multiplexed communication between application processes. Due to issues with compression and the unreliability of low-power and lossy wireless networks (LLNs), TCP does not fit well as transport layer. Therefore, UDP is the preferable transport layer protocol for an IPv6 based communication stack that should suit for constrained devices found in the Internet of Things. Further, it also opens the possibility of having group communication mechanisms based on IPv6 multicasting which is an important aspect that is going to be discussed in the remainder of this document.

The application protocol resides on top of the transport layer. Figure 3 provides an overview of application layer protocols that are suitable to be used within 6LoWPANs. The selection is taken from [1] and is not an exhaustive list of possible protocols, but rather a selection of most suitable and meaningful protocols. Since the selection is from 2008 several new protocols came up in the recent years and will be discussed in the following sub-section. For the Internet, HTTP provides a key protocol for the World Wide Web to exchange documents. It has also become quite popular within enterprise systems in order to replace proprietary integration approaches based on distributed object middleware or remote-procedure call frameworks through Web Services relying on open protocols and message encodings. Beside Web service based protocols, important IP-based application protocols are for example the file transfer protocol (FTP), real-time protocol (RTP), session initiation protocol (SIP), service location protocol (SIP), service location protocol (SLP) and the simple network management protocol (SNMP). For the use within 6LoWPANs, the application layer protocols like the ZigBee Constrained Application Protocol (CAP), the IBM Message Queuing Telemetry Transport (MQTT) and the Open Building Information Exchange (oBIX) provide the required semantic capabilities to represent the information model of smart objects in the Internet of Things and to have standardized message structures.
This is especially important when it comes to the integration of heterogeneous technologies of domains like home and building automation or smart grids. These domain specific systems stacks such as the ones based on BACnet and KNX need to be integrated and a common abstraction needs to be found. For the domain of home and building automation, oBIX is a reasonable choice to provide a common abstraction and communication interface.

As stated in [1]:

“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. Careful design of the oBIX objects and elements used would be important to keep packet sizes reasonable”

This approach was taken for the design of the IoT6 stack. Besides providing an efficient protocol binding to UDP/IP the approach goes further and also uses oBIX as application layer protocol to interact with smart objects and also for the integration of technologies and information sources not related to home and building automation systems.

### 2.4 Multi-protocol integration architecture

The IoT6 architecture addresses the heterogeneity of existing technologies in the Internet of Things environment. For providing interoperability amongst heterogeneous technologies and also for new Internet of Things devices directly communicating using IPv6, the so-called IoT6 stack provides a common message format, message exchange protocol and vocabulary to ensure interoperability. Within the IoT6 architecture several components are responsible to ensure multi-protocol interoperability. The IoT6 gateway component, which can be deployed on the smart board or within a local control and monitoring system, provides an integration middleware for existing relevant technologies. It implements a mapping of the IoT6 stack to each technology that shall be integrated and offers a communication interface according to the IoT6 stack for devices of these technologies. Similarly, the stack can be deployed on constrained devices within a wireless sensor and actuator network (WSAN). The core elements of the IoT6 stack are IPv6, Web services based on CoAP [3] and the object model of the OASIS Open Building Information Exchange standard (oBIX) [4]. IPv6 provides the
common networking layer for addressing devices and for end-to-end connectivity between information systems and Internet of Things devices. Web services support an interoperable and platform independent way to exchange messages by using the Constrained Application Protocol as protocol for message exchange and XML or JSON for message encoding. To have a common vocabulary and language for the exchanged information the object model of oBIX provides a common meta-model to define information models for the Internet of Things. By adhering to this integration approach a unified communication interface for the other components, such as cloud computing, mobile computing, STIS or service discovery, of the IoT6 architecture can be provided. Figure 4 illustrates the overall architecture of the multi-protocol integration approach. Existing technologies that have been integrated within the scope of the IoT6 project are KNX [5], BACnet [6], ZigBee [7], EnOcean [8] out of the domain of home and building automation area. Wired and Wireless M-Bus [9] are integrated as representatives of the upcoming smart meter technologies, which are an essential infrastructure part of smart grids. RFID is integrated by incorporating RFID readers.

Figure 4 Multi-protocol integration architecture

The concept of the multi-protocol integration architecture has been presented in [10] and also demonstrated to the scientific community [11]. For realizing the multi-protocol interoperability the OSGi framework is used and protocol bundles for the relevant
technologies have been implemented. The protocol bundles can be deployed on the smart board, providing together with the gateway components a light-weight access to the heterogeneous technologies through the IoT6 stack. Another deployment scenario of the protocol bundles is directly within the control and monitoring system (CMS). The different deployment scenarios will be discussed in detail in the implementation section.

### 2.5 IoT6 stack overview

The aim of the IoT6 stack is to provide a common communication interface amongst all technologies found in the Internet of Things environment. Through a mapping to existing technologies the multi-protocol gateway offers a uniform interface for all technologies behind the gateway based on the protocol stack illustrated in Figure 5. This stack may also be directly deployed on embedded devices.

#### 2.5.1 Application Layer

On top of the protocol stack, oBIX is used as application layer protocol since CoAP does not provide the required features to map all the application layer functionalities of existing building automation systems (BAS). With oBIX, an information model is delivered representing devices and entities out of the domain of building automation as oBIX objects. Further a simple protocol aligned to RESTful interaction is offered. The functionality includes interaction to query and modify objects and to invoke oBIX operations. To this end, three protocol services are offered (read, write and invoke) that are mapped to the HTTP and CoAP methods `get`, `put` and `post`. The oBIX object model (see Figure 6) follows the typical approach of representing all devices as objects holding collections and references to other objects and collections of data points with a specified data type. oBIX uses the concept of contracts to define certain object types. Currently, 17 standard object types are defined, ranging from basic data items like `bool`, `int`, `real` and `str` to more complex object types. oBIX contracts also provide semantics for standard device types. Special contracts for observing resources, alarms and histories based on the object contracts for watches, alarms and histories are supported.

Within our protocol stack, contracts are used to model standard device types found in building automation systems and wireless sensor networks. These so called IoT contracts provide a generic device representation that can be used on CoAP enabled devices or at gateways for existing non-IP technologies directly. Contracts also define standardized service interfaces. The oBIX core library comes with a set of standard contracts including a watch
service, histories and alarming.

The semantic description of oBIX contracts is only intended for human beings but not usable for machine based semantic processing. In this regard, it can be compared to function blocks of BAS that describe standardized behaviors of sensors, actuators and control devices and define the semantics of input and output data points of devices that implement the function blocks.

The application layer of the IoT6 stack is mainly built around the OASIS Open Building Information Exchange. For conveying the application layer semantics various different protocols and systems stacks can be used.

**IoT6 oBIX contracts**

For the IoT6 stack, a set of generic oBIX contracts has been defined. oBIX contracts allow a common abstraction amongst heterogeneous technologies and devices. Further, they can be used to define platform specific types in an object-oriented type system. A thing is the most basic abstraction and further sub-divided into a sensor and actuator, which provides a capability in a certain domain (e.g. light switch actuator).
Below is a simple selection of example oBIX contracts provided, which define a generic abstraction amongst heterogeneous technologies. The device abstraction ranges from a set of simple devices with only a single data point up to more complex mappings in which devices provide further data points and operations.

```xml
<obj href="iot:TemperatureSensor">
  <real name="value" href="value" val="0.0" unit="obix:units/celsius"/>
</obj>

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

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

<obj is="iot:Cooler">
  <bool name="enabled" href="enabled" val="false" writable="true"/>
</obj>

<obj is="iot:Boiler">
  <bool name="enabled" href="enabled" val="false" writable="true"/>
</obj>

<obj is="iot:Pump">
  <int name="value" href="value" val="0" writable="true" min="0" max="100"/>
</obj>

<obj is="iot:FanSpeedActuator">
  <int name="fanSpeedSetpointValue" href="fanSpeedSetpoint" val="0" writable="true" unit="obix:units/percent"/>
  <bool name="enabled" href="enabled" val="false" writable="true"/>
</obj>

<obj is="iot:HumiditySensor">
  <real name="value" href="value" val="50.0" unit="obix:units/percent"/>
</obj>

<obj is="iot:LightIntensitySensor">
  <real name="value" href="value" val="1000.0" unit="obix:units/lumen"/>
</obj>

<obj is="iot:SunblindActuator">
  <bool name="moveDownValue" href="sunblindMiddleA/moveDownValue" val="false" writable="true"/>
  <bool name="moveUpValue" href="sunblindMiddleA/moveUpValue" val="false" writable="true"/>
</obj>
```

**Listing 1 Selection of IoT6 oBIX contracts**

### 2.5.2 Message Exchange

oBIX defines a protocol binding to HTTP for message encoding (XML or custom binary representation). For message exchange, the gateway offers either HTTP or CoAP. Both are in
general quite similar, but HTTP uses TCP as underlying transport layer protocol. This provides reliability but limits the communication to a connection oriented point-to-point communication and can be considered as heavy-weight communication protocol for sensor networks regarding bandwidth and computational resources of nodes. Furthermore, HTTP only allows a request/response interaction using a client/server communication model. In contrast to that, CoAP uses UDP as transport layer. This provides unreliable packet-oriented communication with group communication and asynchronous interaction within the client/server communication model. Due to these differences, CoAP provides means for non-confirmed and confirmed message exchange and furthermore extends the regular HTTP protocol with an observe verb\(^1\). The enhancement supports observing a resource and avoids frequent polling of resources such as event streams or alarms. Further, it is possible to rely on IPv6 multicasting for group communication mechanism [12].

### 2.5.2.1 CoAP based group communication mechanism

The IPv6 multicast addresses can be assigned to basic oBIX value object types that are adhering to the `obix:Point` contract. Only basic value object types (e.g. `int`, `real`, `bool`) can provide these data point semantics. For the group communication, a new contract `iot:groupComm` is introduced as illustrated in Listing 2.

```xml
<obj name="groupComm" href="groupComm/" is="iot:GroupComm">
  <list name="groups" href="groupComm/groups" of="obix:str" />
  <op name="joinGroup" href="groupComm/joinGroup" in="obix:str" out="obix:list" />
  <op name="leaveGroup" href="groupComm/leaveGroup" in="obix:str" out="obix:list" />
</obj>
```

**Listing 2 Group communication contract example**

The contract is kept simple. It supports to query the current groups of a data point and to invoke two operations which allow either to join or to leave a group. The operations take an IPv6 multicast address as `obix:str` object as input and return the current list of groups. For invoking the operation, the post method of CoAP or HTTP needs to be used.

The group communication object allows managing the assigned addresses at runtime. While some nodes might be configured statically with certain multicast addresses it is possible to add or remove IPv6 multicast addresses on certain data points at runtime. The feature of administrating the assigned group communication relationship at runtime depends on the platform and the available OS socket API.

**Group communication semantics**

To have meaningful group communication, only data points of the same value type or a value type that can easily be transformed to another value type (e.g. `int` to `real` and vice versa) should be put into a single group. Consider a sensor like a push button and an actuator like a switch actuator. Both have a value point represented by a `bool` object. Assume that both data points joined the group represented through the IPv6 multicast address FF12:1::1. Once a user presses the button, a physical signal is detected and the according oBIX object is updated. If an oBIX point is an input value, determined not having the writable attribute is set to true. In this case, the updated value is sent to all registered actuator devices which receive the multicast request and detect that the address is assigned to a local data point. The oBIX value is updated accordingly and triggers the physical signal for the switching

\(^1\) Observing Resources in CoAP draft-ietf-core-observe-05,” IETF Internet Draft, Mar. 2012.
actuator.

<table>
<thead>
<tr>
<th>Object type</th>
<th>CoAP payload example</th>
<th>XML bytes</th>
<th>EXI bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bool</td>
<td><code>&lt;bool val='false'/&gt;</code></td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>Int</td>
<td><code>&lt;int val='58'/&gt;</code></td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>Real</td>
<td><code>&lt;real val='58.12'/&gt;</code></td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td>Str</td>
<td><code>&lt;str val='hello world'/&gt;</code></td>
<td>24</td>
<td>15</td>
</tr>
</tbody>
</table>

*Table 1 Group communication payload examples*

**Group communication request format**

The group communication solely relies on IPv6 and non-confirmed CoAP put requests as recommended², in order to avoid congestion. This limits the application of the current solution to non-safety-critical use cases or use cases in which a human user is in the control loop and compensates any message failures. The IPv6 multicast group communication request is encapsulated into a UDP packet consisting of a CoAP header and the payload. The CoAP header takes 4 bytes. Additionally, 2 bytes are required to provide the content-format information. Since the IPv6 multicast address is directly linked to the value of a data point no further location path or query needs to be transmitted. For simple switching actions the UDP payload is below 10 bytes. In the case of LLNs and 6LoWPANs according to [1], as a rule of thumb, the UDP application payload has to be below 50-60 bytes in order to avoid fragmentation. The presented group communication request format stays well below this limit. For the group communication, the payload can be standard oBIX XML, but preferable it should be binary EXI encoding using the oBIX schema information. In this way, the payload can further be reduced from tens of bytes required to encode basic value objects to a few bytes as outlined for a selected number of value object types in Table 1.

**Group communication interaction scenarios**

For the group communication multiple interaction scenarios and use cases can be identified (cf. Figure 8).
**Grouping similar devices:** First of all, similar devices can be grouped together. For example, light switching actuators of rooms, floors and buildings can be arranged in groups aligned to the building topology and assigned to a separate IPv6 multicast address (e.g. FF12:1::1). Similarly a functional grouping including all devices can be done covering different domains (e.g. lighting/shading, HVAC) available at the local site.

**Process communication:** Besides sending requests to groups of similar devices it is also possible to use group communication for exchanging process data between sensors or actuators. This interaction pattern can be observed in state-of-the-art BAS like for example KNX. The example provided in Figure 8 shows how a sensor can be linked to an actuator based on an IPv6 multicast group address (e.g. FF12:1::2). Together with the highly efficient EXI encoding this interaction pattern is a reasonable choice for automation systems based on LLNs. Furthermore, if decentralized control is based on unicast messaging it provides the advantage of not having a centralized controller that might fail or congestion.

**Integration with existing BAS technology:** Another use case for the group communication can be identified together with the gateway approach, in which BAS devices of non-IP technologies (e.g. KNX, BACnet) are represented through a unique IPv6 per-device CoAP interface at the gateway. In this case, the group communication model can be used to interconnect native IPv6 and CoAP devices but also as integration middleware between KNX and BACnet for example. Furthermore, one use case for the integration with legacy technologies is the seamless integration of LLNs based sensors into existing HVAC control setups, in which IPv6 based CoAP devices can be used within BACnet control logic. As a simple case study, think of a BACnet control logic operating on hard data points like binary input/output and analog input/output objects and soft data points like a set point for a temperature control usually represented through binary or analog value objects. While the current room temperature might be represented through an analog input object this can be replaced through a soft data point represented through an analog value object which is in the same multicast group like a sensor in LLNs (e.g. FF15:1::3 with site-local scope). In this way, LLN sensors can complement existing automation systems and be seamlessly integrated into existing control logic.
2.5.3 Message Encoding

To improve the message encoding of XML based communication interfaces, EXI provides a standardized method to map XML data to a binary representation. The encoding is most efficient if an XML schema is offered. In this case, all communicating partners need to be aware of the schema. However, even if no schema is used, reasonable results regarding message size can be achieved and the problem of a tight coupling among service consumers and service providers is avoided. Alternatively, it is possible to use other encoding formats. For example, the JSON is a light-weight format that became quite popular recently but falls into the same encoding efficiency class as the XML format. The JSON format is a good choice if a direct browser-based communication to the Web service interfaces is desired, since its formatted objected can be directly parsed by a JavaScript interpreter without the need of a further message parsing library. Finally, for oBIX there is a definition for a binary message exchange format. This format is comparable in terms of efficiency to a schema-informed EXI encoding, but it requires a platform specific implementation only usable for the specific binary oBIX format.

2.5.4 Network Layer and Addressing

Since oBIX follows a RESTful system architecture, the addressing of entities rests upon URIs. A URI is based on a hierarchical sequence consisting of a scheme, authority, path, query and fragment as illustrated in the following figure.

\[
\text{URI} = \langle \text{scheme name} \rangle: \langle \text{hierarchical part} \rangle \ [? \langle \text{query} \rangle \] \ [\# \langle \text{fragment} \rangle]
\]

Listing 3 Hierarchical URI scheme

The scheme type defines the URI-type and is used to interpret the following content of the URI. If the scheme defines a protocol like http or coap then the URI can be taken as an URL that can be used to access the resource by the specified protocol. In case of the IoT6 stack the authority can be a DNS based hostname or an IPv6 address.

2.5.4.1 IPv6

IPv6 is a standard specified by the IETF in 1998. According to the specification, the primary changes concerning its antecessor IPv4 fall into the categories i) expanded addressing capabilities, ii) header format simplification, iii) improved support for extension and options, iv) flow labeling capability, and v) authentication and privacy. Changes that are of more interest for the IoT6 stacks are in the following list:

1. **Expanded addressing capabilities**: The address space of IPv6 is extended from 32 to 128 bit. The calculation provided in [13], i.e. \(2^{128} = (4 \times \pi \times 6378137^2) = 6.6 \times 10^{23}\) indicates the number of possible devices per square meter using such a large address space. This allows providing each device with an IPv6 address, to use it for identification in the future IoT and to avoid any overlay mechanism with custom addressing schemes and overlay routing mechanisms. With the extended address space additional improvements regarding auto-configuration of nodes and the scalability of multicast routing are defined. Furthermore, a new address type anycast is introduced, which offers to send a packet to any one of a group of nodes.

2. **Header format simplification**: The header simplifications introduced in IPv6 make the protocol more attractive for deployments on embedded or constrained devices.

3. **Flow labeling capability**: The new flow labeling supports a sender to label the packets of particular flows and to request certain QoS properties. For example, real-
time in the sense of non-interrupted video streaming or voice over-IP are use cases for the flow label. However, in this context, the term real-time should not be confused with real-time constraints known from certain control network applications where timing guarantees need to be met by control devices. In the area of building automation, the flow label capability might be useful.

4. **Authentication and privacy**: IPv6 comes with an extension to provide authentication, data integrity and data confidentiality, all features that are definitely required in the future IoT.

### 2.5.4.2 IPv6 unicast address

The IPv6 128 bit address is structured in a network prefix part and host identifier part. Within the IoT6 stack each device is equipped with an IPv6 unicast address. Further the integration gateway provides an IPv6 address for every non-IPv6 device behind the gateway. However, the integration approach goes even further by allowing a global unicast IPv6 address for every oBIX object, which might be associated with a simple data point type. So the paradigm proposed is not as simple as "IPv6 on every device", but it is even more fine grained and should be read "IPv6 on every data point".

### 2.5.4.3 IPv6 multicast address

IPv6 multicasting is a very important feature exploited within the IoT6 stack. It can be used for group communication purposes.

<table>
<thead>
<tr>
<th>Bits</th>
<th>8</th>
<th>4</th>
<th>4</th>
<th>112</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>prefix</td>
<td>flags</td>
<td>scope</td>
<td>group ID</td>
</tr>
</tbody>
</table>

Table 2 IPv6 multicast address format

Multicast addresses have the prefix FF::/8 followed by two 4 bits groups for defining flags and a scope as illustrated in Table 2. A 112 bits group ID can be used to provide address space for multicast services. The scope allows defining how an IPv6 packet is handled by hosts and routers. The scope can be set to interface-local, link-local, admin-local, site-local, organizational-local or global scope. This allows to specify if a packet is sent to all hosts that joined a certain multicast address on the same link or within the same organization or globally. Using a global multicast address warrants a packet to be eligible for routing over the public Internet, but this does not necessarily mean that the routers must support this feature. If actuators and sensors within the same link shall be connected based on group communication, link-local or site-local scopes can be used which use the prefixes FF02::/16 and FF05::/16. For LLNs it is not optimal to transmit the full 128 bits IPv6 multicast address. For 6LoWPAN the 6LOWPAN IPHC provides stateless multicast address compression allowing compressing the address to either 48 bits or 32 bits carrying inline the flag and scope field or 8 bits without the flag and scope field, fixed to the link-local scope.

<table>
<thead>
<tr>
<th>Bits</th>
<th>8</th>
<th>4</th>
<th>4</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>prefix</td>
<td>flags</td>
<td>scope</td>
<td>group ID</td>
</tr>
</tbody>
</table>

Table 3 IPv6 multicast address format for 6LoWPAN
In order to avoid conflicts with well-known addresses\(^3\) the transient flag must be set. In that case it is possible to use the 48 or 32 bits representation with an address space of either 32 (cf. Table 3) or 16 bits for group communication relationships within local networks.

2.5.5 Data Link Layer
For native wireless IoT devices, basically any link is possible, but IEEE 802.15.4 and the 6LoWPAN adaption layer to IPv6 are desirable.

2.6 Service discovery
For service discovery within the IoT6 architecture, the mechanisms developed within WP3 are used. They are based on local resource registries and global resource repositories relying mainly on the hierarchical DNS system and the enhancements for multicast based discovery (mDNS) and service repositories (DNS-SD).

\(^3\) [http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xml](http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xml)
[3] Home and building automation technologies

The details of the technologies integrated within the IoT6 architecture have been presented in the deliverable D4.1. This section will focus on how the various communication principles and protocols can be mapped to the IoT6 stack. Further, it will be investigated how an automated mapping process can take place that allows an automated configured access to the various technologies through the IoT6 stack without the need for human supervision.

3.1 KNX mapping to IoT6 stack

3.1.1 Addressing

KNX uses two types of addresses and distinguishes between individual and group addresses. Within KNX, individual addresses are used for unicast based client/server communication. The use case scenarios for this interaction style are mainly the configuration and engineering of devices. This can be, for example, the deployment of the so-called application program (firmware) and parameters defining the behavior of the sensor and actuator. For process data exchange, the group addresses are used, which exchange process data between sensors and actuators. For example, a temperature sensor transmits the latest sensor values using a group address and not individual addresses. Further, a device has a unique individual address but the available communication objects representing the input and output signals of the device might be assigned to one or multiple group addresses. Figure 9 and Figure 10 provide again an overview of the addressing scheme of the KNX system.

<table>
<thead>
<tr>
<th>Individual Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet 0</td>
</tr>
<tr>
<td>7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>Area Address</td>
</tr>
<tr>
<td>Sub Network Address</td>
</tr>
</tbody>
</table>

Figure 9 Individual address

<table>
<thead>
<tr>
<th>Group Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet 0</td>
</tr>
<tr>
<td>7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

Figure 10 Group address

Both addresses use a 16-bit long identifier. However, for the integration into the IoT6 stack only the process data exchange via group addresses is of relevance. Any considerations concerning exchange of engineering data via individual addresses are out of scope.

The KNX group address is further hierarchically structured. Either a two-layer or three-layer addressing scheme can be used. For the two-layer scheme the group address contains a 4 bit main group and an 11 bit subgroup. For the three-layer scheme, a 4 bit main group is followed by a 3 bit middle group and finally an 8 bit sub group. Different layering schemes can be used to structure the communication relationships identified through a group address. For example, the building topology or the trade type can be used for the main group and middle group and the sub group can be used to identify a concrete communication relationship for this trade and building. However, the semantics behind the hierarchical group address scheme is left to the system engineer.
Address mapping

To map the addressing scheme of KNX to the IoT6 stack, the KNX group addresses are mapped to data points of oBIX objects that represent a device or functional block. Multiple KNX group addresses are mapped to a single IPv6 global unicast address, which provides access to the device object. In this case, the mapping can be seen on the level of the device specific KNX individual address, although the individual address will never be involved within the actual communication. The gateway is also capable of providing an IPv6 address on any oBIX object level, which allows having a mapping on the data point level. In this case, a one to one mapping of a KNX group address to an IPv6 global unicast address is possible. As the group communication mechanism of the gateway also allows to assign transient multiple IPv6 link-local or site-local multicast addresses on the data point a 1-to-n mapping of the KNX group addresses to IPv6 multicast addresses can be seen.

However, the semantics of this mapping needs to be reflected carefully, because the semantics of a mapped KNX group address to an IPv6 multicast address might not be a straightforward mapping retaining the communication logic. KNX group addresses are used for local bus communication and are assigned to the group communication objects of multiple devices that act upon these KNX group addresses following the paradigm of a shared network variable. So first is a mapping to a RESTful resource representation through an oBIX object. If this object aims to represent a certain device, a clear mapping is in general not possible since a KNX group address might always address multiple devices. The clear mapping is only possible if a group communication object of a certain KNX device has a KNX group address attached that is solely attached to this group communication endpoint.

The available IPv6 address types that can be assigned to the oBIX object have nothing to do with the KNX addressing scheme and only rely on the RESTful object representation. In this case, the IPv6 unicast address provides a meaningful way for a device centric or data-point centric client/server communication. The IPv6 multi-cast addresses on the other side provide a group-communication similar to the KNX group communication mechanism. So the semantics is about IoT6 enabled devices that communicate with each other.

Summarizing these approaches, a KNX group address may be mapped to a variety of different IPv6 address types. But the main point here is to provide a device centric and RESTful mapping between an oBIX object and KNX group addresses, which are used for process centric communication. Once this oBIX object can be populated with KNX process data, all IPv6 related addressing does not concern the KNX group addresses any more.

3.1.2 Transport layer

The KNX transport layer defines 5 communication modes, consisting of one connection-oriented and four connectionless-oriented modes:

1. Point-to-point (unicast) for reliable connection oriented communication
2. Point-to-point (unicast) for connection-less oriented communication
3. Point-to-multipoint (multicast) for connection-less oriented communication
4. Point-to-domain (broadcast) for connection-less oriented communication
5. Point-to-all-Points (system broadcast) for connection-less oriented communication

The different communication modes are used for different interaction scenarios: the connection-less point-to-point communication is used for device configuration to assign a KNX individual address; the connection-oriented point-to-point communication is used for storing the application program firmware on the device; the point-to-multipoint
communication is used for the process data exchange between KNX devices at runtime using the KNX group address as described in the previous section.

### 3.1.3 Application Layer

The KNX application layer uses the services provided by the KNX transport layer. KNX does not specify a presentation or session layer. The following table provides an overview of how the KNX application layer services can be mapped to the IoT6 stack. The mapping mainly addresses the communication services used at runtime within the KNX system for exchanging process data. The communication services to directly interact with the memory are not mapped since these are only relevant for the engineering process and application program related firmware download [14].

<table>
<thead>
<tr>
<th>KNX Kommunikations-Services</th>
<th>IoT-Stack Kommunikations-Services</th>
<th>IPv6</th>
<th>HTTP</th>
<th>CoAP</th>
<th>eBNI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_ADC_Read</td>
<td>Unicast</td>
<td></td>
<td>GET</td>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>A_Memory_Read</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>A_Memory_Write</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>A_MemoryBit_Write</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>A_InterfaceWrite</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>A_InterfaceBitWrite</td>
<td>-</td>
<td></td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>A_UserInterfaceRead</td>
<td>Unicast</td>
<td></td>
<td>GET</td>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>A_UserInterfaceWrite</td>
<td>Unicast</td>
<td></td>
<td>GET</td>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>A_Key_Write</td>
<td>Unicast</td>
<td></td>
<td>PUT</td>
<td>PUT</td>
<td></td>
</tr>
<tr>
<td>A_DeviceDescriptor_Read</td>
<td>Unicast</td>
<td></td>
<td>GET</td>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>APropertyValue_Read</td>
<td>Unicast</td>
<td></td>
<td>GET</td>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>APropertyValue_Write</td>
<td>Unicast</td>
<td></td>
<td>PUT</td>
<td>PUT</td>
<td></td>
</tr>
<tr>
<td>APropertyDescription_Read</td>
<td>Unicast</td>
<td></td>
<td>GET</td>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>ALink_Read</td>
<td>Unicast</td>
<td></td>
<td>GET</td>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>ALink_Write</td>
<td>Unicast</td>
<td></td>
<td>PUT</td>
<td>PUT</td>
<td></td>
</tr>
<tr>
<td>A_FunctionPropertyState_Read</td>
<td>Unicast</td>
<td></td>
<td>GET</td>
<td>GET</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4 Mapping KNX application services to the IoT6 stack**

### 3.1.4 Mapping KNX group communication to IoT6 stack RESTful interaction

The interworking-model of KNX relies on the concept of group communication based on group communication objects (GO). These group communication objects are used by the end devices to exchange process data within the KNX network. In this way, communicating devices are linked together without the need of knowing a concrete communication endpoint illustrates the KNX group communication concept.
If the so-called user application (firmware at device) recognizes a change on an input state the group object is updated. In a further step, the network stack checks all assigned group addresses to this group object and transmits a group value write application service request to the associated addresses using a network broadcast. Other devices receive these updates and the network stack checks if there are associated group objects. If these associated group addresses are found, the state of the group object is updated and the user application performs the change on the output state. An important advantage of this communication mechanism is that multiple devices can interact with each other without the need of a centralized controller. Further, all the process related device interactions are independent from each other. So if one device fails the communication of other related devices is not affected. However, this type of communication makes a mapping to a RESTful client/server based interaction problematic. If a device-centric view on resources should be provided it might not be possible without having side-effects, since there might be more than one device associated with the group address.

For having a standardized set of device features, KNX uses the concept of function blocks. A function block defines a set of input and output data points that a device needs to implement in order to fulfill the contract defined by the function block. Typically, such input and output data points are related to the physical I/O signals of the device, but also so-called soft data points can be provided. For example, a set point value of a temperature controller is realized as such a data point that provides access to a variable in memory. The function block of a device provides a good starting point for a resource-centric representation of a KNX device. In this case, the oBIX contract mechanism can be used to define similar contracts like KNX function blocks. The oBIX object provides a stateful representation of the associated KNX group objects. Therefore, it is required to keep the stateful representation synchronized with the bus events that occur.
For mapping the KNX bus system, the functional blocks provide a device centric view. For the associated data points, a mapping of the KNX data point types need to be defined.

A KNX data type includes a format and encoding information and a dimension with a range and unit. Figure 14 provides an example of a KNX data type that can be used to represent a simple boolean data point. For the main data point type, the information is encoded using a 1 bit representation with the given format. Further the semantics of a “1” or “0” state are fully specified through the sub data point types. For example, a logical “1” or “0” can either have the meaning associated of “on” or “off”, or “enabled” or “disabled”, “open” or “closed”. The concrete semantics is specified through the KNX standard.

The KNX main types and formats can be directly mapped to the oBIX value data point types (cf. Table 5 Mapping KNX data point types to oBIX value object types [14]. Range and unit information can be provided using a standard unit attribute provided by oBIX and the use of “enum” objects to represent the range and also to provide the semantic information about the encoding.
Table 5 Mapping KNX data point types to oBIX value object types [14]

<table>
<thead>
<tr>
<th>KNX data point type</th>
<th>oBIX value object type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean Value (DPT_B)</td>
<td>bool</td>
</tr>
<tr>
<td>Unsigned Value (DPT_U)</td>
<td>int</td>
</tr>
<tr>
<td>Float Value (DPT_F)</td>
<td>real</td>
</tr>
<tr>
<td>Character (DPT_A)</td>
<td>string</td>
</tr>
<tr>
<td>String (DTP_A)</td>
<td>string</td>
</tr>
<tr>
<td>Time</td>
<td>abstime</td>
</tr>
<tr>
<td>Date</td>
<td>abstime</td>
</tr>
</tbody>
</table>

A more sophisticated mapping is shown in Listing 4 which illustrates how the KNX main and sub types can be represented using oBIX contracts and further semantics can be preserved by representing the encoding.

```
<obj name="P-0341-0_DI-3_M-0001_A-9803-03-3F77_O-3_R-4" href="switch_channel_a/"
    is="knx:DPST-1-1 knx:DPT-1 knx:Datapoint" display="On / Off"
    displayName="Switch, Channel A">
    <bool name="value" href="switch_channel_a/value" val="false"
        displayName="On / Off" null="true" writable="true" />
    <enum name="encoding" href="switch_channel_a/encoding" val="off"
        null="true" writable="true" range="/encodings/onoff" />
</obj>

<list href="onoff/" of="obix:bool" is="obix:Range">
    <bool name="on" href="onoff/on" val="true" display="On" />
    <bool name="off" href="onoff/off" val="false" display="Off" />
</list>
```

Listing 4 KNX data point mapping with encoding information

3.1.6 Automated mapping of KNX devices

**Automatic run-time discovery**

KNX does not provide any discovery services, that allows an automatic discovery of KNX devices at the bus during runtime. Without a-priori knowledge of the available devices, their installed application firmware and group address relationships, it is usually not possible to derive the semantics of messages that are exchanged on the bus. Only the length of the binary exchanged application payload allows anticipating the type of communication. Due to the ambiguity of the semantics only the knowledge of a human observer could provide the required meta-information about the configuration.

**Discovery based on KNX project configuration**

For KNX, the engineering tool ETS provides the means to configure devices, download the application firmware on devices and the configuration of communication endpoints. Therefore, all required information to provide an automated mapping to the IoT6 stack is available within the ETS project file. The ETS provides an export functionality that allows exporting an XML-based description of the KNX bus system that can be used for an automated configuration of the IoT6 gateway.
### 3.2 BACnet

BACnet is a prominent representative of a building automation technology for commercial buildings. By having a collapsed protocol architecture of 4 layers as shown in Figure 15, it is tailored and optimized to the specific needs of a building automation control network.

#### 3.2.1 Addressing

For addressing devices, BACnet uses a MAC address on the data link layer that depends on the used media. For the routing within a BACnet internetwork, a network number is further used to identify a device.

As can be seen in Figure 17, BACnet/IP is defined as medium within the BACnet protocol stack, yet it should not be mistaken that IP is used as network protocol for end-to-end communication between hosts. IP acts as a virtual link layer simply to tunnel BACnet packets.
through existing IP based networks. For end-to-end communication, still the BACnet network number and media dependent address information are required. Further, for a mapping of the address information, application layer specific interaction mechanism of BACnet needs to be considered. Since BACnet follows a client/server based interaction model in which controllers provide access to a moderate number of I/O signals and software based control objects, a straight mapping to IPv6 global unicast addresses is possible. The IoT6 gateway can provide an IPv6 global unicast address for a BACnet device identified through the BACnet network number and the media dependent address information. But to map the addresses the application layer of BACnet needs to be considered. BACnet uses an object-oriented approach to represent control devices and their provided data points. For the application layer services, a so-called BACnet object identifier is used to uniquely identify objects. As illustrated in Figure 18 it consists of a 10 bit object type and a 22 bit instance number. Further, it is mandatory that each device provides a device object with a unique instance number within the whole BACnet internetwork. In this way the maximum number of possible BACnet devices interconnected is limited by the 22 bit address space.

![Figure 18 BACnet object identifier](image)

In principle, the BACnet device instance number is of interest for a mapping to an IPv6 address [15]. The mapping of IPv6 multicast addresses can happen on the level of the exposed data points of an oBIX based mapping of the BACnet object model, which will be shown in the next subsection.

### 3.2.2 Transport Layer

The responsibilities of the transport layer, which are providing reliable connection-oriented or unreliable packet oriented communication, are addressed by BACnet within the application layer services by defining a custom acknowledgement and message exchange patterns.

### 3.2.3 Application Layer services

The BACnet application layer uses an object-oriented information model to provide access to the control network. The application layer services provide all required functionalities for inter-process and end-to-end communication between BACnet enabled devices. Every information asset or interface is modeled in an object-oriented way. Each BACnet device holds one or more BACnet objects. Every object can contain a list of standardized properties, where a set of standardized property and object types is available. Further, a property might be optional, mandatory but read-only or mandatory and read as well as writable. A minimum set of required properties are the Object_Identifier, the Object_Name and the Object_Type which must be provided by each BACnet object. The BACnet standard defines an enumeration of the available property identifiers.

BACnet follows an object oriented approach in modeling its devices and data points. A BACnet device can be seen as a collection of objects. Every device must have the special “device”-object that provides additional information about itself, including its device identifier, its name, and a list of all objects available on the device.

The BACnet standard defines a number of object types. These objects can represent physical
points or describe processes or internal operations. The *Program* object type, for example, represents a process running within a BACnet device. Physical points can be modeled using generic object types for binary and analog values.

There are also separate object types for input, output and value objects, resulting in a total of 6 object types: *AnalogInput*, *AnalogOutput*, *AnalogValue*, *BinaryInput*, *BinaryOutput*, *BinaryValue*. Input objects receive their value from an external source (e.g. sensors) and therefore are not writable. Output objects are writable and represent control outputs. Value and output objects can be used as set points (e.g. the target temperature in a heating system). Every object has a collection of properties. Among these properties are name and type of the object, as well as a description. For analog objects, a property provides a way to specify the units of the value. There are a number of other properties, and each object type has its own set of properties that are useful for that type.

**Present value property**

An important property is the *Present_Value*. If this property is writable, then the properties *Priority_Array* and *Relinquish_Default* are also present on the object. These properties are part of the prioritization mechanism. The priority array is a read-only array property of 16 values which correspond to 16 available levels of priority. The elements are in order of decreasing priority, so the first element (priority 1) has the highest priority. An entry in the priority array can be either a value or *NULL*. The non-*NULL* value with highest priority gets mapped to the *Present_Value* property. If there is no non-*NULL* value, then the value of the *Relinquish_Default* property is used. To set entries in the priority array, a *WriteProperty* request including the priority to override is issued on the *Present_Value* property. To clear entries, the value of the *WriteProperty* request shall be *NULL*. If no priority is specified on the request, a default priority of 16 (the lowest priority) is assumed.

**Object identifier**

Objects within a device are identified through their object identifier. This identifier consists of two parts: The object type and an instance number. In order to uniquely identify a property of an object inside a BACnet network three values are required: the device identifier of the device, the object is residing on, the object identifier (object type and instance number) and the property identifier.

### 3.2.4 Mapping BACnet objects

A way to represent a BACnet network and its devices and objects inside an oBIX server is desired in order to be able to access and manipulate them through an oBIX representation. To achieve this, BACnet objects can be mapped to oBIX objects. This document will only discuss the mapping of the BACnet object types *AnalogInput*, *AnalogOutput*, *AnalogValue*, *BinaryInput*, *BinaryOutput* and *BinaryValue* to oBIX objects, as they are sufficient to model a wide variety of devices and applications, including sensor and actuator values, as well as set points. Other object types can be mapped in a similar fashion if needed.
Data types

First, the mapping of data types is examined. The group of analog objects in BACnet is of the data type real, which maps directly to the real oBIX object type. BACnet binary objects are of the data type BACnetBinaryPV, which can assume the values active and inactive. These values map to true and false of the oBIX object type bool, respectively. To create a functional mapping, the present value is the only property required to be mapped, other properties like the description are not vital. Therefore, we can already define basic oBIX contracts corresponding to the BACnet objects. These contracts are shown in Listing 5. BACnet input objects are by default not writable, output objects are writable by default. This is reflected in the contract definition.

```
<obj href="iot:AnalogInput">  
  <real name="value" val="0" writable="false" />
</obj>

<obj href="iot:AnalogOutput">  
  <real name="value" val="0" writable="true" />
</obj>

<obj href="iot:AnalogValue">  
  <real name="value" val="0" />
</obj>

<obj href="iot:BinaryInput">  
  <bool name="value" val="false" writable="false" />
</obj>

<obj href="iot:BinaryOutput">  
  <bool name="value" val="false" writable="true" />
</obj>

<obj href="iot:BinaryValue">  
  <bool name="value" val="false" />
</obj>
```

Listing 5 Basic oBIX contracts corresponding to BACnet object types

Mapping of read and write requests

Assuming the address of a BACnet object known, the appropriate oBIX contract can be chosen for an oBIX object mapped too, since the object type is part of the address. To get the current value of the BACnet object when reading a mapped oBIX object, the oBIX read request has to be mapped to the BACnet ReadProperty service. The ReadProperty service takes three arguments: the object identifier of the object to read from, the property identifier of the property to be read, and optionally a property array index. The object identifier is already known. The property identifier to choose is the identifier of the Present_Value property. The property array index is only needed if the property being read is an array. As the present value is a single value this argument is omitted. The response of the service contains the read property value. Its data type is either REAL or BACnetBinaryPV, depending on the object type. The value can be interpreted as previously discussed. For write requests to the oBIX object a similar mapping has to be defined. The write request maps to the WriteProperty service. This service takes five arguments: the object identifier, property identifier, property array index, property value and priority. For the object identifier,
property identifier and property array index the same considerations are made as with the ReadProperty service above. The property value argument contains the value we want to write to the object. For the priority, an integer in the range 1-16 can be chosen. If the priority is omitted, a priority of 16 (lowest priority) is implied. The highest priorities should be reserved for life safety emergency overrides, so a mid-range priority of around 10 is suggested. The priority mechanism allows to override a value with a lower priority and later revoke the new, high priority value and to return to the original value. To revoke a value with a certain priority, its entry in the priority array has to be reset to NULL. oBIX does not have a null object, instead it uses a facet, or attribute, to indicate a null value. Therefore, writing to an oBIX object with a null value can be used to reset its value. In this case, the BACnet NULL data type is the property value to be used in the WriteProperty service request.

Check for writable BACnet objects

The contract definitions already set the default of the writable facet for each object type. However, under certain circumstances the present values of Input and Value objects can become writable. In order to correctly reflect this in the oBIX object, these conditions have to be checked. Output objects are always writable. Input objects are writable, if the property Out_Of_Service is set to TRUE or if the Present_Value property is commandable, otherwise they are read only. The boolean property Out_Of_Service indicates whether physical input to the object is currently in service. While it is not in service, the physical input is decoupled from the Present_Value property and manually changing the Present_Value is allowed. If the Present_Value is commandable, then the properties Priority_Array and Relinquish_Default are both present on the object, too. To check if these properties are available, a ReadProperty service request can be attempted. If they are not available, the request will result in an UNKNOWN_PROPERTY error, indicating that the Present_Value property is not commandable.

These conditions have to be repeatedly checked every time the oBIX object is read, as they can change over time. Using the techniques described so far, a BACnet object can be mapped to a functional oBIX object that can be read and written. An example of an object using this mapping is shown in Listing 6.

```xml
<obj href="/BACnet/10003/AnalogOutput1" is="iot:AnalogOutput">
  <real name="value" href="value" val="100" writable="true" />
</obj>
```

Listing 6 Example oBIX object representing a BACnet Analog Output object

Additional properties

While the object may be functional, it is still not satisfactory. There is no information about the functionality or role of the object, only how its value is mapped. Without knowledge about what these values represent, it is quite useless. BACnet provides additional properties to describe objects which can be used to create a more easily understood oBIX representation.

Name

Every object has an Object_Name property. A name is unique within the BACnet device that contains the object. In oBIX, the direct children of an object must have unique names too. Names in BACnet are restricted to printable characters only, and a minimum length of one character. oBIX imposes stricter constraints on names and only allows ASCII letters, digits, underbars, and dollar signs. Additionally, a digit must not be used as first character. Invalid characters have to be stripped from the Object_Name before it can be used as name for the
The problem of duplicate names from different BACnet objects is largely avoided by structuring the oBIX representation as discussed in the next chapter.

**Description**

BACnet also provides a Description property. This property can be useful to understand the purpose of the object and can be included in the constructed oBIX object. To this end, a new child object of type `str` is added to the object. Its name is `description` and its value is the value obtained from the Description property by a ReadProperty service request. No conversion of the obtained string is required, as it is only restricted to printable characters.

**Units**

For analog values, a unit is required to be able to interpret it correctly. Both BACnet and oBIX provide means to specify the units for a value. In BACnet, the Units property of the data type `BACnetEngineeringUnits` represents the units of the Present_Value property. `BACnetEngineeringUnits` is an enumeration of many units from different domains, such as acceleration, area, currency, electrical, energy, enthalpy, entropy, force, frequency, humidity, length, light, mass, mass flow, power, pressure, temperature, time, torque, velocity, volume, volumetric flow and others. Units with an enumerated value in the range 0-255 are reserved for definition by ASHRAE, values in the range 256-65535 may be used freely. oBIX features a flexible way to define units mathematically. Dimensions are specified using the `obix:Dimension` contract using the seven fundamental SI units and their exponent (Listing 7).

```xml
<obj href="obix:Dimension">
  <int name="kg" val="0" />
  <int name="m" val="0" />
  <int name="sec" val="0" />
  <int name="K" val="0" />
  <int name="A" val="0" />
  <int name="mol" val="0" />
  <int name="cd" val="0" />
</obj>
```

Listing 7 oBIX dimension contract

An actual unit is represented with the `obix:Unit` contract (Listing 8). It contains a dimension that can be scaled and offset (unit = dimension * scale + offset) and the unit symbol. Listing 9 shows how a kilowatt can be expressed using this system.

```xml
<obj href="obix:Unit">
  <str name="symbol" val="kW" />
  <obj name="dimension" is="obix:Dimension">
    <int name="m" val="2" />
    <int name="kg" val="1" />
    <int name="sec" val="-3" />
  </obj>
  <real name="scale" val="1000" />
</obj>
```

Listing 8 oBIX unit contract

```xml
<obj href="obix:units/kilowatt" display="kilowatt">
  <str name="symbol" val="kW" />
  <obj name="dimension">
    <int name="m" val="2" />
    <int name="kg" val="1" />
    <int name="sec" val="-3" />
  </obj>
  <real name="scale" val="1000" />
</obj>
```

Listing 9 Kilowatt as oBIX unit

Some units do not fit into this model, like logarithmic units or units dealing with angles. Such units should use a dimension where every exponent is set to zero. oBIX provides a database of predefined units. If possible, `BACnetEngineeringUnits` should be mapped to the...
corresponding unit in this database. New units can be defined. For example, BACnet revolutions-per-minute (with an enumerated value of 104) maps to obix:units/revolutions_per_minute. BACnet has a special enumerated value representing the absence of a unit, no-units. This value can be mapped by simply omitting the unit attribute of the oBIX object.

**Complete oBIX object representation**

By mapping these additional properties, a much more meaningful object is obtained. If we compare Listing 6 with Listing 10. The second example gives a much better idea of what it does, even though both may actually represent the same BACnet object.

```xml
<obj name="lfan" href="/BACnet/10003/AnalogOutput1" is="iot:AnalogOutput">
  <real name="value" href="value" val="4200" units="obix:units/revolutions_per_minute" writable="true" />
  <str name="description" href="description" val="left fan speed setpoint" />
</obj>
```

*Listing 10 Complete oBIX object representing an BACnet Analog Output object*

### 3.2.5 Automated mapping of BACnet devices

**BACnet Remote Device Management Services**

Among the services that BACnet provides there are a group of services collectively known as “Remote Device Management Services”. These services provide a number of functions, including operator control and auto-configuration functions. Among them there are two services that can be used to discover devices, the *Who-Is* and *Who-Has* services. They eliminate the effort needed to program the network addresses of other devices into each device. The service messages are broadcast in the BACnet network to every device, and the receiving devices may respond with an acknowledgment message containing their address.

**Who-Is and I-Am**

The *Who-Is* service is used to determine the device identifier, the network address, or both of other BACnet devices that are on the same network as the device issuing the service request. It is an unconfirmed service, meaning that it does not require a response. There are multiple ways of using the service. It can be used to determine the device object identifier and network addresses of all devices on the network. If a device object identifier is already known, then the service can be used to determine the address of the corresponding device. The *I-Am* service informs other devices about its sender. It broadcasts an unconfirmed request containing the device identifier among other information. The service may be used at any time. Usually a *I-Am* request is sent after a device has initialized to inform other devices about its availability. When a device receives a *Who-Is* request whose parameters include the device, then it answers with a *I-Am* service request. *Who-Has* and *I-Have* Another way to locate devices on the network is provided by the unconfirmed services *Who-Has* and *I-Have*. The *Who-Has* service is similar to the *Who-Is* service. It can be used to ask for the device identifier of devices that contain an object that has a given object name or object identifier. In response, devices that have the requested resource send an *I-Have* service request, containing the device identifier, as well as both the object name and object type of the requested object.

**Device Instance Ranges**

Optionally, a range of device instances can be specified as an argument for the *Who-Is* and *Who-Has* service requests. If the range is omitted, then every device that receives the request will process it. Otherwise, only devices whose device object’s instance number are
within the specified range will answer. In big networks, an unbound Who-Is request to all devices at once would result in a lot of answers at the same time and sudden network load. As a consequence, packets containing the I-Am response are more likely to be dropped on their way to the requesting device, and since the I-Am service is unconfirmed, the packets will not be resent. This may lead to not all devices being discovered. The device instance ranges provide a solution to this problem. Instead of one Who-Is request to all devices, the request can be split up into several smaller requests, thereby reducing the network load per request.

**Device objects and object lists**

Every BACnet device is required to have a device object. Its instance number is the same as the device identifier. The device object offers a property called *Object_List*. It is an Array type property that contains all objects available on the device. This makes it possible to query all the objects contained in a device which has previously been discovered by the Who-Is service.

### 3.3 ZigBee

As outlined in deliverable D4.1 ZigBee resides on top of IEEE 802.15.4, which provides the functionality of the ISO/OSI physical and data link layer. ZigBee uses the non-beacon enabled CSMA/CA model of IEEE 802.15.4 and thus does not use features such as deterministic network access or the possibility to reserve bandwidth for applications. ZigBee is aligned to the ISO/OSI model and defines a network and application layer on top of IEEE 802.15.4 as outlined in Figure 20. The application layer is further differentiated into an application support sublayer (APS), a ZigBee device object (ZDO), a ZigBee cluster library (ZCL), security layers and an application framework (AF).

![Figure 20 ZigBee stack protocol stack](image)

**3.3.1 Addressing**

Addressing within ZigBee is based on the IEEE 802.15.4 addressing, which provides a 16-bit PAN ID and either a 16-bit short address or a 64-bit long address. On the network layer a 16-bit network address (*NwkAddr*) is used by the application sub layer to address a node within the wireless network. Within a node an endpoint identifier between 1 and 240 can be used to address a certain application object. Within the application object a 16 bit cluster identifier is used to select the object of interest. The attribute of the object can be also addressed through a 16 bit attribute identifier.
In relation to the IoT6 stack the NwkAddr can be directly mapped to an IPv6 global unicast address. A direct mapping of the group addressing mechanism of ZigBee to IPv6 multicast addresses would be not meaningful, since the IPv6 multicasting addressing is used in the IoT6 stack to build a concept similar to the group communication mechanism of ZigBee. First the mapping to a resource-oriented representation has to happen. Therefore, also ZigBee group addresses could be used, although a device centric mapping if possible is preferable. Finally, the endpoint, cluster and attribute identifier are subsumed by the URI identifier within the IoT6 stack.

3.3.2 Application layer

At the application layer the available commands and object types depend on the application profile. For example the home automation profile or the smart energy profile defines the object structures, the commands and the data structures that are exchanged.

![Figure 21 Clusters support by On/Off switch of home automation profile](image)

So the device definition and the clusters can be roughly compared to the concept of KNX functional blocks. Taking the On/Off switch as example there might be devices that offer only a simple switch and there might be devices where multiple switches are offered. In this case, ZigBee would require using multiple application objects that are accessible through dedicated endpoint identifiers.

This application layer concept is mapped to the IoT6 stack by relying on the standard set of generic oBIX contracts for the IoT6 stack. The different application objects are realized as a set of different resources adhering to the oBIX contracts. Similarly, the group communication
concept is mapped to the IoT6 group communication principle that can be used for any object that is based on the oBIX basic value object.

3.4 **EnOcean**

EnOcean is a wireless automation technology that is gaining importance, due to the fact that it relies mainly on energy harvesting mechanisms to power its sensors. The energy can be harvested from mechanical processes, like pressing a button, or by retrieving the energy from solar panels. The EnOcean alliance is behind the technology and it aims at establishing an international standard. Therefore, the so called EnOcean Equipment Profiles specify the application layer services that rely on the three lower level layers as specified in the ISO/IEC 14543-10 standard (wireless short-packet protocol stack).

![Figure 22 ISO/IEC 14543-3-10 protocol architecture (WSP stack) [8]](image)

### 3.4.1 Addressing

EnOcean uses 4 byte identifier for addressing devices. This address identifier can be directly mapped to an IoT6 representation of a device. Therefore, an IPv6 global unicast address mapping is suitable for a direct mapping to the 4 byte identifier of the EnOcean network layer.

### 3.4.2 Application layer services

Due to the constraints imposed by the required energy efficiency the protocol is kept quite simple. Most important for the mapping to IoT6 stack are the RPS telegrams that are used to transmit switching signals in the network. The RORG field represents a 1 byte identifier used to distinguish between the different telegram fields. The standard provides further the bit encoding of the different signals that are used as payload for the switching of devices.
### Table 7 EnOcean Equipment Profiles telegram types

The telegrams for the learn mode of EnOcean would allow to engineer the control logic found in EnOcean and might only be of interest if a new system is deployed.

For the mapping the existing generic IoT6 oBIX contracts can be used.

<table>
<thead>
<tr>
<th>Telegram</th>
<th>RORG</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPS</td>
<td>F6</td>
<td>Repeated Switch Communication</td>
</tr>
<tr>
<td>1BS</td>
<td>D5</td>
<td>1 Byte Communication</td>
</tr>
<tr>
<td>4BS</td>
<td>A5</td>
<td>4 Byte Communication</td>
</tr>
<tr>
<td>VLD</td>
<td>D2</td>
<td>Variable Length Data</td>
</tr>
<tr>
<td>MSC</td>
<td>D1</td>
<td>Manufacturer Specific Communication</td>
</tr>
<tr>
<td>ADT</td>
<td>A6</td>
<td>Addressing Destination Telegram</td>
</tr>
<tr>
<td>SM_LRN_REQ</td>
<td>C6</td>
<td>Smart Ack Learn Request</td>
</tr>
<tr>
<td>SM_LRN_ANS</td>
<td>C7</td>
<td>Smart Ack Learn Answer</td>
</tr>
<tr>
<td>SM_REC</td>
<td>A7</td>
<td>Smart Ack Reclaim</td>
</tr>
<tr>
<td>SYS_EX</td>
<td>C5</td>
<td>Remote Management</td>
</tr>
<tr>
<td>SEC</td>
<td>30</td>
<td>Secure telegram</td>
</tr>
<tr>
<td>SEC_ENCAPS</td>
<td>31</td>
<td>Secure telegram with R-ORG encapsulation</td>
</tr>
</tbody>
</table>
[4] Smart grid technologies

Within this section the mapping of a selected smart meter communication protocol found in smart grid systems is discussed.

4.1 Wired M-Bus

Also, the M-Bus protocol stack is based on the OSI-model of the International Organization of Standardization (ISO) [17]. Since M-Bus does not provide all features of an ISO-/OSI computer network, only a subset of this protocol stack needs to be used. Thus, the M-Bus protocol stack only uses three layers and builds upon the standard defined by IEC EN 61334-4-1.

<table>
<thead>
<tr>
<th>Number</th>
<th>Layer</th>
<th>Function</th>
<th>Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Application</td>
<td>5</td>
<td>EN 13757-3</td>
</tr>
<tr>
<td>2</td>
<td>Data Link</td>
<td>8</td>
<td>EN 13757-2 or EN 13757-4</td>
</tr>
<tr>
<td>1</td>
<td>Physical</td>
<td>4</td>
<td>EN 13757-2 or EN 13757-4</td>
</tr>
</tbody>
</table>

*Table 8 M-Bus protocol stack*

An advantage of using this protocol stack is that higher protocol layers are independent from their low level counterparts, since the communication is only realized between layers of the same level.

This means that the implementation of the lower level layers can be changed and the functionality of the system is not compromised. Applying this configuration allows for example the usage of wired (EN 13757-2) as well as wireless (EN 13757-4) communication media without any changes to the actual application. Defining a protocol for these layers allows interchangeability of meters. Special gateways/bridges are in place to connect network parts with different protocol implementations.

4.2 Wireless M-Bus

Using the ISO/OSI model as a reference model has the advantage that implementations of layers can be changed and the overall system still is operational. This allows new technologies to be put into place to replace existing ones. Furthermore, it is possible to interconnect different kinds of implementations by providing a bridge for each network part so that communication can be established throughout the network.

Modern approaches often rely on wireless connections of devices since it is very cost effective. The same applies also for M-Bus systems and thus a specification for a Wireless-M-Bus system has been formed its result has been accepted as a European norm. The norm specifies how the physical and data link layer for wireless communications operate and is specifically targeting short range devices in unused frequency bands.

Wireless-M-Bus systems allow communication between measurement entities and non-stationary units (for example, master devices such as a laptop as a data collector). To achieve the communication, several operation types are specified (all operation types are identified by a name and a number):

1. Stationary operation method (S): This method is used for the unidirectional or bidirectional data transmission between measurement units and flexible master.
devices. Sub-methods of this type include optimized methods for long message headers (S1) and mobile devices (S1-m).

2. Frequent send operation method (T): Here, small telegrams are used to transfer data in very short time frames (seconds). This allows tracking measurement data in a very short time and is utilized by mobile devices that are not constantly in range of the meter itself. Furthermore, this method allows creating a measurement graph on almost real-time data. Sub-methods include operations for only sending data in periodic time frames or at random (T1) and a bidirectional method that uses a short initialization telegram to create a transmission channel (T2).

3. Time frequent receive operation method (R): A measurement unit listens for incoming messages (in a frequent interval). If it receives one it issues a transmission channel with the sender. This method allows the master device to query several meter units at the same time as all of them use a separate frequency channel.

Figure 23 above shows an example for a data exchange between a master and a slave unit. The communication mode used is a bidirectional method that is based on a short initialization frame (ADR - Access Demand Request) that is sent in periodical intervals (T2). This frame is sent out by the meter (slave) to be detectable for retrieval devices (master). After the slave device sends such an ADR it waits for a specific time frame to receive an acknowledgement packet (ACK-Time). If such an acknowledgement is received a bidirectional communication channel is opened, otherwise the slave issues a time-out until the next ADR is sent. In case an acknowledgement packet is received at a master device the slave is ready to receive requests from the user side (master). After receiving a request the slave answers with the corresponding response and continues with its normal operation. This example simplifies the process as there are several additional constraints that apply for the communication like considering response time or time-outs for not receiving requests after an acknowledgement [18] [19].

The current standard only covers the basic operation methods described above and is not sufficient for modern purposes any more. Thus, a new standard has been described and is currently under review and will replace the existing European norm. The new norm introduces several additional operation types that allow new implementation fields (meter
devices which operate mainly as receivers of commands for example). Furthermore, a more
detailed definition of how the physical layer operates is given. Additionally, the data link
layer, and especially how the frames including their control headers are built, is described in
more detail.

![Figure 24 Wireless M-Bus example telegram](image)
The Wireless M-Bus telegram consists of a header and body part. Within the header part a
length field, control field, manufacturer field and address field are provided. Within the body
the control information field identifies the application payload and contained information.
For parsing the telegram AES decryption needs to be used.

### 4.2.1 Addressing
For mapping of the wireless M-Bus address space the 6 byte address field can be mapped to
an IPv6 global unicast address.

### 4.2.2 Messaging
Wireless M-Bus provides a variety of different interaction modes. For deployed smart meters
it is quite likely that only the non-solicited transmission mode will be used. This means the
smart meter will transmit the current energy and power readings in a certain time interval
(e.g. 15 minutes) without having a client requested any data. Clients like the IoT6 gateway
can then monitor the wireless communication and record the transmitted information.
Beside the address information it is required to have the 128 bit AES key to decrypt the
telegram.

### 4.2.3 Application Layer
The wireless M-Bus standard allows relying for the application layer to different data
representation formats. For example the M-Bus related application layer data formats or the
DLMS/COSEM related formatting can be used.

COSEM follows an object oriented way to provide access to the registers of a smart meter
holding the desired information (cf. Figure 25).
An interface class in COSEM provides attributes and methods. The first attribute is the **logical_name** which uses the Object Identification System (OBIS) to identify commonly used data items in metering equipment. OBIS follows a hierarchical structure for the identifier. It defines ID-codes that link to data object entities of a certain domain (e.g. electricity). The identifier is a combination of six value groups (A to F), which describe together the exact meaning of a data item.

**Table 9 OBIS code structure**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
</table>

The following information is conveyed by the value groups:

1. A: defines the media  
2. B: defines the channel used  
3. C: defines the data related to the measured value  
4. D: defines the way the data has been processed  
5. E: defines where relevant the related tariff or allows for a further classification  
6. F: defines, where relevant, the storage of data or allows for further classification

Numerous data assets are defined for different meter domains. The typical domains are defined by the A value group limited to an identifier between 0 and 15. The currently standardized domains are:

1. 0: Abstract objects  
2. 1: Electricity related objects  
3. 4: Heat cost allocator related objects  
4. 5: Cooling related objects  
5. 6: Heat related objects  
6. 7: Gas related objects  
7. 8: Cold water related objects  
8. 9: Hot water related objects

For the electricity domain around 100 codes are defined in the C value group. So a full enumeration is out of scope for this document. The most relevant registers for the smart grid application scenario are the current power load, the accumulated energy consumption and quality parameters of the local power grid.
Another way to interface the smart meter is to use the application layer format defined by M-Bus. In this case one or many data container structures are directly following after the M-bus protocol header (cf. Figure 26).

<table>
<thead>
<tr>
<th>DIF</th>
<th>DIFE</th>
<th>VIF</th>
<th>VIFE</th>
<th>Daten</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>0 to 10 (per 1 Byte)</td>
<td>1 Byte</td>
<td>0 to 10 (per 1 Byte)</td>
<td>0 to N Byte</td>
</tr>
<tr>
<td>Data Information Block (DIB)</td>
<td>Value Information Block VIB (VIB)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data Block Header

The data information field (DIF) and the data information field extension (DIFE) specify the format of the following data, e.g. being an integer or floating point number and the bit length. The value information field (VIF) and the value information field extension (VIFE) specify the encoding and provide the semantics whether the following data bits are representing energy, volume or power.

4.3 Mapping Wireless and Wired M-Bus to the IoT6 stack

A mapping of wired or wireless M-Bus is straightforwardly created through a smart meter oBIX object that represents a smart meter and all its contained data registers. This structure can be used independent of the underlying communication be it wired or wireless M-Bus.

Within the project a mapping of the current power and energy consumption of an electricity smart meter based on wireless M-bus has been defined. Further, a wired smart meter for a floor heating system has been integrated.

The following oBIX contract (Figure 27) defines the used abstraction specified within the IoT6 project to represent smart meter devices. The smart meter object contains a flat list of data registers provided by the smart meter. The data registers are mapped to basic oBIX value objects that define the format through the value type (e.g. bool or real) and a unit attribute.

```xml
<obj href="iot:SmartMeter">
  <real name="power" href="smartmeter/power" val="0.0" unit="obix:units/watt"/>
  <real name="volume" href="smartmeter/volume" val="0.0" unit="obix:units/liter"/>
  <real name="energy" href="smartmeter/energy" val="0.0" unit="obix:units/kilowatthours"/>
</obj>
```

For the basic value object types it is again possible to assign IPv6 multicast addresses, allowing the smart meter data points to interact with other IoT6 enabled devices.
[5] Identification technologies and other information sources

This section discusses the mapping of technologies not related to home and building automation or the smart grid.

5.1 RFID

For mapping RFID to the IoT6 stack the RFID tag reader devices and the conveyed tag information are the entities of interest (cf. Figure 28).

5.1.1 RFID address format

5.1.2 Mapping RFID tag reader to IoT6 stack

```xml
<obj href="rfidReader/" is="iot:RfidReader">
    <str name="tag" href="rfidReader/tag" val="" writable="true" />
    <abstime name="time" href="rfidReader/time" val="2014-03-25T13:31:06.809+01:00" tz="Europe/Vienna" />
</obj>
```

Figure 28 oBIX contract for representing RFID reader

5.2 Weather data

Weather data and forecasts can be considered relevant for use cases found in home and building automation and smart grid use cases. For example the expected solar radiation of the next hours or the expected wind allow forecasting the expected energy production through renewable energy data sources. Further, if a bad weather front is expected, open windows or doors may cause a safety issue and might require the attention and attention of a human person.

5.2.1 Weather data source

For the integration of the weather data, a free weather data API is used. It is provided by the meteorological institute of Norway4. It provides access to weather data in a REST-style. An XML schema document is provided together with the structure of the weather data. The weather data service provides a weather data element that holds a time series of weather forecasts for a certain location.

Data structure and types

A weather forecast consists of a predicted temperature, wind direction, wind speed, humidity, pressure, cloudiness, fog, cloudiness and dew point temperature. The service uses XML schema to describe complex data types to represent these XML elements. Basic information is provided using the data types and formats provided by XML schema.

Message exchange

To retrieve the weather data, an HTTP request is performed to retrieve the XML formatted weather data. The location is provided as query parameters within the URI of the request.

4 www.yr.no
5.2.2 Mapping weather data to the IoT6 stack

Figure 29 oBIX Mapping of Weather Data

For mapping the weather data, a representation based on the oBIX meta-model needs to be defined. This can be done by providing a complex weather forecast oBIX object, which contains a location object, an upcoming weather forecast and a time series of further weather forecasts for the near future (cf Listing 11).

Listing 11 Weather data representation in oBIX

As it can be seen the simple elements of the weather forecast are mapped to the oBIX basic value objects where the unit attribute provides further information of the semantics.
[6] **IoT6 stack on constrained devices**

To realize the IoT6 stack on constrained devices, a 6LoWPAN based platform is used which comes with a microcontroller and IEEE 802.15.4 transceiver. To realize the firmware, the Contiki operating system is used which provides an optimized implementation for IPv6 and further comes with a CoAP reference implementation. Based on this implementation, an IoT6 application firmware has been developed that provides an oBIX based object representation of the sensors and actuators attached to device and further implements the IPv6 based group communication mechanism.

[7] **IoT6 gateway implementation**

7.1 **IoT6 gateway architecture**

Figure 27 illustrates a closer view on the required components of the IPv6 multi-protocol gateway. The gateway provides interfaces according to the IoT6 stack for several existing state-of-the-art technologies, which are non-IPv6 based. It has a common interface for direct browser-based interaction with these devices, for control and monitoring systems, for Cloud-based software-as-a-service interaction, for global discovery services and interfacing through mobile computing.

Multiple HTTP and CoAP handlers are offered and bound to virtual and physical network interfaces. An HTTP and a CoAP handler can be used to offer a centralized interface conforming to the traditional oBIX approach to interact with devices. In 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.

The message encoding and decoding is accomplished transparently between the HTTP and CoAP handlers and the oBIX server. As encoding XML, JSON, EXI or oBIX binary encoding can be used to encode the transferred oBIX objects.

The oBIX server takes care of read, write and invoke requests to the oBIX objects that are internally controlled by an object broker. The oBIX server is independent of the protocol that is used to interact with the objects. The object broker takes care about oBIX watches, histories and alarms. The oBIX watches can be used to monitor changes of objects and follows the observer pattern to realize this functionality. In case of a CoAP get request with an observe option, it also takes care of sending asynchronous responses whenever changes occur. For clients, the traditional oBIX polling, the CoAP observe approach or both ways can be used to observe a resource.

Furthermore, the oBIX broker publishes the represented devices through the traditional oBIX lobby which can be easily mapped to the CoAP /well-known listing of available resources according to the CoRE link format [20]. 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. It is even possible to offer centralized and per-device access in parallel, since requests to all endpoints will be handled by the same oBIX objects. Further it is even possible to assign an IPv6 address on a more fine-grained level by using assigning it to any kind of oBIX object. IPv6 multi-cast addresses can be assigned to basic oBIX value objects.
Figure 30 Gateway architecture

For realizing the IoT6 stack group communication mechanism a group communication service intercepts the requests that arrive at the CoAP handler and directly interacts with the object broker to update the internal states of the objects. Further, it maintains a group communication address table and monitors the oBIX objects which are linked to an IPv6 multicast address. On a state change, the according update is sent out on the network using a non-confirmable CoAP PUT request.

Beside this, the object broker uses a discovery service client to register the available devices and resources at the IoT6 global discovery service.

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 underlying technologies the connections to different physical and data link layers need to be provided. Furthermore, the mapping of BAS specific concepts to the generic objects adhering to the IoT-oBIX contracts happens here. These contracts allow to map various technologies into a common object oriented representation. Next to the BAS-specific adapters, also other data sources are integrated based on this mechanism, for example the weather data service or “IoT Logic components”.
7.2 Integration with local control and monitoring system

Beside a standalone deployment of the gateway components it is possible to directly include the protocol bundles in the local control and monitoring system (CMS) based on the UDG. Since the CMS is also based on OSGI, the bundles can be deployed within the same container.

Figure 31 Deployment of protocol bundles within the control and monitoring system (CMS)

The control and monitoring system, so called UDG, consist of an actions and events engine that allows to specify control logics as described in the deliverable D4.2. Therefore, the main design paradigm within the CMS is an actions- and events-oriented design. Furthermore, all the components are contained in so-called service-bundles that follow the design of the UDG
On the first sight, this might contradict somehow the resource centric design provided by oBIX. However, a so called “IoT application module” provides a mapping between the resource-oriented design of the multi-protocol abstraction layer and the action and event orientation of the CMS.

For the mapping a simple oBIX API is used to access the data points provided by an arbitrary object. This allows to create generic adapter that is able to integrate every technology that is accessible through the oBIX abstraction. By doing so the CMS only needs a single adapter based on oBIX and not a separate adapter for each technology.

Figure 32 IoT application module for the control and monitoring system

Figure 32 provides an overview of the API provided by the IoT application module. Simple actions are provided to update a basic data value object of oBIX. Furthermore, an object observer is used to monitor all the states of the objects that are provided by the object broker of the gateway. If an update is detected an event is generated that can be processed by the CMS.

```java
@Action
public void writeBool(@Param("propertyName")String propertyName, @Param("value")boolean value) {
    if(objectBroker != null){
        String obixUri = this.DGGetDevicePhysicalCode();
        log.info("writeBool called on " + obixUri + " value: " + value);
        Obj boolObj = objectBroker.pullObj(new Uri(obixUri));
    }
}
```
Listing 12 IoT6 application module - action example

Listing 12 illustrates how the access to a basic bool oBIX object can be provided as action. This method is provided on a per-device object basis, and the caller needs to specify the property name of the data point and the new Boolean value. Similarly, Listing 13 illustrates how an observed Boolean data point can be mapped to an event.

Listing 13 IoT6 application module - event example

The rules and scenarios of the CMS can act upon these actions and events, independent of the underlying technology. The available oBIX objects are configured in the database and a persistence API provides the required information at startup to configure all available devices mapped by oBIX objects.

7.3 Implementation

The implementation of the gateway is based on Java 7 and published as open source project IoTSyS\(^5\), which acts as a reference implementation of the developed IoT6 stack. This allows making the project results best reusable for the research community and industry. A partly complete reference implementation of the oBIX 1.1 standard has been performed.

The oBIX implementation is based on the oBIX toolkit. NanoHTTP\(^6\) is used as HTTP server. The open source implementation Californium\(^7\) is used as CoAP server. Also for the message decoding and encoding mainly other open source libraries are reused.

For the implementation, the OSGI service framework is used, which allows a component-oriented development and nice modularization of the different connectors to the integrated technologies. The communication drivers and connectors as well as technology-specific mappings of oBIX objects based on the generic abstraction are handled via so called protocol bundles which are described in more detail in the following subsections.

7.4 Gateway configuration and device discovery

The gateway can be configured dynamically on which protocol bundles should be used or not. In an OSGi container the bundles can be added or removed at runtime. For discovery of

---

\(^5\) www.iotsys.org

\(^6\) https://github.com/NanoHttpd/nanohttpd

\(^7\) https://github.com/mkovatsc/Californium
A static XML configuration can be provided, which contains the information about the bus-specific communication address of the various technologies to the IoT6 gateway. Beside this static information source, for some technologies an automatic discovery of devices can be realized at run-time.

The so-called devices.xml defines the information on how technology specific address information is mapped to the oBIX based implementation. If the gateway is started in a standalone mode without an OSGI container a device loader section in the XML file allows to configure which technologies should be used. Listing 12 gives an example of how the configuration can be done for the KNX bus system. Either a static mapping of the KNX group addresses or an automated mapping based on the KNX project file export can be used.

```
<devices>
  <!-- Configure which technologies should be loaded -->
  <deviceloaders>
    <device-loader>at.ac.tuwien.auto.iotsys.gateway.connectors.knx.KNXDeviceLoaderImpl</device-loader>
    ...
  </deviceloaders>
  <!-- static mapping between oBIX objects and KNX bus system -->
  <knx>
    <connector>
      <name>KNX IoT Suitcase</name>
      <enabled>false</enabled>
      <router>
        <ip>192.168.1.14</ip>
        <port>3671</port>
      </router>
      <localIP>auto</localIP>
      <device>
        <type>at.ac.tuwien.auto.iotsys.gateway.obix.objects.iot.actuators.impl.knx.TextDisplayActuatorImplKnx</type>
        <address>null, 0/0/3</address>
        <href>textDisplay</href>
        <historyEnabled>true</historyEnabled>
        <historyCount>200</historyCount>
      </device>
    </connector>
    <!-- automated configuration based on ETS project file -->
    <knx-ets>
      <connector>
        <name>KNX TP1 Suitcase</name>
        <enabled>false</enabled>
        <forceRefresh>false</forceRefresh>
        <router>
          <ip>192.168.1.100</ip>
          <port>3671</port>
        </router>
        <localIP>auto</localIP>
      </connector>
    </knx-ets>
  </knx>
</devices>
```

### Listing 14 Devices.xml example

#### 7.5 Protocol bundles

This section provides some insights into the internal structure of a selected number of realized protocol bundles.

#### 7.5.1 KNX protocol bundle

The KNX protocol bundle reuses the KNX Java library Calimero provided as open source
library by the Automation Systems Group\(^8\). The class diagram in Figure 33 shows the concept of how the KNX bus is monitored and the mapping to the IoT oBIX objects is performed. Since the bus communication of KNX relies on transient event-based communication that might not provide a state-based read access, it is required to provide an according oBIX object that retains the state of the current bus communication.

![Figure 33 KNX protocol bundle internals](image)

Due to the RESTful nature of the IoT6 stack it is required to provide a resource that keeps the current state of the KNX device. To reach this goal, group communication on the KNX bus is monitored and the oBIX objects mapped to the group communication address are updated. To provide an example for this process, Figure 33 illustrates the concept for a KNX temperature sensor. The KNX connector of the KNX protocol bundle provides a Java API to interface the bus. Part of the connector is a watch dog mechanism that follows the observer design pattern and allows an interested client to be notified if an update on a KNX group address occurs. A client is in this case another Java object that will get notified. The base oBIX object contract is defined through the Java interface TemperatureSensor, which provides a simple data point value with the current temperature and directly inherits from the base oBIX IOBJ interface. This representation is technology-agnostic and a generic implementation is provided through TemperatureSensorImpl. This allows to instantiate the object within the gateway and to perform oBIX related requests on the object representation. A technology-specific implementation, as it is done for KNX, has to subclass the implementation and to provide the related KNX connector API calls, in the case a read or write on the object occurs. The TemperatureSensorImplKnx provides this implementation and further relies on the watch dog mechanism to get notified if an update on the bus occurs.

The following UML sequence diagram (cf. Figure 34) illustrates the read of a KNX temperature sensor by issuing a CoAP get request. The request is first handled by the CoAP server which relies on the message decoder/encoder to parse the message. The oBIX read or write message is then passed to the ObixServer which retrieves the according oBIX object

\(^8\) https://github.com/calimero-project
through the *ObjectBrowser*.

Further, another application of the observer design pattern is done within the gateway design. Other oBIX related services like the watch service of oBIX or the CoAP get with observe option want to be notified if an object state has changed. For that purpose, a client object can subscribe to the changes on any subclass of oBIX `Obj` types. Based on this mechanism the oBIX watch service and the history service are realized and they automatically keep track of the changes and the history of objects. Also the CoAP server implementation keeps track of the CoAP clients that observe resources. The following sequence diagram illustrates the full interaction and the different observe relationships between the IoT6 gateway and the KNX bus and between the CoAP client and the IoT6 gateway.

**Figure 34 CoAP based GET on KNX temperature sensor**

**Figure 35 Observe relationships between a CoAP client and the KNX bus device**

### 7.5.2 BACnet protocol bundle

The BACnet protocol bundle reuses the open source BACnet4J library[^9] provided by Serotonin Software to interact with BACnet controllers. The structure of the bundle is similar to the KNX protocol bundle. The `BACnetConnector` class provides an API to access the BACnet devices and the BACnet specific implementation of the generic IoT oBIX objects map the oBIX related communication to the BACnet communication. Further, the custom BACnet

[^9]: http://sourceforge.net/projects/bacnet4j/
specific oBIX contracts for AnalogInput, AnalogOutput, AnalogValue, BinaryInput, BinaryOutput and BinaryValue have been defined as Java interfaces according to the used oBIX framework. For the device configuration either the BACnet objects can be mapped in an explicit manner to the generic oBIX device representation or an automated configuration based on the BACnet specific object representation can be done.

7.5.3 Wireless M-Bus protocol bundle

For the wireless M-Bus protocol bundle, a custom Java API needed to be created due to the lack of a proper open source alternative [21]. The main goal of the API is to provide a higher level of abstraction and in doing so providing an API (Application Programming Interface) that is easy to use and allows interaction with a meter device. Thus, the API should provide methods to receive and write telegrams as well as automatically process them. The API allows applications to interact with wireless M-Bus meter devices and process their telegrams in an automatic way. Data extraction functions have been put into place to provide telegram parts in a more meaningful way. In addition, support for encryption and decryption of the communication via the AES (Advanced Encryption Standard) standard has been added. Telegram headers can be parsed and their payload can be processed accordingly. Using this functionality it is also possible to process telegrams with various header formats. Since the idea was only to interact with meter devices only a subset of the entire M-Bus standard has been implemented.

Figure 36 Package diagram illustrating the schematic overview of API

Figure 36 provides a schematic overview of the API. It shows the main packages or components of the API and how they are related to each other. The main components are:

1. **Manager-Component**: This component is used to set up and initialize the overall API. Its basic functionality is to initialize the connector and the telegram manager itself. Although all other components can be used standalone, this component provides an easy way to initialize the API.
2. **Connector-Component**: To establish communication with a meter device a connector is necessary. This component establishes a connection via a COM port and listens for
incoming messages in a periodic interval. The messages are then handed to the telegram manager and processed from there.

3. **Telegram Manager-Component**: Retrieves incoming telegrams from the connector and initializes the parsing process. Telegrams are stored in raw format as well as in parsed format. In addition, the telegram manager provides several API functions that make the telegram values easy accessible.

4. **Wireless M-Bus Device Loader-Component**: As the API also provides integration into the IoTSyS project it has to provide necessary connecting interfaces. This loader component is used to map meter telegrams according to the oBIX standard so that they can be used in the IoTSyS framework.

### 7.5.3.1 API architecture

The prototype has been implemented in the JAVA programming language together with a test suite as well as a configurable set up for meter devices. To access the virtual COM port, a third party library RXTX has been used which takes care of handling communication over COM ports. To use RXTX, a library has to be registered in the JAVA_HOME/lib folder as well as the corresponding jar-File needs to be on the classpath.

```java
Packages:
at.ac.tuwien.auto.iotsys.gateway.connector.wmbus.reader
at.ac.tuwien.auto.iotsys.gateway.connector.wmbus.telegrams
at.ac.tuwien.auto.iotsys.gateway.connector.wmbus.telegrams.body
at.ac.tuwien.auto.iotsys.gateway.connector.wmbus.telegrams.header
at.ac.tuwien.auto.iotsys.gateway.connector.wmbus.telegrams.util
at.ac.tuwien.auto.iotsys.gateway.connector.wmbus.test
at.ac.tuwien.auto.iotsys.gateway.connector.wmbus.util
```

**Listing 15 Wireless M-Bus API package structure**

The listing above shows the overall package structure of the prototype and each package may contain multiple classes. The package reader contains all classes necessary to receive and send data to a smart meter device. In particular, this means that it listens on a configurable COM port for incoming messages. It is realized as a thread-based implementation so that messages can be received asynchronously from the overall application. Through a specific listener interface the application is then notified upon the arrival of new information/telegrams. The telegrams package consists out of several POJOs (Plain Old Java Objects) among other classes. These classes represent the telegram structure as JAVA objects and provide meaningful methods for accessing distinct parts of the telegram itself. Several of these objects are nested as a telegram consists out of both a header and a body where the body itself consists out of several objects.

Since a lot of the fields in the classes correspond to specific parts in the telegram, a utility package has been introduced which takes care of conversion or calculation operations. Additionally, it is used to construct an output that can be easily processed by consumers. Last but not least a test package is part of the prototype. Through this package several test cases can be tested without relying on the actual connection to the meter device itself. These tests include conversion, calculation, output and telegram construction. Additionally, there are also tests available with an actual meter device.

To give a more detailed view into the internal structure of the API, the class diagram has been split into several diagrams where each diagram focuses on a particular part.

Figure 37 gives an overview of how telegrams are constructed in the API. Telegrams are quite complex constructs since they can vary in length and field definitions depending on the
header and several other fields. Telegrams consist of several distinct parts. In any case, they have a header of a specific length (which can vary) and a body. The body usually consists of another header and the payload which contains the actual data values. Each field is represented through a class `TelegramField`. As there are multiple fields with different functions the `TelegramField` class can be seen as a template class for several other, more specific, classes. The payload of the body usually is made up out of variable data records presented as `TelegramVariableDataRecord`.

![Figure 37 Class overview for parsing W-MBus telegramms](image)

In addition to the telegram structure, a manager is necessary to control and store all telegrams. This manager is used to connect to the actual metering device as well as parse telegram values. The manager can be used to observe the connector itself and react to
incoming telegrams from the metering device. Figure 38 gives an overview.

![Telegram manager class diagram](image)

**Figure 38 Telegram manager class diagram**

To instantiate a telegram manager, the interface `TelegramManagerInterface` has to be implemented. This interface already provides the necessary methods to process incoming telegrams. Incoming telegrams are received in the `ComPortReader` class and are then handed over to the class that implements `TelegramManagerInterface` (in this case `WMBusConnector`). The interface enables the creation of custom telegram managers. The telegram manager is also responsible to encrypt and decrypt telegrams as well as parse them as soon as they are decrypted. Both values, raw and parsed, are stored in the telegram manager.

This concludes the overview of the API and the short introduction of its internal structure.

### 7.5.3.2 Gateway integration

For the integration into the IoT6 gateway, the wireless M-Bus API is used within a wrapper that provides the OSGI module and a connector class that can be used by the wireless M-Bus specific implementation of the `SmartMeter` oBIX base class.
The SmartMeter interface provides the mapping for oBIX objects and is specified according to the defined contract. The WMBusBundleActivator is used as an entry point for the IoT6 gateway and activates the device implementation through which the actual WMBusConnector is then initialized. The WMBusWatchDog interface allows registering a watchdog which is used to notify about new incoming data values.

### 7.5.4 Other protocol bundles
The internal design of the other protocol bundles is quite similar to the already presented designs of the protocol bundles. Therefore, a detailed description has been left out. Further details can be found at the open source project repository.

### 7.6 Local HTML5 control interface
For intelligence distribution within the IoT6 architecture a local HTML5 interface named Obelix provides a direct browser based user interface on the resources and devices integrated through the gateway. Therefore, an HTML5 file and according JavaScript and CSS resource files are provided with the gateway and are also served using the HTTP server. The HTML5 browser application relies on the HTTP/JSON binding of the IoT6 stack and enables a user to interface the system without the need of any fat-client installation. Obelix is also provided as open-source HTML5 and can be used as a control engineering interface. It is a generic oBIX client which directly operates on the Web service interfaces provided by the gateway component. An object browser is the central entry point for a user as illustrated in Figure 40. It lists the available objects based on a query using the oBIX lobby. Since everything is an object in oBIX which may have an arbitrary number of sub objects the structure is recursively analyzed and an entry is displayed in the object browser based on the name. The user can drag an element out of the browser into the object canvas for display and to update values.

For each oBIX object in the object canvas an object component is displayed. Following the oBIX object model, an object consists of sub objects which are either base value types like, for example, bool, int, real, str or complex objects. An object is rendered as component that provides HTML5 input elements for all base value types. For rendering the object, a simple GET request is performed on the object URI and the object structure is parsed dynamically. On a change of a base value property an according PUT request is performed and the object
is updated at the server side.

The object component allows a simple interaction with devices and virtual objects represented through oBIX objects. For engineering the group communication relationships, a graphical wire tool allows to group data points of different objects together. Whether a data point can participate in group communication or not is determined through a group communication object that is attached as child object to the basic value object. If such an object is present, connectors are displayed that can be used to graphically wire objects using a drag and drop mechanism. Once a connection is established a dedicated IPv6 multicast address is added to the according group communication objects.

The HTML5 control logic editor allows a direct user control on the devices and provides further means to configure communication relationships based on the IPv6 multicasting mechanism.

**Deployment**

Since the gateway is based on Java, it can be deployed on any platform that provides a Java 7
environment. For the execution it is possible to run the gateway as a standalone Java application or to deploy it within an OSGi container. Therefore, the IoT6 gateway can operate on any embedded PC platform, like for example the popular Raspberry PI\textsuperscript{11} or Beagleboard\textsuperscript{12}.

Figure 41 IoT6 smart board and Raspberry PI

Figure 41 provides an overview of the used platforms for the experimental evaluation. The use of Java as application platform comes at the cost of having no access to hardware specific operating system features. This is a problem if a direct interaction with device drivers for bus connectivity is required. However, the use of virtual serial interfaces provides a convenient way to avoid this problem and to use Java as gateway platform. For the connection to the heterogeneous technologies and bus systems various USB based connectors can be used, which provide a virtual serial interface. By using the RXTX library support for Java, this universal serial interface can be used to exchange telegrams with the different technologies. Figure 42 provides some example USB based connectors that can be plugged to the smart board depending on the required technology. In this way the platform can be kept flexible to the requirements of the environment.

Figure 42 USB bus connector examples for KNX, EnOcean, Wireless M-Bus\textsuperscript{13, 14}

For some platforms even extension boards for the various technologies can be found.

\textsuperscript{11} http://www.raspberrypi.org
\textsuperscript{12} http://beagleboard.org
\textsuperscript{13} http://www.bus-ware.de
\textsuperscript{14} http://amber-wireless.de/406-0-AMB8465-M.html
Figure 43 Demonstration deployment

Figure 43 illustrates an early stage of a test deployment, in which the IoT6 gateway provides access to a selected number of heterogeneous technologies, which will be described in detail in the following section. The demonstration setup was used at the IPSO Challenge 2013, in which the prototype was submitted to the challenge and was selected as one of the top 10 finalists.¹⁵


In order to test the multi-protocol interaction, a set of representative interaction use cases found within a typical building automation system has been used. To illustrate and to show the feasibility of a common multi-protocol integration heterogeneous technologies have been used to realize the defined use cases. The test environment involves state-of-the-art non-IPv6 based technologies combined with new IoT6 enabled sensors and actuators.

9.1 Multi-protocol interaction tests

For testing the multi-protocol interaction as methodology, an experimental evaluation based on a real test lab is done. Since a complete test of all possible protocol commands and interactions would be too complex a selected number of typical home and building automation use cases is taken into account to validate the multi-protocol interoperability. The aim of these scenarios is to show the benefit of the taken integration approach based on the IoT6 gateway using the IoT6 stack.

9.1.1 Lighting control

The most simple interaction scenario is the use of heterogeneous light switch actuators combined with push buttons of other technologies. For this scenario, BACnet and KNX switching actuators are combined with an EnOcean push button.

9.1.2 HVAC control
In this scenario, an existing HVAC control process based on BACnet is enhanced with KNX room automation devices and 6LoWPAN temperatures sensors. The BACnet controller is responsible for controlling a water boiler to create heating water and a chiller, which is used for the cooling process. The hot and warm water are circulated using pumps that are also controlled through BACnet and led the heating and cooling media to the heating and cooling registers. These registers are combined with fans and valves that allow the air flow and finally cool or heat a room to a desired set point.

Through the IoT6 architecture, the closed BACnet HVAC process can be enhanced with KNX devices. A KNX room thermostat allows adjusting the desired room set point and can further be used to visualize the current mode (comfort or standby) of the HVAC process. An EnOcean window contact sensor can be used to activate the standby mode if a window is opened.

Further, a smart meter can measure the current energy consumption in real-time and depending on a given policy the HVAC process can be adjusted to reduce the energy consumption. This can be done either in an indirect way by overwriting the room set-point, or directly by adjusting the mode of the chiller, boiler, fans and pumps.

9.1.3 Alarming
For the alarming scenario, the weather data can be used to inform the residents of an upcoming storm. If a storm is approaching and the windows are closed a simple visual notification and warning using a KNX text display can be done. If the EnOcean window contact sensor indicates an opened window at that time an acoustic signal is also provided to warn the inhabitants about the approaching storm.

9.1.4 Access control and room adjustment
In this scenario, a RFID reader is combined with a KNX switching actuator that controls an automated door opener. An employee identity card or a passport can be used for identification and upon a successful identification the door is opened. Further, personalized settings of the card holder can be used to configure the room temperature set point and lighting scenario.

9.2 Test environment
For the multi-protocol interaction tests the lab environment of the Automation Systems Group at the Vienna University of Technology is used.

Figure 44 RFID tag reader and KNX door opener, ID cards, passport
An RFID tag reader mounted next to the lab door provides access to lab room with further test equipment. To test the interaction RFID based employee identity cards and a passport is used.

### 9.2.1 KNX equipment

The KNX lab environment provides typical room automation devices, including a presence sensor, room lights, push buttons, a room thermostat, an air conditioning unit, sunblind actuator, window contact sensor and a room lighting model.

*Figure 45 KNX lab equipment - 1*
• Contact sensor

• Sun blind actuator

Figure 46 KNX lab equipment - 2
9.2.2 BACnet HVAC process model and Wireless M-Bus smart meter

Figure 47 BACnet HVAC process model

An existing room HVAC model based on BACnet has been further enhanced with a 6LoWPAN temperature sensor that measures the current room temperature. An electricity smart meter can be used to measure the current power load in real-time.
• BACnet HVAC model room equipped with a 6LoWPAN temperature sensor

Figure 48 BACnet HVAC model with smart meter and 6LoWPAN sensor

- Smart meter provides energy consumption and power value in real-time
9.2.3 Wired M-Bus floor heating model
A wired M-Bus floor heating model is used to test the M-Bus protocol bundle and IoT6 integration.

9.2.4 EnOcean wireless room automation
Several actuators and sensors based on EnOcean are available to test room automation integration.
9.2.5 6LoWPAN test bed

A 6LoWPAN test bed is deployed within the research group office area.

*Figure 51 EnOcean room automation components*
The fixed lab environment is accompanied by a mobile IoT test bed that provides a combined selection of the heterogeneous technologies and which allows illustrating the mentioned use cases scenarios.

**Figure 52 6LoWPAN test bed**
9.3 Manual gateway interaction test

For the interaction with the gateway the HTTP and CoAP protocol binding can be tested using standard Web browsers.

Figure 53 Web-browser based interaction with the IoT6 gateway
For the Firefox Web browser, a CoAP plugin exists (Copper\textsuperscript{16}) that allows interacting with CoAP enabled devices. The examples below illustrate the CoAP based interaction. A similar interaction is possible with the HTTP binding.

9.3.1 Device Discovery
For device discovery, the CoRE link format description is provided by the IoT6 gateway and allows a client to discover the available resources and their types.

Request:

\texttt{CoAP GET \textit{coap://localhost/.well-known/core}}

Response:

```plaintext
</watchService>;rt="obix:WatchService";if="obix"
</alarms>;rt="obix:AlarmSubject";if="obix"
</VirtualDevices/virtualPresence>;rt="iot:PresenceDetectorSensor";if="obix"
</VirtualDevices/virtualPresence/presenceStatus>;rt="obix:bool";if="obix"
</VirtualDevices/virtualBrightnessActuator>;rt="iot:BrightnessActuator";if="obix"
</VirtualDevices/virtualBrightnessActuator/value>;rt="obix:obj";if="obix"
</VirtualDevices/virtualBrightnessActuator/value/history>;rt="obix:int";if="obix"
</VirtualDevices/virtualBrightnessActuator/value/history/count>;rt="obix:int";if="obix"
</VirtualDevices/virtualBrightnessActuator/value/history/start>;rt="obix:abstime";if="obix"
</VirtualDevices/virtualBrightnessActuator/value/history/end>;rt="obix:abstime";if="obix"
```

The oBIX contract defines the resource type attribute.

9.3.2 Query sensor/actuator
With a CoAP GET request on the URL that identifies an oBIX object the according object can be retrieved. It is also possible to simple retrieve basic data point values. If the observe option is used the CoAP client is kept notified about changes on the current state of the object.

Request:

\texttt{CoAP GET \textit{http://localhost:8080/VirtualDevices/virtualLight/}}

Response:

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

9.3.3 Modify actuator
A CoAP PUT request allows to modify the current state of an I/O signal on an actuator or to set a virtual data point like a set point of a temperature controller.

Request:

\texttt{HTTP PUT \textit{http://localhost:8080/VirtualDevices/virtualLight/}}

Payload:

```plaintext
<obj href="/VirtualDevices/virtualLight/" is="iot:LightSwitchActuator">
  <bool name="value" href="value" val="true" writable="true"/>
</obj>
```

Response:

```plaintext
<obj href="/VirtualDevices/virtualLight/" is="iot:LightSwitchActuator">
  <bool name="value" href="value" val="true" writable="true"/>
</obj>
```

9.3.4 Further interaction scenarios
The IoT6 gateway allows further interaction scenarios to retrieve time-series of historic data

\textsuperscript{16} https://addons.mozilla.org/en-US/firefox/addon/copper-270430/
point data, a watch service for HTTP based interaction and a feed and alarming service.

9.4 Multi-protocol interaction use case scenarios tests

The following interactions have been used to test the multi-protocol interoperability. These tests represent a basic experimental evaluation. More formal integration and interoperability tests are scope of work package 7. The related test results will be published in the D7.2 deliverable.

The use case scenarios have been realized as simple Java applications, which are also deployable as OSGI module within the IoT6 gateway. A realization based on the local control & monitoring system or the cloud-based control and monitoring system also be possible and is going to be evaluated within the scope of WP7. Further, it is also possible to realize some of the use cases based on the logic components.

9.4.1 Lighting control scenario

For the lighting control scenario the group communication of the IoT6 stack is used. The bool data point representation of the EnOcean push button, the KNX light switch actuator and the BACnet light switch actuator are assigned to the same IPv6 multicast address (e.g. FF12::1). As soon as the EnOcean push button is pressed, the IoT6 gateway receives the telegram and checks internally the group communication table. It transmits the new value by making a CoAP PUT on the multicast address. It receives the multicast request by itself and checks again the related oBIX objects that reside in the same group and updates the corresponding values. This leads to the transmission of a KNX value write on the according KNX group address. Similarly, a write property request is performed for the related BACnet object.

![Figure 54 Lighting control interaction scenario - sequence diagram](image)

9.4.2 HVAC control scenario

One of the most sophisticated interaction scenarios is the HVAC control scenario. It uses the BACnet HVAC process model that operates on the various devices to control the room temperature. This process model is enhanced by a 6LoWPAN temperature sensor that monitors the current room temperature. A logic object realizes a two-point temperature inside the gateway and operates on the oBIX object representation of the various technologies. It observes the current room temperature, the current energy consumption with the W-MBus smart meter and the state of an EnOcean window contact sensor. A KNX room thermostat allows adjusting the desired set point with a relative value. It controls the BACnet control devices if the current room temperature is not close to the desired set point. If the window is opened the HVAC process is set to stand-by. This is visualized on the KNX room thermostat. Further, if a certain threshold of the current power consumption is reached the control value is reduced to a relative value.
9.4.3 Alarming scenario

For the alarming scenario the weather data is used. In case of an approaching storm the user receives a warning. This is realized through a KNX text display that provides a flashing light together with a simple text message. The user needs to manually acknowledge the warning in order to stop blinking light. If a storm is approaching and a window is open an acoustic alarm is added that requires the immediate attention of a resident. A simple Java based application based on the IoT6 stack interfaces is also deployed in the OSGI container and realizes the required control logic.
9.4.4 Access control and room environment settings scenario

In this scenario, a RFID reader is combined with a KNX-controlled door opener. Based on the employee card or the passport the access to the lab room is granted and further the environment is adjusted to the predefined settings for the person.
Conclusion

This deliverable presented the integration efforts within the work package 4. It first discusses different possible integration approaches and explained the design choices that led to the definition of the IoT6 stack. The integration architecture is illustrated and how it is embedded in the overall IoT6 system architecture. The features of the IoT6 communication stack are described and how the different communication principles of existing technologies can be mapped to the communication principles of the IoT6 stack.

The implementation and design of the Java based IoT6 gateway is sketched and how it can deployed on the smart board developed in work package 5. The integration into the local control and monitoring system developed in the scope of work package 4 is also described.

The work between work package 4 and 5 was strongly aligned which is shown through the seamless deployment on the smart board.

The feasibility of the proposed multi-protocol integration concept is experimentally evaluated with a set of heterogeneous lab devices.


