



Boosting Energy Awareness
with Adaptive Real-time Environments
INFSO-ICT-224557

Specification and interfaces to and from the sensing platform

[Report]

[Limited to BeAware consortium]

Document. Ref: [BeAware-D3.1-Basen-V0.5-20080922]

DELIVERABLE [D.31 WP3]

Programme Name:	ICT
Project Number:	224557
Project Title:	Boosting Energy Awareness with Adaptive Real-time Environments
Partners:	COORDINATOR: TKK (FI) CONTRACTORS: Helsinki University of Technology, TKK BaseN Corporation, BaseN Interactive Institute II AB, II Engineering Ingegneria Informatica, ENG University of Padova, UNIPD Enel.si, ENELSI IES Solutions, IES Vattenfall Research and Development AB, VRD
Document Number:	D3.1
Work-Package:	WP3
Deadline Date:	30th September 2008
Date of Delivery:	
Title of Document:	Specification and interfaces to and from the sensing platform
Author(s):	Topi Mikkola, Tatu Nieminen, Matti Lehtonen, Niilo Neuvo, Markus Harva, Katri Oinonen, Luigi Briguglio, Francesca Menici
Responsible Partner:	BaseN
Reviewer:	Luigi Briguglio
History:	v0.5 Final comments v0.6 Client java interface added v1.0 Formatting, comments added from TKK
Availability:	[limited to BeAware consortium distribution]

Table of Contents

- Executive Summary 4**
- 1. Introduction to BeAware Sensing Platform..... 5**
- 2. Modelling the Sensing Platform 7**
- 3. Sensing Platform Functional Requirements 11**
 - 3.1. Security..... 12
 - 3.2. Sensor Protocol..... 12
 - 3.3. Base station user protocol 12
 - 3.4. Google Protocol Buffers 13
 - 3.5. Wireless Sensing Nodes 13
 - 3.6. Sensor Base station 15
 - 3.7. Data Storage 15
- 4. Technical Specifications..... 15**
 - 4.1. Roles and Actors 15
 - 4.2. Sensor Protocol Use Cases..... 18
 - 4.3. Base station user protocol use cases 22
- 5. Modelling the use cases..... 27**
 - 5.1. Sensor Protocol modelling 27
 - 5.1.1. Wireless sensor communication 27
 - 5.1.2. Generalised Sensor Protocol 28
 - 5.2. Base station user protocol modelling 30
 - 5.2.1. GPB model of the BA-BUP 31
 - 5.2.2. Java client library for BA-BUP 37
- 6. Conclusions 38**

Executive Summary

This document contains the functional specifications for Sensing Platform, and its' protocol and interface specifications, both physical and software. Sensing platform is used to ubiquitously monitor household wide sensor networks, using energy consumption as an example. Data from sensor networks is consolidated to Data Storage for storing and analysis. Data Storage can be accessed either via end user applications, or through M2M interface. Data Storage also handles real-time pattern analysis and alarm generation.

Sensing Platform is designed to be massively distributed, allowing the system to gather data from multiple sources to Data Storage. Using non-local data storing enables system to analyse data based on either single household sensor network, or on multiple local networks, possibly correlated with third party sensor data. System also allows direct access from user to sensors in if real-time availability is needed. All services authenticate themselves, to prevent unauthorised access to the system, its' data or configurations.

Communication between BeAware services is implemented on top of standard protocols; functionally we are in OSI 5 and above. Energy consumption is used as an example, but the described Sensing Platform itself can be used for any similar sensor network. Sensing Platform is designed so that it can incorporate any sensor that complies with protocols defined here. It is also important to note that from the platform point of view, humans can act both as clients and sensors, giving feedback to the system.

The specification is organised into four main sections:

- Introduction – a brief introduction to the BeAware architecture
- Modelling – an overview of the system model
- Requirements - functional requirements for each layer and subservice
- Technical specifications – use cases, software APIs, protocols, data payload, and physical interfaces for each separate subsystem

1. Introduction to BeAware Sensing Platform

The BeAware project aims to provide household residents a better availability of the information about their electricity consumption, and provide guidelines on how this can be influenced. For this we need to model both electrical appliances and the behaviour of people using them. Furthermore, in order to build better models and thus increase our understanding we need to measure actual behaviour of household's energy consumption.

As shown in Figure 1 and Figure 2, BeAware consists of three separate layers. At the bottom is Sensing Platform, which consists of sensors, their base stations, data storage and associated real-time analysis services (WP3). On top of that Service and User Application layers offer services for more refined data. This document consists of interface definitions for Sensing Platform northbound interface and its' internal interfaces. Some parts like Analysis and Event correlation are shared between layers – real time correlation is done in Sensing Platform, while more CPU intensive analysis might be done in Service layer.

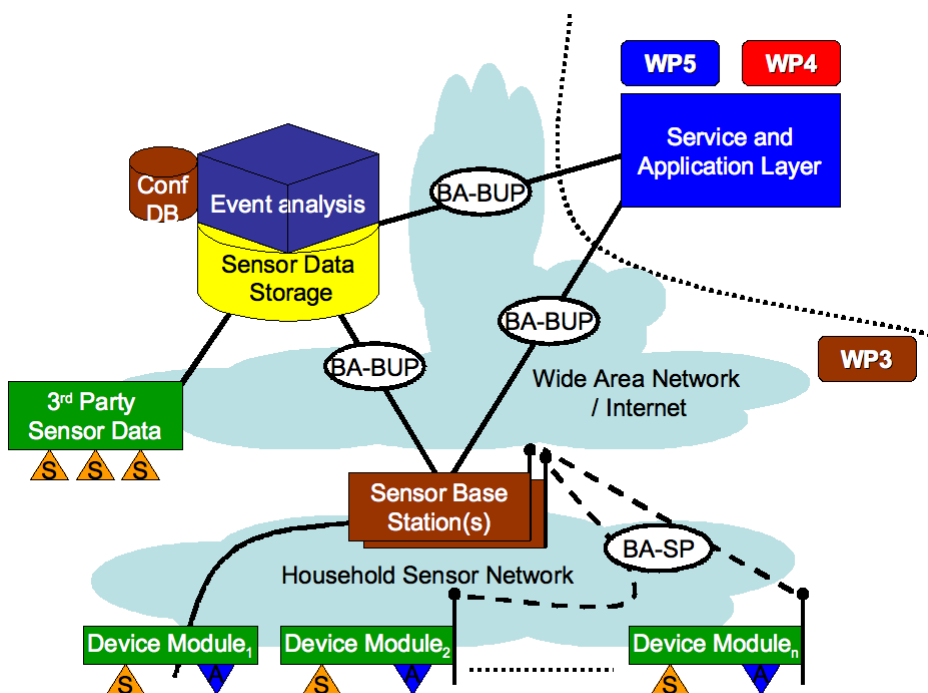


Figure 1 Sensing platform

Between the three layers there are two main protocols. As shown in Figure 1, they are:

- BA-SP - BeAware Sensor Protocol for messaging between individual sensors and a Sensor Base station. This protocol is mainly for unidirectional messaging from sensors to Base station
- BA-BUP - BeAware Base station User Protocol allows data retrieval from devices. This can be either Data Storage or Sensor Base station This also includes alert functionality.

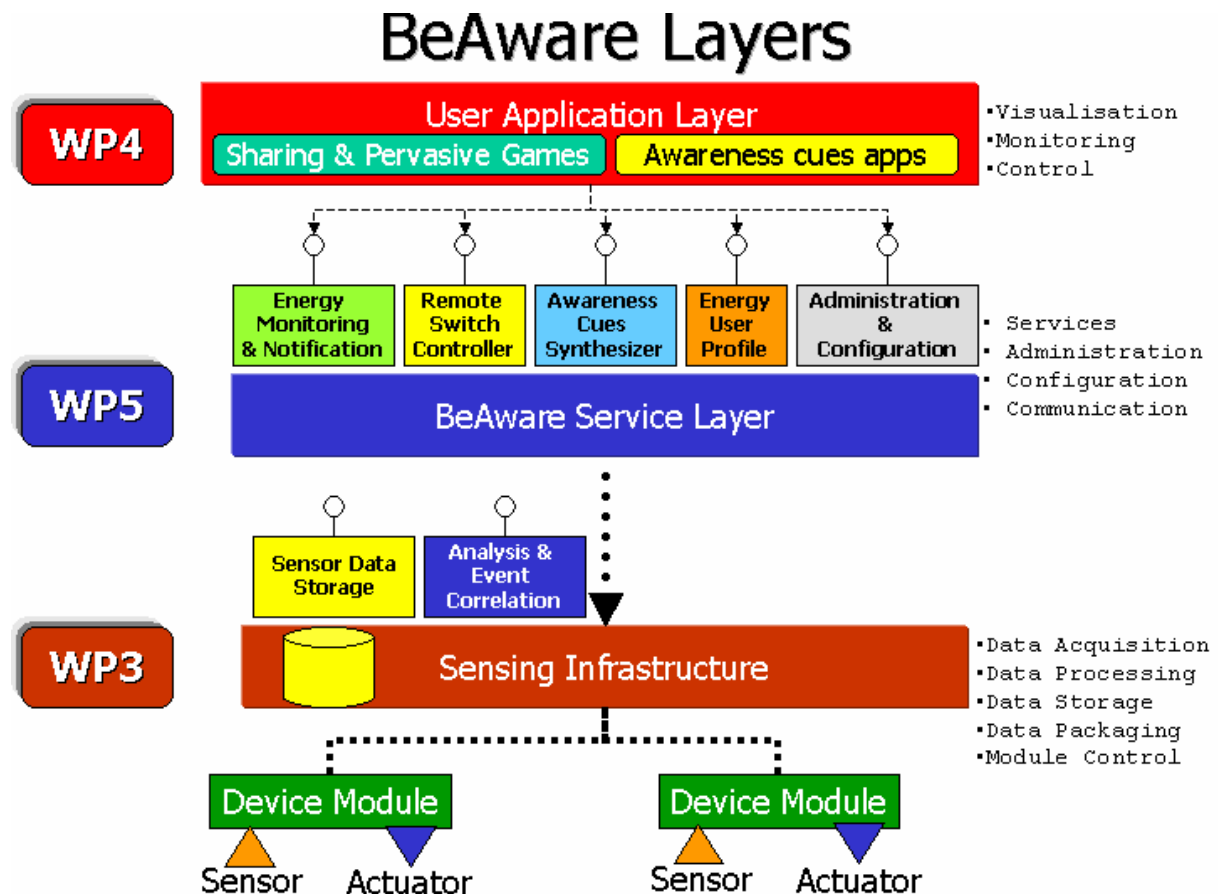


Figure 2 BeAware Layers

For this document's purposes, each layer and its services are treated as Black Boxes defining only the in- and outbound communication channels. This allows the overall design to be open and all needed protocols to be open source, while different kinds of service implementations can co-exist. It is not in the scope of this document to define the inner workings of each black box.

Terminology used in the document:

- Data Storage - a server or a grid of servers for data storage, event correlation, alert generation, and configuration management

- Sensing Platform - a combination of one or more Local Sensor Networks and Data Storage for them. Due to some confusion in BeAware Annex, also called Sensing Infrastructure in some documents.
- Local Sensor Network - a combination of Sensor Devices and the Sensor Base station(s)
- M2M - machine to machine
- Google Protocol Buffer (GPB) - a protocol specification language. (see <http://code.google.com/apis/protocolbuffers/>)
- Channel - an abstract entity in a Sensor Device over which measurement data or control commands are communicated
- Sensor Device - a device in a Local Sensor Network measuring values (e.g. power consumption) and sending them to Sensor Base station
- Sensor Base station - an intermediate device in a Local Sensor Network communicating sensor data to Data Storage or directly to users
- Actuator - a mechanical device for controlling a system (e.g. for turning the lights off to save energy when no one is home)
- RPC – Remote Procedure Call

2. Modelling the Sensing Platform

This chapter models the Sensing Platform main actors as UML and briefly discusses their roles. It does not yet take a stance on exact methods or variables, but acts as an overview to the system - it must also be noted, that their exact methods are not in the scope of this document, as they are not publicly available. More exact payload can only be modelled after requirements from other work packages have arrived. Whole model is based on a subscriber-source pattern, which tells the system what data is needed and where.

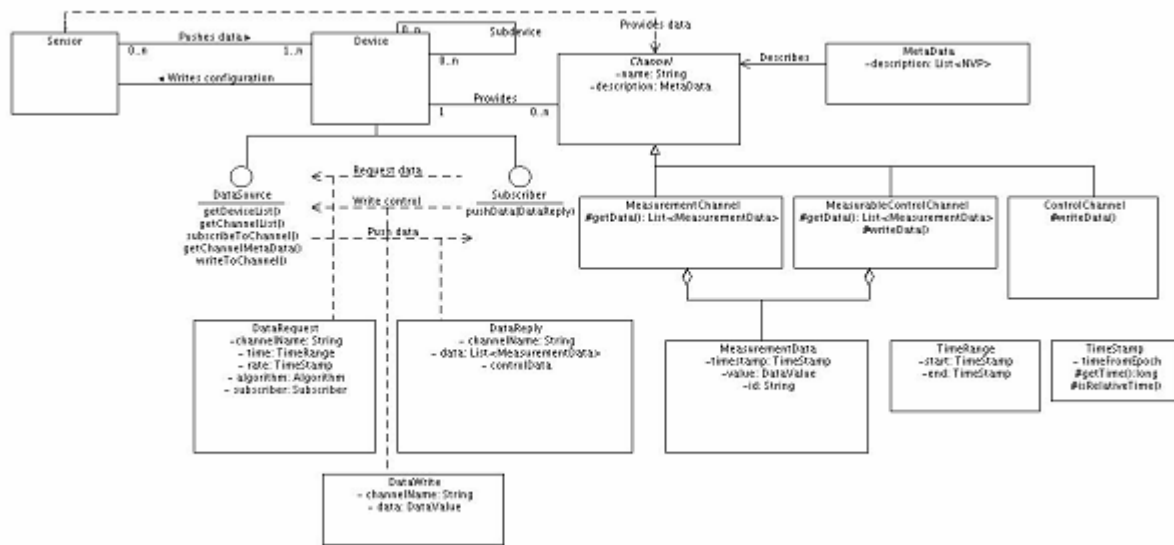


Figure 3 Sensing Platform main actors

As shown in Figure 3, conceptually the Sensing Platform has the following components.

- Device, which can contain both Sub-Devices and Channels
 - Sensor Base station is main level Device, Sensor measuring multiple things is a Sub-Device
- Sensor which reads data
 - Sensor has a type
- Channel which contains Data. There are three types of Channels
 - MeasurementChannel which contains MeasurementData and Metadata common to all channel entries. Sensors can write to MeasurementChannel, in normal cases all others can only read it
 - ControlChannel which contains ControlData for Sensors. Devices and only write to ControlChannel, its' data cannot be read once written to Sensor.
 - MeasurableControlChannel which contains ControlData for Sensors. Devices can write to it and read Sensors data
- MeasurementData contains TimeStamp and DataValue
- Metadata containing

- Description of Channel
- DataType of the channel (all Google Protocol Buffer data types are supported)
- ControlData contains DataValue
- TimeStamp can either contain exact times tamp, or relative times tamp if Device cannot know time.
 - SamplingRate is a special case of relative times tamp, denoting how often ChannelData should be read. It can either specify time period, data only when something changes, all all data read from sensor.
- TimeRange for data requests can contain either beginning and end time-stamps, or NOW for only most current data. Notice that TimeRange can also point to future, denoting that this request is a subscription that ends only later.
- DataSource which can be either a single Device or Data Storage for multiple devices
 - DataSource knows what devices are available for user
 - DataSource knows which Channels are available for each TimeRange
- Subscriber who requests data for some channel from DataSource
- Subscriber sends to DataSource a DataRequest which contains
 - List of Channel names
 - TimeRange
 - Applied Algorithm
 - SamplingRate
- DataSource sends Subscriber a DataReply containing
 - Channel name
 - List of MeasurementData
 - LastPacket flag, if this is the last DataPacket of an subscription
 - Other subscription or transmission specific data, if needed. Priority, sending window size etc. would be possible.
 - Ack requested for receipt of data

- DataWrite is used to write data to ControlChannel
- Algorithm specifies a mathematical function or set of them to be applied to specified channels. This can be either a set of predefined functions (SUM, AVERAGE, MAX, MIN) or some sort of algorithm definition.

As shown in Figure 4, Subscriber-DataSource pairs form the basis of communication, hiding the underlying details. From the end user point of view, DataStorage is invisible and reading data can be either via DataStorage or directly from Sensor Base station For most energy consumption applications using DataStorage is recommended as Base stations don't have much processing capacity, but both for prototyping and real-time applications direct access from end user to base station is needed. This also means that the Data Storage itself is not modelled in the system, as from the protocol point of view it should be invisible.

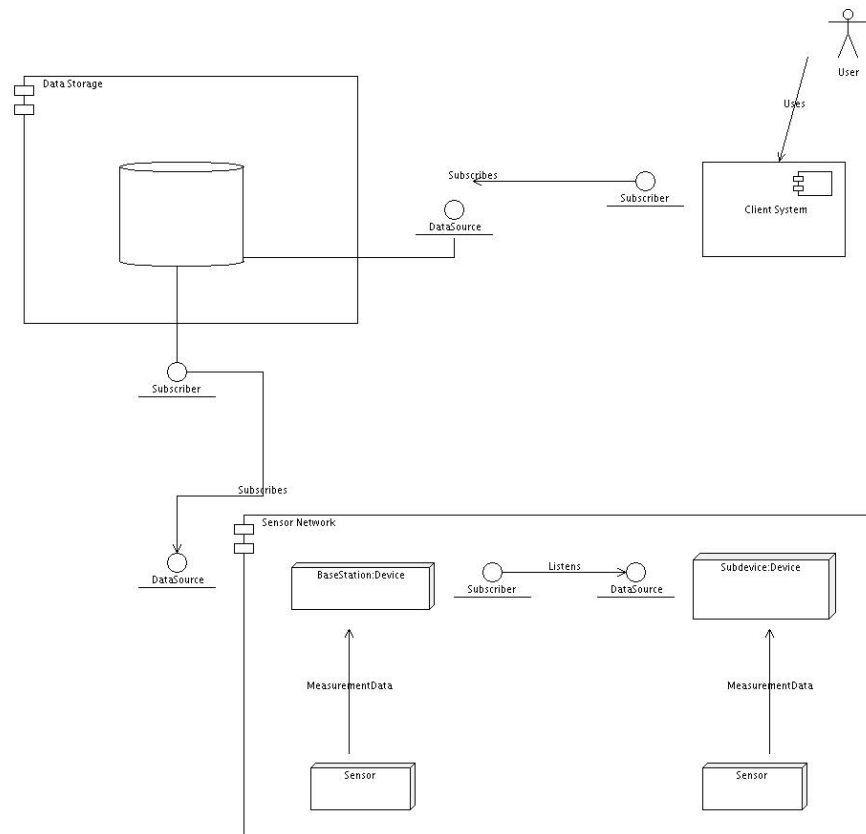


Figure 4 Sensing platform main interfaces

So for BeAware's case, direct access for base station over BA-BUP is not implemented unless needed, but for completeness sake, it is included in the design.

Realtime analysis of MeasurementData is a special case of applying algorithm to existing Channels. It creates a temporary MeasurementChannel which is not stored anywhere, only send to subscriber.

In practise most of BeAware sensors will not be modelled as objects as they are either transmitting byte arrays via radio, or measurement values will be read via general purpose io-pins in base station,

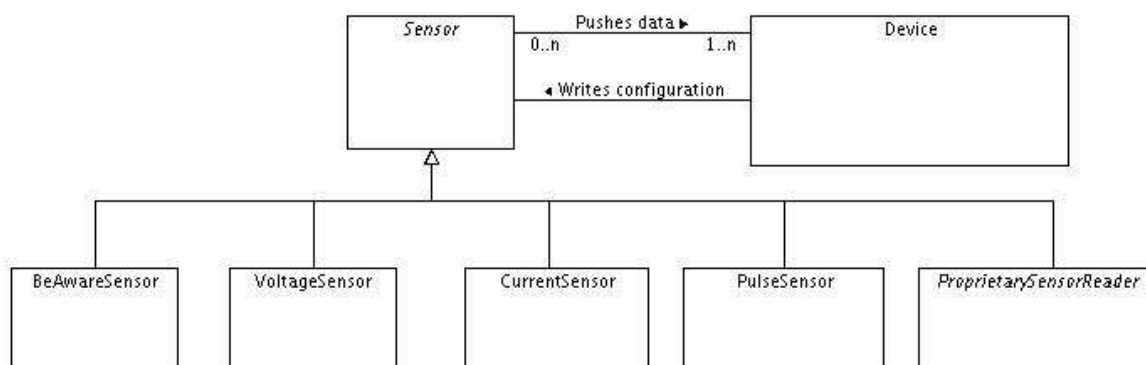


Figure 5: Sensor types

but in principle supported sensor types will be:

Of particular interest will be ProprietarySensorReaders, as quite a few commercial electricity meters have their own protocols (via RS232, RS485 or some such) and they must be supported case by case. It is just one design parameter for the base station.

3.Sensing Platform Functional Requirements

Functionally the platform must be able to perform the following basic tasks/answer the questions.

- What Devices does user see?
- What Channels does Device have on any given time range?
- Read Channel data over the given time range
- Write data to channel
- Apply algorithm to Channel, potentially creating virtual user specific channels
- Configure Device via ControlChannel
- plus authentication and authorisation, data encryption etc.

These requirements combined with the model presented in previous chapter suggests that two different protocols are used. These are

- BeAware Sensor Protocol (BA-SP) which handles the communication between Sensor and Device or Device and Sub-Device. It is compacted to the Sensor requirements.
- BeAware Base station User Protocol (BA-BUP) which handles communication between Device and rest of the system.

Both protocols will be asynchronous at heart, even if synchronous interfaces can be developed. They are based on subscriber-source patterns, sending request-reply data pairs, which are then matched based on systemwide unique message ids.

Some of these requirements are both for BA-SP and BA-BUP, some only for the BA-BUP. These functional requirements are here formulated both as UML use cases and Google Protocol Buffers.

3.1. Security

For the first iterations of the system, if HTTP is used as a carrier, its' security features can also be used for data encryption and authentication. Similarly HTTP reply status can be used for retransmission logic, if needed.

3.2. Sensor Protocol

Sensor protocol handles communication between Device and Sensor and inside a hierarchy of Devices. So it handles the following tasks:

- Sensor data reading
- Sensor identification
- Sensor configuration (also includes actuator handling)
- Mapping data to channel (basically Device acts here as conversion point between protocols)

3.3. Base station user protocol

Base station User Protocol handles all communication upward of a Device (SensorBaseStation). Its' tasks include:

- Device and Channel availability
- Data Reading

- Data writing
- Configuration
- Merging Channels (algorithms)

3.4. Google Protocol Buffers

For both BA-SP and BA-BUP the amount of data transferred might be critical. In BA-SP sensor has very limited processing capability and bandwidth to Base station might be restricted. And in BA-BUP's case while bandwidth itself is not a problem, amount of users connected to the system might be very high, so transferred data should be minimised.

This requirement alone rules XML-based formats out (own XML schema, SensorML etc.), as data needs to be as compact as possible. Also, XML processing with standard implementations in Java is very CPU extensive and scripting languages like Python will suffer from same problem. This is particularly problematic in Data Storage with almost unlimited number of potential connections.

Pure binary formats make data very compact, but marshalling and unmarshalling must be implemented at each supported platform. This particularly means that extending the protocol becomes much more tedious task. In effect we are developing an own version of binary protocol for Sensor Protocol for limited scenarios, but same approach is probably too work intensive for Base station User Protocol.

Google Protocol Buffers is Google's answer to the same problem. It has implementations for common programming languages and abstracts the data to a very human readable format via proto-files. This means that development time for working prototypes is cut, as development effort goes to higher level work than data marshalling. In our case particularly Java and Python implementations are of interest, with C/C++ also of interest with some sensor types.

Format is also open source which means that BeAware can, if necessary, develop its' own extensions and keep using the system even if Google drops the support at some point.

GPB in itself has very limited support for RPC, offering only stub generation. If RPC style communication is desired, it can be implemented on top of some simple carrier, for example HTTP. A client jar will be supplied which abstracts the carrier away from ServiceLayer. See Chapter 6.2.2.

3.5. Wireless Sensing Nodes

Wireless sensing nodes (sensors from now on) are ultra low power embedded radio transceiver systems build to measure different values. Basic sensor is measuring line voltage and current to calculate used energy accurately. Measurements is done with accuracy that differentiating connected appliances is possible. Basic sensors have also connection for external measurement devices. It accepts

as input at least: switch; voltage level 0 to 10 V DC and current 0 to 20 mA DC. Sensor provides power to external sensors if their power usage remains under order of few mill watts. Current generating measurement devices have to have their own power supply to keep sensor power usage at minimum.

Sensors get power from line where it is connected. Because target is energy aware technology sensors power usage has to be ultra low. Measuring many different values accurately and reasonable fast has its own requirements. Sensors average power usage should be less than 50 mW. Taking operating power directly from connected line makes risk of electric shock higher with connections to external devices. To minimise risk of electric shock external connections to measurement devices should be isolated from power supply of the sensor and from the sensors processor. Further to minimise power usage unused parts of sensor electronics should be possible to turn off without any problems to rest of the electronics or to measurement accuracy.

For handling measurements later on sensors need to time stamp all sent messages with time of measurement or action taken. All same network sensors need to be synchronised to base station of that network for the time stamping to be accurate. Required accuracy of time between sensors is less than a second. Timers or clocks are not accurate enough to keep constant synchronisation between sensor and base station. Sensors will monitor radio channel and base station will provide time and synchronisation to all sensors with proprietary protocol.

Basic sensors are designed to be installed easily and installation needs no special skill or other equipment. Sensors are housed in a small plastic box connecting to electric outlet on wall or extension cord. It will not hinder installation of other appliances or sensors to the next outlet socket. When new sensor is installed to networks base stations will know it automatically and user can set up necessary settings for it through the base station user interface. Different sensors don't need special settings for measuring but for handling and operating on measurement results and such some information of where and what kind of appliances are connected to it is needed.

Sent messages need to keep short enough to be able to have multiple sensors in same area. Messages are divided in types and all messages are same length to keep sensor protocol simple. Sensors will send measured energy and other measurements relating to that information with fastest rate of ones every two seconds. Other measurements from external devices are sent when they happened; i.e. on-off switch; or in a rate of ones in a minute or slower if no change in measurement; i.e. luminosity level, temperature.

Costs for making sensors should be small enough for them to be affordable in large amounts for general public to buy as many as they require. Final product manufacturing/component price should be less than 20 euros.

Measurement capability of sensor is 16 A on current and 270 V on voltage. Sensors will operate on any voltage level enough to give sensor operating power. Accuracy of measurements has to be good; error less than 1%; also on all voltage and current waveforms which fit electricity requirements.

3.6. Sensor Base station

BeAware Base station is an embedded radio transceiver system with memory and connection capability over TCP/IP protocol. Connection of base station is done with Ethernet or modem. For determining necessary operating requirements and other properties first base station prototypes will be combined from radio transceiver board with micro controller and a small computer with internet capability. Base station has to have user interface for settings, sensor management and quick view of measurement information.

Base station should have memory enough for few (2 - 4) days measurements in case of communication network failure and for local history. Memory type can be combination of embedded systems internal static memory and some other external if more space is needed, preferable types are flash memory cards (SD).

Radio operation of sensors and base stations is designed to operate over license free ISM band. At least first versions will use proprietary transmit protocol on frequency band 433 MHz between sensors and base station. For future improvements in electronics support for other protocols has to be considered to make it possible adding third party sensors. One of such protocols is IEEE 802.15.4 standard which is used as a base for ZigBee protocol. For now ZigBee is too complicated system for such system but it is evolving fast.

For measurement time stamps base station and sensors needs an accurate clock to at less than a second. Base station will provide all sensors time base and sensors will provide measurements based on that time base.

3.7. Data Storage

Data Storage gathers data from multiple Devices, storing the MeasurementData for each Channel. It also contains configurations for user and device authorisations to make sure no unauthorised access to system is allowed. Also, it must make sure that all relevant data can be accessed quickly and history data is stored.

4. Technical Specifications

In this section we first define the most common use cases for both protocols and then present GPB representation of them and outline the data flow between interfaces.

4.1. Roles and Actors

Main actors of the system are:

- Sensor - data measuring system which in basic case just pushes data upwards

- Device - listener for the sensor data, also acts as a DataSource for the BA-BUP
- User - any user of the system
- Provider - energy provider
- Consumer - energy consumer
- Administrator - administrator of the system, has wider operational privileges than normal user.

Their relations are shown in Figure 6.

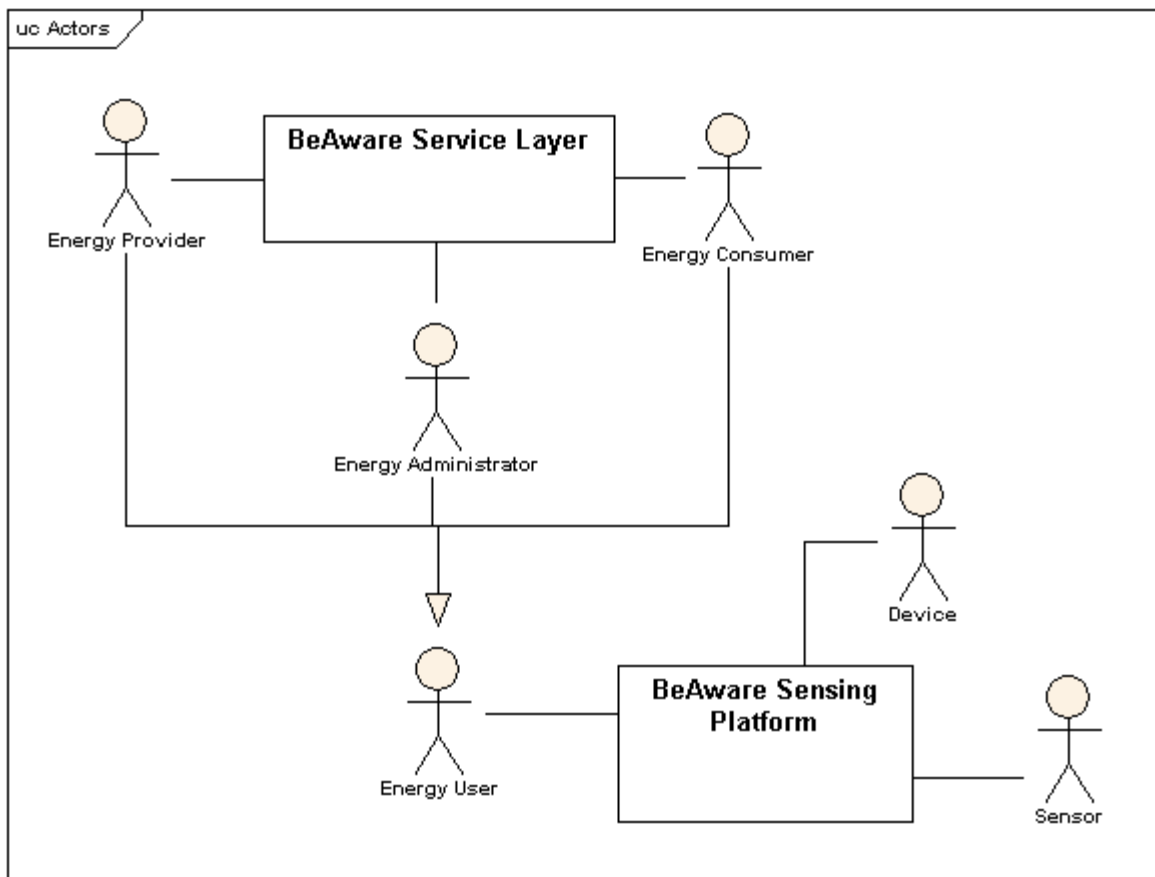


Figure 6: Use case actors

Use cases per actor for Sensing Platform are summarised in Figure 7. For clarity's sake, for each use case, only the main actor is shown.

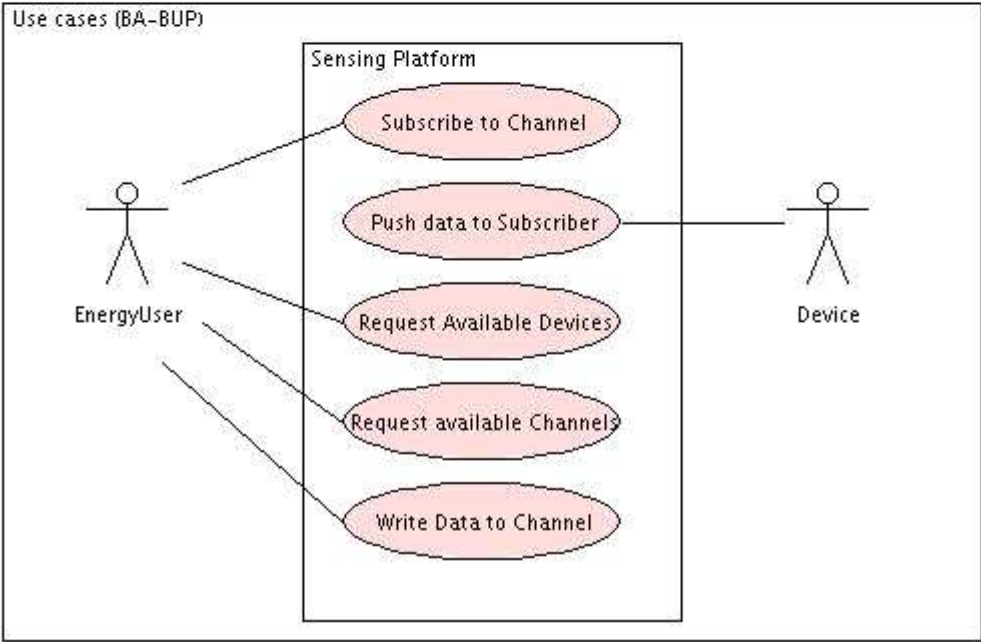
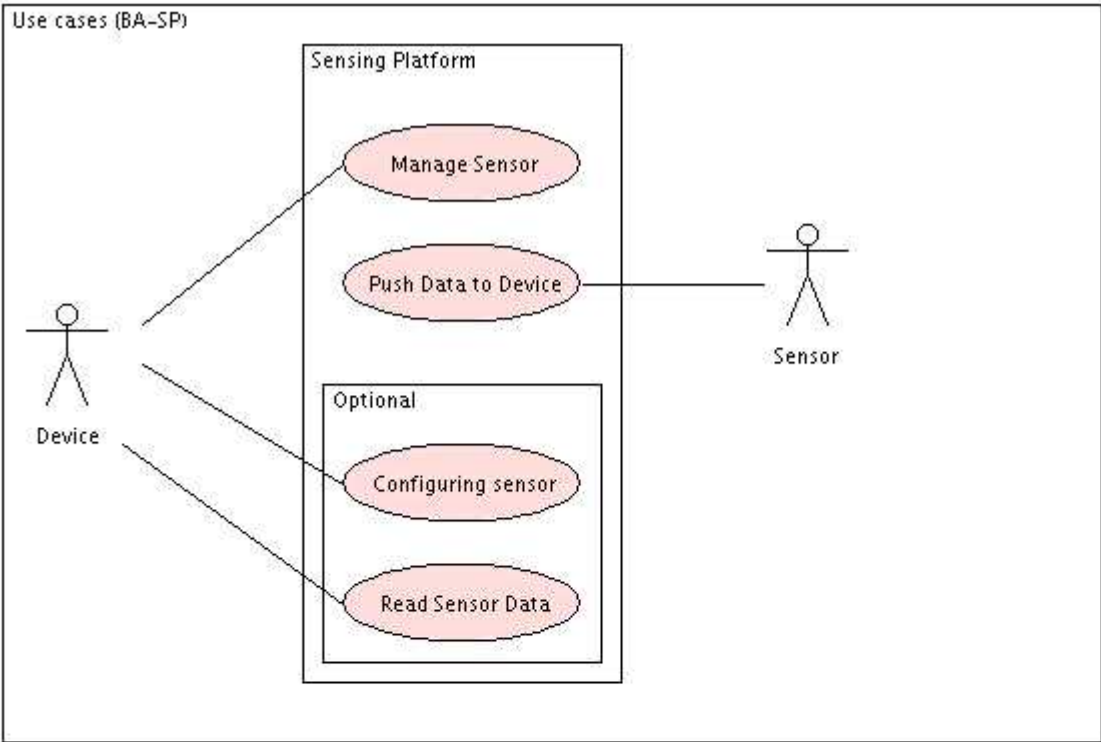


Figure 7 Sensing Platform use case actors

4.2. Sensor Protocol Use Cases

Use Case Name	Push Data to Device
Actors	Sensor
Preconditions	Device accepts measurements from Sensor and is configured to read data from Sensor's output. Sensor is configured to push MeasurementData to Device
Postconditions	
Description	Sensor pushes MeasurementData to Device
Scenario	<ol style="list-style-type: none"> 1. Sensor opens connection to Device 2. Sensor pushes MeasurementData to Device 3. Device stores MeasurementData to correct Channel 4. New Channel is created if none exists 5. Device acknowledges
Example	<ol style="list-style-type: none"> 1. Sensor reads kWh meter energy usage every minute 2. Sensor converts reading to MeasurementData 3. Sensor sends MeasurementData packet via wireless connections 4. Device stores MeasurementData into Channel DeviceId.SensorId.Energy
Alternative scenario	

Table 1: Use case: push data to device

Use Case Name	Configuring sensor (optional)
Actors	Sensor, Device
Preconditions	Device can open connection to Sensor, Sensor accepts configuration data. Device has new ConfigurationData to Sensor
Postconditions	
Description	Device sends configuration to Sensor
Scenario	<ol style="list-style-type: none"> 1. Device opens connection to Sensor 2. Device sends ControlData packet
Example	<ul style="list-style-type: none"> • Device has been informed that kWhMeter.relay should be closed • Device opens connection to kWhMeter • Device writes ControlData to Channel kWhMeter.relay • Device reads status of Channel kWhMeter.relay
Alternative scenario	Device cannot always read new state, for example if sensor has been configured to shut down communication.

Table 2: Use case: Configuring Sensor

Use Case Name	Manage Sensor
Actors	Device
Description	Device handles sensor management
Scenario	<ol style="list-style-type: none"> 1. New Sensor1 appears into network, reports to Device 2. Device checks if it has been configured to accept new Sensors of Sensor1.type 3. Device adds Sensor1 top list of accepted Sensors 4. New Channels are created for Sensor1 reading only when data is received. 5. (Device sends a DataRequest to Sensor1 and waits for data)
Example	<ol style="list-style-type: none"> 1. New wireless kWh Sensor is added 2. kWh Sensor informs Device that it is now operational 3. Device accepts connection 4. Device sends kWh Sensor DataRequest informing that it wants MeasurementData on Channel EnergyUsage, every 5 minutes, next 12h.
Alternative scenario	<p>New Channels for ConfigurationData created immediately</p> <p>Some Sensors (passive) are added manually</p>

Table 3: Use case: Manage Sensor

Use Case Name	Read Sensor Data (optional)
Actors	Device
Preconditions	Sensor which can only accept read data connections, probably a proprietary system Device supports desired read data system
Postconditions	
Description	Data is read from Sensor
Scenario	<ol style="list-style-type: none"> 1. Sensor.type indicates that data can only be read from Sensor 2. Device opens read connection to Sensor 3. Device sends DataRequest (or proprietary protocol command to same effect) 4. Sensor replies 5. Sensor reply is stored to proper Channel 6. Device closes connection
Example	<ol style="list-style-type: none"> 1. ModBus kWh meter acts as a slave to Device 2. Device sends command to meter reading register "energy" 3. Contents of register "energy" is converted to MeasurementData to Channel kWhMeter.energy and times tamped
Alternative scenario	

Table 4: Use case: Read sensor data

4.3. Base station user protocol use cases

Use Case Name	Subscribe to channel
Actors	EnergyUser
Preconditions	Client is authenticated, is authorised to fetch desired channels, C has configuration stating which Channel is needed
Postconditions	
Description	C requests measurement data for a set of Channels
Scenario	<ul style="list-style-type: none"> • C requests predefined Channels and time range desired • Data Storage checks which of the requested Channels are available and fetches data from them • DS returns requested Channel data to C
Example	<ol style="list-style-type: none"> 1. C requests OverallEnergy for a set of users for given month 2. DS checks which of the users have at least one measurement in their Channel OverallEnergy for given month 3. DS returns raw data for all found Channels
Alternative scenario	<ol style="list-style-type: none"> 1. Only a subset of requested Channels is returned, if all do not exists 2. C can request all Channels matching given criteria 3. C can also access Sensor Base station directly in some cases 4. C can also request TimeRange that extends to future. This is basically a subscription to data push. 5. Algorithm can be added, creating a virtual Channel

Table 5: Use case: subscribe to channel

Use Case Name	Push data to subscriber
Actors	Device
Preconditions	Subscriber has requested subscription to Channel from DataSource
Postconditions	
Description	Push data to subscriber
Scenario	<ol style="list-style-type: none"> 1. DataSource opens connection to Subscriber 2. Subscriber accepts connection 3. DataSource pushes DataReply to Subscriber 4. Subscriber stores data 5. Subscriber acknowledges receiving the data 6. DataSource closes connection
Example	<ol style="list-style-type: none"> 1. DataStorage has subscribed to SensorBaseStation to read its' Channels every minute 2. SensorBaseStation opens connection to DataStorage 3. DataStorage accepts connection 4. SensorBaseStation sends a DataReply to DataStorage 5. DataStorage stores MeasurementDatas for each Channel contained in DataReply 6. DataStorage informs SensorBaseStation that data was received 7. SensorBaseStation closes connection
Alternative scenario	

Table 6: Use case: push data to subscriber

Use Case Name	Request Available Devices
Actors	EnergyUser
Preconditions	Subscriber can contact DataSource
Postconditions	
Description	Request Available Devices
Scenario	<ol style="list-style-type: none"> 1. Subscriber opens connection to DataSource 2. Subscriber requests all available Devices 3. DataSource checks which Devices are available to Subscriber 4. DataSource sends a list of Devices to Subscriber
Example	<ol style="list-style-type: none"> 1. User Client opens 2. User Clients queries from DataStorage which devices are available to User 3. User Client shows a list of Devices to User
Alternative scenario	

Table 7: Use case: request available devices

Use Case Name	Request available Channels
Actors	EnergyUser
Preconditions	Subscriber is allowed to view Device
Postconditions	
Description	Request available Channels from a Device
Scenario	<ol style="list-style-type: none"> 1. Subscriber opens connection to DataSource 2. Subscriber requests Channels from Device for TimeRange 3. DataSource checks if Subscriber is allowed to view Device 4. DataSource checks if Device has any Channels available for TimeRange 5. DataSource checks Subscriber is allowed to access Channels 6. DataSource sends Channels to Subscriber 7. Subscriber closes connection
Example	<ol style="list-style-type: none"> 1. User picks a Device he wants to view 2. User picks a time range 3. User Client sends request to DataStorage 4. DataStorage fetches available Channels 5. User Client visualises available Channels
Alternative scenario	

Table 8: Use case: request available channels

Use Case Name	Write Data to Channel
Actors	EnergyUser
Preconditions	Subscriber is allowed to write to DataSource's Channel, Channel exists
Postconditions	
Description	Write Data to Channel . This is a way to Subscriber to control the system, usually data flows from Source to Subscriber. In a way this is just a special case of reversed Subscriber-Source
Scenario	<ol style="list-style-type: none"> 1. Subscriber opens connection to DataSource 2. Subscriber sends a DataWrite to DataSource 3. DataSource checks if Subscriber is allowed to write to Channel 4. Data is written to Channel 5. Subscriber closes connection
Example	<ul style="list-style-type: none"> • User wants system to switch lights off • User Client shows that Channel House.Room.LightSwitch exists and accepts On/Off commands • User sends an Off command to Channel House.Room.LightSwitch • Lights switch off
Alternative scenario	<p>Configuring a sensor is a special case of this</p> <p>Success of write operation might be readable from same Channel or observable elsewhere</p>

Table 9: Use case: write data to channel

5. Modelling the use cases

Based on the use cases and data model presented previously, protocols have the following data model and GPB descriptions.

5.1. Sensor Protocol modelling

For prototyping purposes, a test protocol outlined below is used and the fully functional BA-SP is developed parallel. BA-SP is so strongly tied to the real-world limitations on data length etc., that fully specifying everything at start is not feasible. But in all cases it must be remembered that the actual protocol should be easily modified and new things easily addable.

5.1.1. Wireless sensor communication

Initially there will be 4 input ports for various external sensors + output for powering them.

Component	Wireless Sensor
Responsibilities	Measuring the household environment
Output Interfaces	Power output for external measurement sensors
Input Interfaces	Current 4-20 mA Voltage 0-10 V Contact I/O (potential free) (1-wire) (power + digital port for measurement sensors)

Table 10 Sensor interfaces

Since the goal is to develop a cheap device which could be produced in a mass scale, it might be necessary to remove some of the above features from the final sensor, or to divide the developed device into two units: the cheap basic unit and the advanced model with all the possible hardware interfaces.

A central research problem is the intelligence embedded in the sensor and how the functionality is divided between the sensors and the base station, or even the database server. By developing the analysis techniques, it is possible to identify devices as they are turned on according to their electrical finger prints. Most likely these finger prints need analysis to be embedded in the sensor level. At first we transmit the raw sensor input to the data base where there is enough processing capability to process this information. Keeping the logic at the database server also has the advantage that it is easier to keep the system up to date and changes can be applied faster. If the logic is in the sensors this might require physical access to the sensor itself which is unfeasible in large scale.

The data messages the sensors send to the base station is initially specified as follows. Lengths are in bytes. The total message length is always the same (20 bytes):

bytes	1-4	5-6	7	8	9-20
SensorData	id	time base	timediff	type	data

Table 11 Message length

In the message *id* identifies the sensor. This identifier is assigned during the handshake. The the sensors provide a time stamp which is encoded in the *time base* and *timediff*, where *time base* is the time received from the base station in BCD-coded HHMM 24h format, and *timediff* is the number of elapsed seconds from time base. The message body is contained in the *data* block the type of which is encoded in *type*, which for now has two values POWER and TEMPERATURE. Messages of type POWER has the form:

bits	1-24	25-36	37-48	49-56	57-
PowerData	apow	pf	thd	cf	atypes

Table 12 Message payload

Where *apow* is the apparent power level (S) as a fixed point value with the two first bytes encode the integer part and the last byte is the fractional part. The power factor (*pf*) is encoded in fixed point where the first byte is the integral part and the following half byte is the fractional part. The total harmonic distortion (THD) is encoded in a similar fixed point encoding. The value *cf* encodes the crest factor (CF) in a fixed point of the first half byte encoding the integral part and the second half byte the fractional part. At the end of the message *atypes* is a list of application identifier codes one byte each.

All measured values are BCD-coded values. Appliance finger prints are determined from test measurements and they have certain limit values for detection [17]. Appliance codes are given in order the appliances were added. However, detecting different appliances from a simultaneous plugging is an underdetermined problem, i.e., when connecting an extension cord with multiple appliances behind a single socket the measurements does not give us enough information to decide uniquely which appliances were turned on. In case of blackouts the sensor might save the current detected appliance list and refer to that if the power usage stays the same after the power is restored.

Initially the response time requirement for the sensors is 2 seconds and the bandwidth should be high enough to enable 20 sensors per base station.

5.1.2. Generalised Sensor Protocol

In a more general view the same protocol can be expressed in GPB in the following way. Please note that all BA-SP communication is directed to broadcast address to which all devices listen and then

based on message identifier react to. Only requestData expects to have a reply, which will again be broadcasted, using the same messageId which receiver will then match. No data acknowledges will happen in normal case, but could be added to protocol later, if some data is deemed critical enough.

Sensor and Device are defined as services

```
service Device {  
  
  rpc receiveData (SensorDataReply);  
  
  rpc newSensor (NewSensor);  
  
}  
  
service Sensor {  
  
  rpc receiveConfiguration(SensorConfiguration);  
  
  rpc requestData(SensorDataRequest) returns SensorDataReply;  
  
}
```

The contents of the messages are still undefined, but should contain roughly.

```
message SensorDataReply {  
  
  optional long messageId = 1;  
  
  required long sensorId = 2;  
  
  required repeated Byte data = 3;  
  
}  
  
message NewSensor {  
  
  required long sensorId = 1;  
  
  required long sensorType = 2;  
  
}  
  
message SensorConfiguration {  
  
  required long sensorId = 1;  
  
  required repeated Byte data = 2;  
  
}
```

```

message SensorDataRequest {
required long messageId; //which will be matched in SensorDataReply
required long sensorid;
required repeated Byte data = 2; //specify request here
}
    
```

5.2. Base station user protocol modelling

The basic data query has 2 parts, request and asynchronous reply. In most cases reply should happen almost instantaneously, but in some cases data generation might take some time. Data flow here is modelled on top of RPC like protocol, in reality it is probably easier for prototyping that in the first iteration data is just transmitted over HTTP/PUT and RPC is added only after that. In both cases, the flow is modelled in Figure . In all BA-BUP examples, authentication is assumed to have happened beforehand, for example with HTTP. Subscription reply address is handled by the carrier protocol (tcp, udp), but cases where 2-way network is restricted (NAT, firewall) must be handled separately.

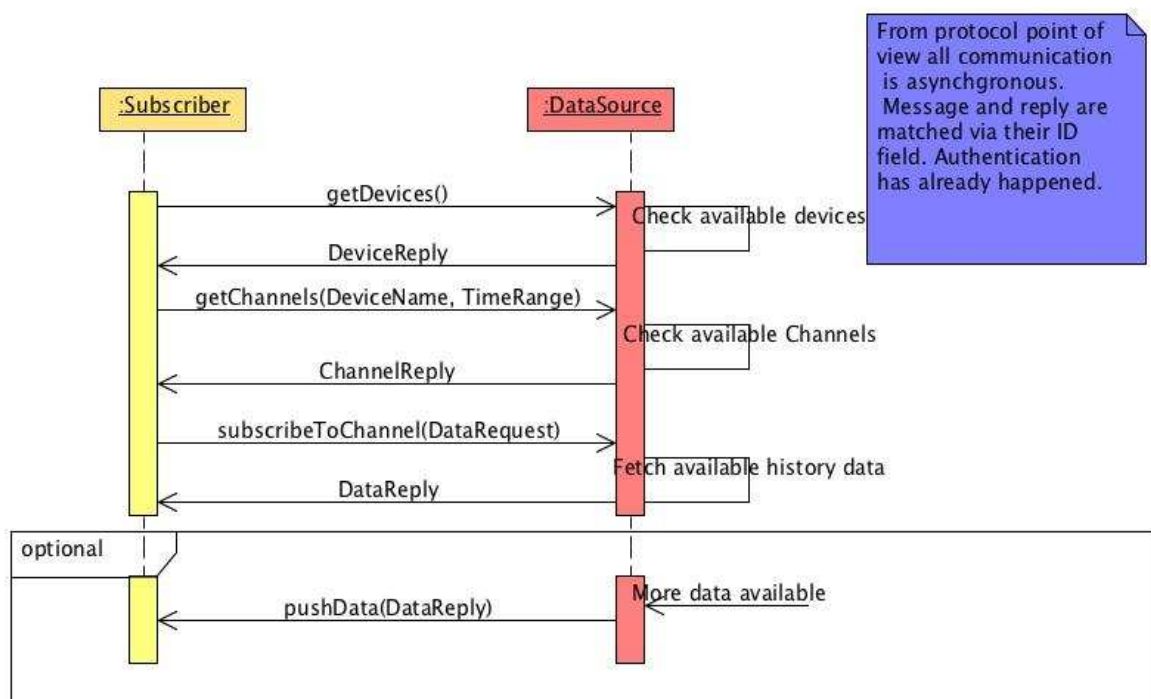


Figure 8: Data flow in data request

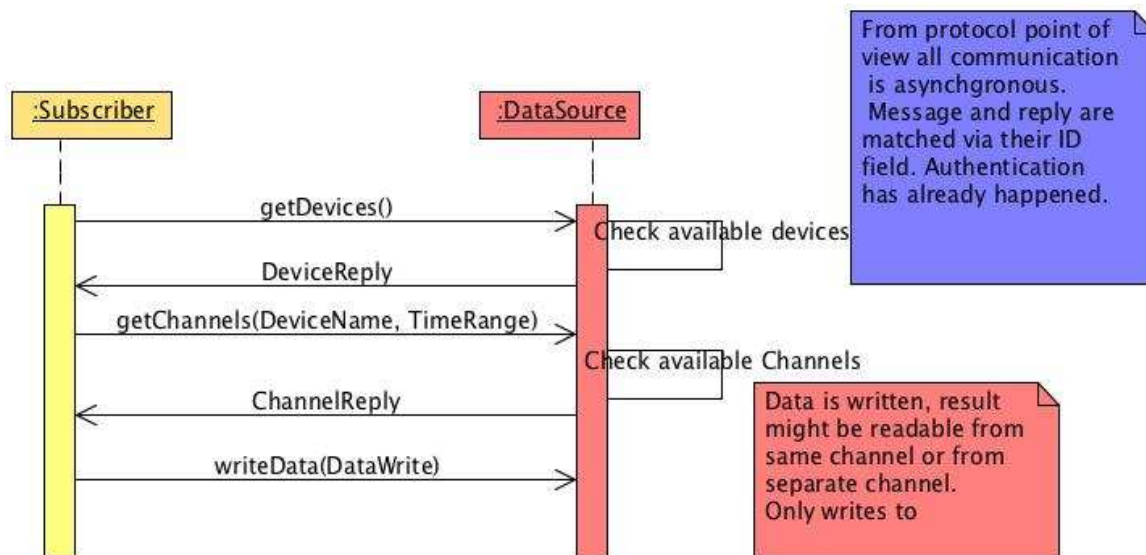


Figure 9: Data flow in data write

Please note that due to GPB imposed restrictions, UML based object model and implementation might differ slightly as the UML gives an object oriented view and GDB does not support inheritance and requires us to encapsulate even simple lists inside a message type.. UML is here to model the functionality and responsibilities, while GPB models the actual data. This is to keep the UML as readable as possible.

5.2.1. GPB model of the BA-BUP

With the GPB definitions, it is worth noting that for each data type, a separate DataReply type is needed. To make data conversions less error prone, only signed values are used. More data types will be added as needed, basically all GPB data types can be supported if needed.

All time/data data will be in UTC, regardless of location. Similarly only most commonly used time definitions will be used.

Reference implementations for Java/Python versions of protocol will be provided.

We define 2 services:

```
service DataSource {
```

```
rpc getDevices(DeviceRequest) returns (DeviceReply);
rpc getChannels(ChannelRequest) returns (ChannelReply)
rpc subscribeToChannel(DataRequest);
rpc writeDataLong(DataWriteLong);
rpc writeDataDouble(DataWriteDouble);
rpc writeDataString(DataWriteString);
rpc writeDataByte(DataWriteByte);
}
service Subscriber {
rpc pushData(DataReplyLong) return DataAck;
rpc pushData(DataReplyDouble) return DataAck;
rpc pushData(DataReplyString) return DataAck;
rpc pushData(DataReplyByte) return DataAck;
}
```

Associated message objects are:

```
message DeviceRequest {
  required long id = 1;
}
message DeviceReply {
  repeated String deviceName = 1;
  required long id = 2;
  optional Error = 3;
}
message ChannelRequest {
```

```
required long id = 1;

required String deviceName = 2;

required TimeRange timeRange = 3;

}

message ChannelReply {

    required long id = 1;

    repeated ChannelMetaData channel = 2;

    optional Error = 3;

}

message Error {

    required long errorCode = 1;

    optional String errorMessage;

}

message DataWriteLong {

    String channelName = 1;

    repeated Data {

        String name = 1;

        Long value = 2;

    }

}

message DataWriteDouble {

    String channelName = 1;

    repeated Data {

        String name = 1;

        Double value = 2;

    }

}
```

```
}  
  
}  
  
message DataWriteByte {  
  String channelName = 1;  
  repeated Data {  
    String name = 1;  
    repeated Byte value = 2;  
  }  
}  
  
message DataWriteString {  
  String channelName = 1;  
  repeated Data {  
    String name = 1;  
    String value = 2;  
  }  
}  
  
message DataRequest {  
  required repeated ChannelName channelName = 1;  
  required TimeRange timeRange = 2;  
  optional RelativeTimeStamp samplingRate=3;  
  required long id = 4;  
  //add algorithm here later, when we have defined what is needed  
}  
  
message DataReplyLong {
```

```
required long id = 1;  
repeated long data = 2;  
required String channelName = 3;  
required TimeStamp startTime = 4;  
required RelativeTimeStamp interval = 5;  
optional Error error = 6;  
}
```

```
message DataReplyString {  
required long id = 1;  
repeated double data = 2;  
required String channelName = 3;  
required TimeStamp startTime = 4;  
required RelativeTimeStamp interval = 5;  
optional Error error = 6;  
}
```

```
message DataReplyDouble {  
required long id = 1;  
repeated double data = 2;  
required String channelName = 3;  
required TimeStamp startTime = 4;  
required RelativeTimeStamp interval = 5;  
optional Error error = 6;  
}
```

```
message DataReplyByte {  
  required long id = 1;  
  repeated byte data = 2;  
  required String channelName = 3;  
  required TimeStamp startTime = 4;  
  required RelativeTimeStamp interval = 5;  
  optional Error error = 6;  
}
```

```
message ChannelMetaData {  
  required String channelName = 1;  
  enum ChannelType {  
    MEASURABLE = 1;  
    MEASURABLECONTROL = 2;  
    CONTROL = 3;  
  }  
}
```

```
enum DataType {  
  LONG = 1;  
  STRING = 2;  
  BYTE = 3;  
  DOUBLE = 4;  
}
```

```
message TimeRange {
    optional TimeStamp start = 1;
    optional TimeStamp end = 2;
}

message TimeStamp {
    optional long timeSinceEpoch;
    optional RelativeTimeStamp relativeTime;
}

//time is always UTC, only universally recognised

message RelativeTimeStamp {
    double multiplier = 1;

    enum Range {
        NOW=1;
        CURRENT=2;
        MINUTE=3;
        HOUR=4;
        MILLISECOND = 5;
        SECOND = 6;
        FOREVER = 7;
    }
}
```

5.2.2. Java client library for BA-BUP

A java client interface will be provided as a reference. The following methods will be included initially, in the final version both synchronous and asynchronous data requests will be supported. Whole project should have a common hierarchy of exceptions, they are grouped here as BeAwareExceptions.

Similarly, an authentication method for customer has to be designed for whole project. The client library will hide the protocol details from end users.

public List<String> getDevices() throws BeAwareException - fetch a list of Devices (BaseStations) the user can see

public List<ChannelMetaData> getChannels(List<String> deviceId, TimeRange range) throws BeAwareException - fetch a list of channels available to the user on a given time period

public List<DataValue> getChannelData(String channelId, TimeRange range, RelativeTimeStamp samplingRate, Map<Object, Object> params) throws BeAwareException - get the channel data on a given time period with desired sampling rate. Params map for additional parameters, like algorithms.

public void writeToChannel(String channelId, DataValue data) throws BeAwareException - write value to the given channel.

6. Conclusions

The specification is aimed at providing an overall picture of data flow within the BeAware system. A more specific view of Sensing Platform is provided, keeping in mind that this is a prototype design that must be allowed to live as the system is designed. The following principles should then be kept clear:

- Separation of tasks. Each component has clearly divided tasks, communication is abstracted via interfaces.
- New elements and functionality must be easily addable
- Implementation under the interface does not matter for the client
- Data and how it is transferred are 2 different things

The following items must be agreed between all parties:

- Authentication: how user is authenticated to the system
- Authorisation: what data user can access
- Encryption: at which points and how
- Unique id generation: how unique ids are generated