INFSO-ICT-224557

BeAware

Boosting Energy Awareness
with Adaptive Real-time Environments

Instrument:	CA	STREP ✓	IP	NOE

ICT - Information and Communication Technologies Theme

D3.10 Public summary of Sensing Platform

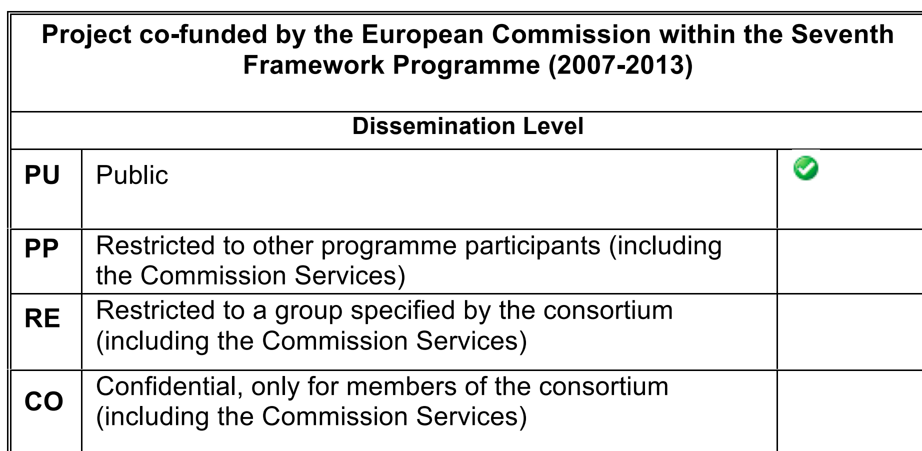
Due date of deliverable (as in Annex 1): April 30th 2011

Actual submission date: May 5th 2011

Start date of project: May 1st 2008 Duration: 36 months

Organisation name of lead contractor for this deliverable: BaseN

Revision: 1



Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	✓
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

European Commission
Information Society and Media

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:	D3.10
Work-Package:	WP3
Deadline Date:	30.4.2011
Date of Delivery:	5.5.2011
Title of Document:	D3.10 Public summary of Sensing Platform
Author(s):	Topi Mikkola, Fitta Manyazewal, Solomon Biza
Responsible Partner:	BaseN
Reviewer:	Tatu Nieminen
History:	
Availability:	[public]

Table of Contents

Executive Summary 5
1 Introduction 6
1.1 Supported use cases 6
1.2 Design principles, system requirements and licensing 7
2 Architecture 7
2.1 Sensing platform 8
2.2 Data receiver agents 10
2.3 Data storage 10
2.3.1 Inbound data 11
2.3.2 Outbound data 11
2.3.3 Data filtering 12
2.3.4 Available services 12
3 Interfaces 14
3.1 Babup 14
3.1.1 Using babup 15
3.2 Southbound data receiver interface 25
3.3 Northbound client interface 26
3.4 Other supported protocols 28
3.4.1 Base station 28
3.4.2 Data receiver 29
4 Analysis capabilities 29
4.1 Underlying system 29
4.2 Usage calculations 30
4.3 Baseline 30
4.4 Standby detection 30
4.5 Usage cycles 31
4.6 Alerting user 31
4.7 Advice 32
5 Load fingerprinting 32
5.1 Overview 32
5.1.1 Resistive load 33
5.1.2 Power electronic load 33
5.1.3 Motive (inductive) load 34
5.2 Fingerprints and load library 34
5.3 Fingerprinting process 35
5.4 Steady state detection 35
5.5 Main type detection 36
5.6 Device subtype 37

5.7 Multistate devices 37
5.8 Load disaggregation 41
6 Sensing infrastructure changes 43
7 Results 44
7.1 Advances in state of the art 44
7.2 Challenges encountered 45
8 Future 46

Executive Summary

This document covers the second part of wp3, namely the Data Storage. Data storage is meant for gathering, storing and analyzing all the data that arrives from BeAware sensors and various 3rd party sources.

The first part of this document gives an overview of the system and its' logical parts, individually and as a part of whole BeAware system. Also a brief overview of the underlying cloud system is given.

The second part documents the available public interfaces and services available via them. Also, the chose transmission protocol BABUP, a Google Protocol Buffer based system, is described and details on how to use it are provided.

The next chapters give an overview of various analysis capabilities offered and gives an example of how they are used in device fingerprinting via power quality analysis.

Finally we discuss how this part of the project advanced state of the art, what problems the project discovered and the future of the wp3 results.

1 Introduction

BeAware sensing layer consists of 2 logically separate parts. D3.7 describes the physical Sensing Infrastructure, so this document concentrates on the Data Storage and just gives an update to new functionality available to Sensing Infrastructure. Data Storage handles the collection of incoming data from base stations, stores and analyzes it and provides upper layers of the system, namely Service Layer, interfaces for fetching data.

Data Storage has been built to support high granularity data from multiple sources so that all the data is available in raw, non-processed format for later studies.

Data Storage has been built on top of BaseN's proprietary computation cloud, so not all details are public. This document gives also a brief overview of the internal workings of the cloud - all the interfaces and protocols are public. Reader should note that all the algorithms have been made public domain and are published in the BeAware public software repository.

1.1 Supported use cases

The required uses cases for the Data storage come from requirements documents D3.1, which in case derives requirements and use cases from D4.1 and D5.1 D4.1. The base station uses cases have been covered in previous documents, so data storage needs to cover the following use cases:

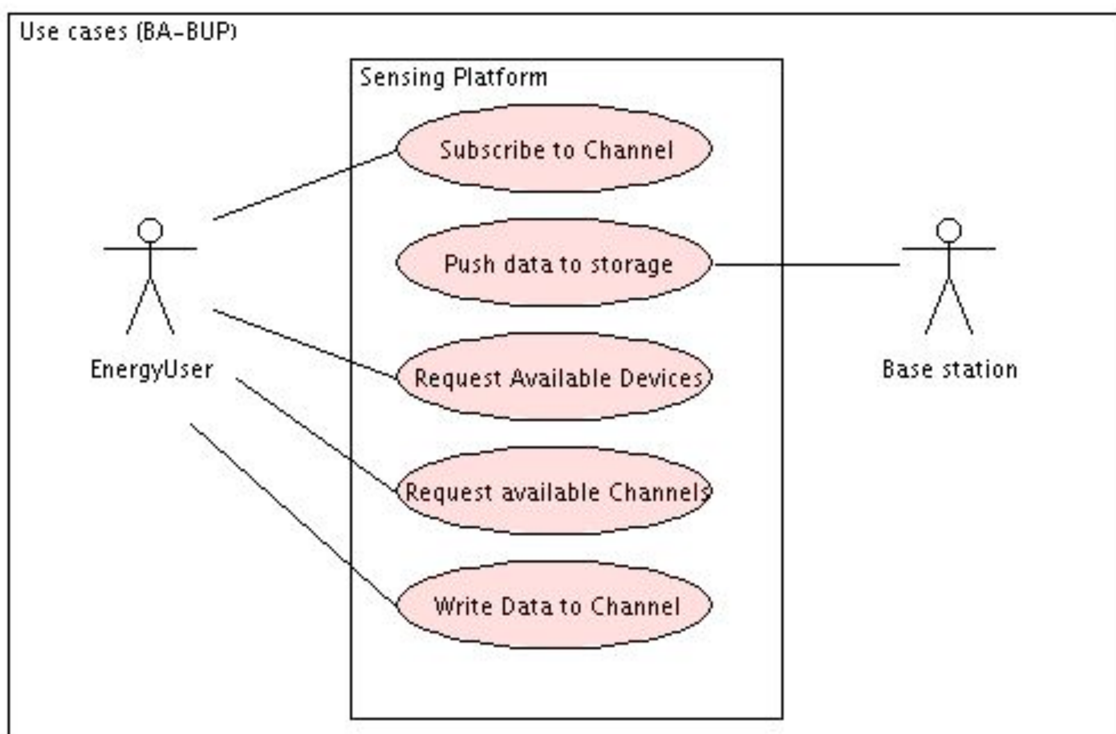


Figure 1 Supported use cases

- Subscribe to channel: BeAware was initially thought to work on subscribe-publish pattern working asynchronously, but was later changed to synchronous reading via method calls.
- Push data to storage: Data storage supports data writes for the data receiver interface:
- Request available devices: EnergyUser is allowed to see only a configured subset of all available base stations. (Base stations were originally called Devices.)
- Request available channels: EnergyUser is allowed to see only a configured subset of channels beneath every base station.
- Write data to channel: EnergyUser is allowed to write both measurement data and configuration data to configurable set of channels.

1.2 Design principles, system requirements and licensing

With cloud and grid computing, the system can be optimized for several things. In this case the main design principles have been near linear scalability of processing power and redundancy of both stored data and services themselves, both on physical and logical level.

Based on the use cases above, the main requirements identified were:

- System must be able to handle measurements at 1Hz level
- System must be able to update UI within a minute of measurement value being received.
- Algorithms must be configurable from external source, without need to recompile

The computation cloud itself is used as a black box from BeAware point of view. The overall architecture and functionality of APIs is covered here.

All BeAware code (protocol buffers, Java library to access the system plus all developed algorithms) are available as part of the BeAware SVN repository under Lesser GNU Public License. All base station code is available under Gnu Public License.

2 Architecture

In the layered BeAware architecture (see Figure 2), the Sensing layer forms the 2 lowest levels. The physical sensing infrastructure consists of sensors and base a station as described in D3.7, and each of the base stations is connected to the data storage via a

data receiver agent. The data storage itself is a service discovery base computation cloud with various services available. The following chapters explain the logical parts of the sensing platform.

Figure 2 BeAware whole picture

2.1 Sensing platform

The base station and data storage parts of the sensing platform are completely decoupled, all communication happens via BABUP (see 3.1) over HTTP. So the base station itself can be thought just as an example on how to implement a home sensoring system - it is completely hardware and language independent, as long as the protocol is supported. Similarly, the client interface can be accessed by any system via BABUP over HTTP.

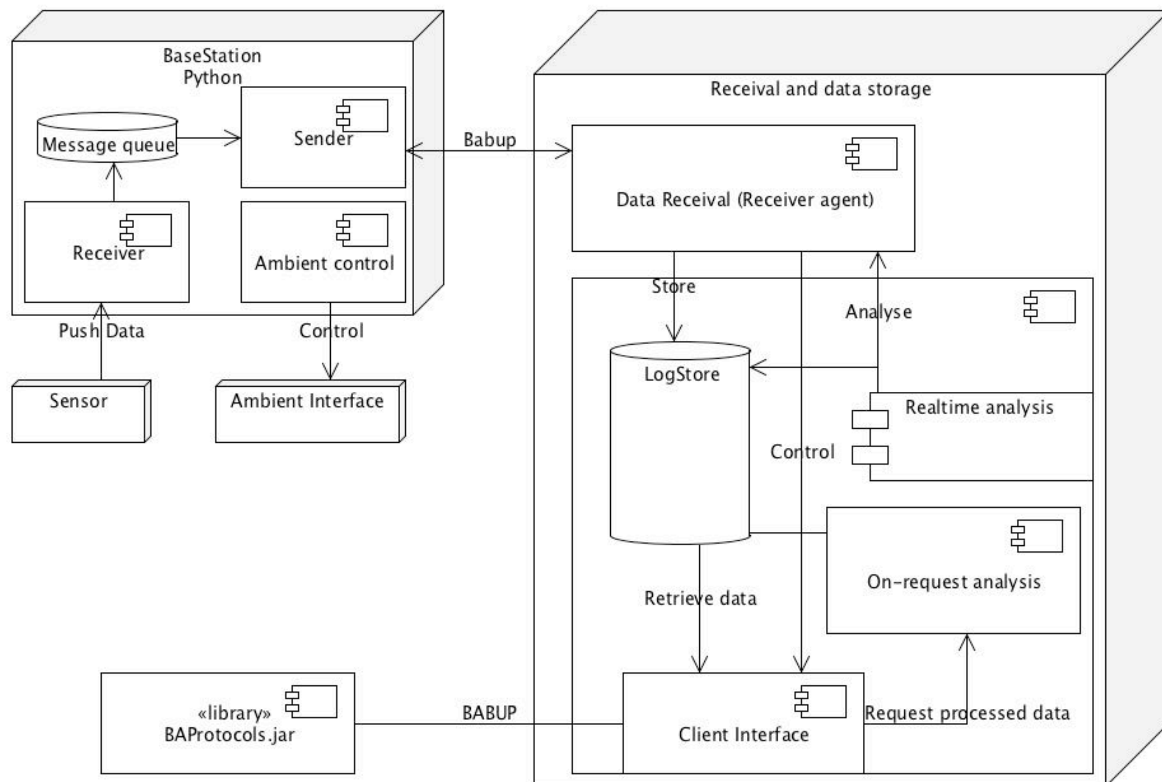


Figure 3 Data storage components

One of the distinguishing features of BeAware is the amount of data read from a household. Unlike normal AMR projects where cumulative energy is read from 1/h to 1/min at best, each BeAware sensor provides roughly 9 measurement values per 2 seconds, meaning that a household equipped with a complement of 9 BeAware sensors and 1 BeAware pulse counter produces 2490 measurements per minute. Each measurement is 64bit, and is tagged also with a 64bit timestamp and varying length sensor id tag. For production purposes id should also be 64bits (for completely unique sensor id) plus 8bits (data channel identifier - power, energy, thd etc), as in trials complete sensor id has been kept mostly for debugging purposes even if it wastes some space . For babup in Google protocol buffer wire format, outgoing dataflow from a household can be estimated to be 40 kB/min, with gathered data being sent twice a minute.

The architecture has been from start designed to be such that it scales easily.

- The base station is able to cache data for several years, meaning that no data is lost, even if due to network bottlenecks data gets queued
- The number of data receiver agents can be changed if more processing power is needed. Basically base stations can periodically check to which receiver agents they send data and load can be thus balanced on the fly.

- Service discovery architecture means that new processing power can be added on the fly to parts that need it. All services are logical and each physical computer can run multiple services.
- In beaware case the more static data like baselines can be precalculated and cached. The cache hits on these cases is near 100% (last weeks consumption etc) and cached data is neglible in size.
- Both data receiver and analysis are configurable via configuration templates, so adding new measurement points or new analysis options for all users is easy.

The computation cloud itself is protected with normal firewalls and access control lists, only the data receiver agents and client, export and visualization interfaces are available to public internet and they all required authentication to access.

2.2 Data receiver agents

The data receiver agents handle only two main tasks. They process authenticated data from all base stations and other measurement points, and send that data in processed format to the actual data storage via another authenticated and secured channel. Chapter 3.2 contains the explanation of actual data receiver. In BeAware's case data receiver accepts Babup messages and also periodically fetches Finnish meteorological data. It has a capacity to interface with a variety of other measurement sources, detailed in Chapter 3.4.2.

The receiver agents are usually single dedicated computers running only the data collection. In normal production systems they are always at least duplicated, but for the BeAware the testing system was run on only one agent machine. Even with project lasting only a couple of years, this highlighted how important it is not to have a single point of failure, as broken hardware ended in a few day break in BeAware test-environment data receiver. Of course, data caching in base stations meant that no data was lost, but the real time functionality was lost for that period.

2.3 Data storage

Data storage is a computation cloud, handling 3 main tasks, namely data collection, analysis in both real time and on request, and visualization. As in BeAware visualization is done on upper layer, for BeAware purposes data storage collects data and allows various view to it. All data is tagged with the data path it is associated with and timestamp, this allows the system to spread the data around the system and then retrieval based on above 2 keys.

The architecture itself is based on distributed software services with service announcement and discovery. All services are logical and will have multiple copies running. Each service instance will advertise itself in the cloud network with the information about available services and available data. Every service will listen to these advertisements and when it needs a service, it will contact the best match. A simple sequence might be:

1. Data request comes in via GUI
2. GUI looks for data visualizer
3. Data visualizer looks for on request data analyzer
4. Analyzer looks for a set of data storages containing the needed data

For cloud stability, services can also be restarted on the fly, blacklisted if they fail to provide service, etc.

2.3.1 Inbound data

Data arriving to the data storage is immediately copied to two separate streams, one storing the data and one doing immediate analysis on it.

Long-term storage mirrors the data to at least 2 physically separate stores and gives an acknowledged-message only when data has been stored to a physical device. Once saved into storage, data cannot be modified. (Even admin has only rights to remove data.)

Real time analysis keeps track of most recent data (usually last 5-15 minutes) for all channels that have been tagged for immediate analysis. This data is kept in memory for immediate access and as a new measurement arrives to any channel, that channel is automatically checked for any trigger conditions. If any of these conditions is met, an automatic alert is triggered via desired export API, usually email.

2.3.2 Outbound data

The outbound (aka northbound) data interface has two modes of operations, namely push and pull.

In push mode, the real time analysis (or any scheduled export, like monthly report) automatically sends data to user, when a trigger condition is met. This can be via email, sms or HTTP request to a predefined address, or just an alert indication in the GUI. Trigger condition can be either an upper or lower limit reached by the raw data, data missing, or a predefined result from a filter chain (see next chapter) applied to the data.

In the pull mode user requests data with desired filter chain applied to it. Outbound interface first authenticates the user and after that authorizes his data request checking that it does not contain entries user is not permitted to see. After that the filter chain is parameterized with user given parameters:

- What data is needed (channels)
- What period is needed
- What is the averaging time window, if needed
- Any additional parameters for functions

The filter chain produces a numeric result (usually a time series), which is then converted to the format user requested, either an image or excel and babup formatted time series.

2.3.3 Data filtering

One of the main problems in BeAware is the sheer amount of data. Even for a simple power channel, 1 minute produces 30 data points, so even theoretically iPhone resolution of 640x960 cannot be used to display 30 minutes of data, so some filtering is necessary. BeAware has several different uses to same data, all of which use some basic common analysis components, which can be reused. Also the project highlighted the fact that for quick prototyping of different GUI options, the underlying analysis engine must be configurable without a need to recompile and restart everything. Thus a system of chainable and user configurable filters was designed.

Chapter 5.3 shows an example of a configurable chain of filters, with sub chains coming from ready-made templates. There steady state detection, fingerprint creation, fingerprint clustering and chaining are chained together for detection of a multistate device.

Filters are of 3 main categories:

- Data reading fetches data from the long-term storage, handling things like splitting tasks to smaller subtasks, distributing subtasks etc.
- Data processing handles data preprocessing before actual analysis and visualization. This includes things like calculating averages, smoothing data with filtering, joining channels with mathematical operations etc.
- Data analysis includes the final stages of analysis, of something else that the basic time series is needed. This includes future estimates, state detection, fingerprint matching etc.

For this reason the Configuration sub message was added to main BabupMessage in BABUP description, so there would be an easy interface for parametrizing the filter chains.

2.3.4 Available services

- Data reception in cloud accepts data from the receiver agents, and sends it both to long-term storage and event&alert service for real time analysis. All data paths have their unique hash codes and are stored according to them to at laest 2 separate long term storage instances. Data receiver agent receives an acknowledgement that data is stored only when it actually has been stored to disk in long-term storage.
- Long-term storage handles storing and retrieving of data For security reasons, data is always offered as read only to prevent any tampering attempts.

- Event&alert analyzes the incoming data stream for any trigger conditions and performs an alert via the specified export service if any condition is met. All data is kept in memory for quick access and analyzer via same filter chains as in on request data filtering. Default value is 15 minutes worth of data in memory for each path.
- Data filtering applies requested filter chain to data. Whereas Event&alert works with in-memory data, Data filtering fetches data from long term storage and can thus handle much longer periods, with few year being a practical limit
- Export pushes the requested data to a 3rd party via desired protocol. Babup is one example of this, email/sms alerts of sensor data missing is another.

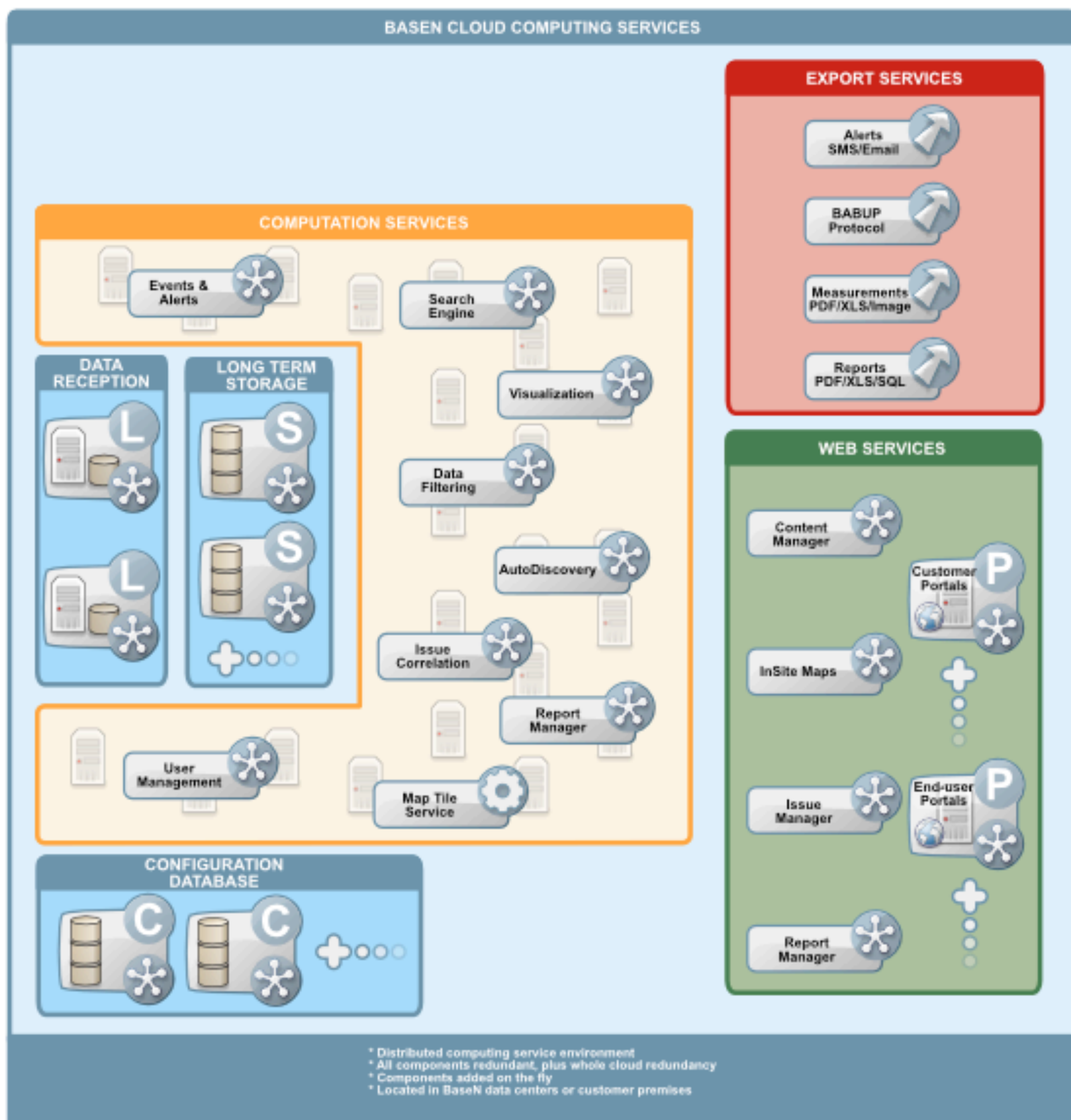


Figure 4 Data storage services

Data storage also offers several other services, which are not used in BeAware. These are mainly related to data visualization and geolocation.

3 Interfaces

3.1 Babup

Basic data transfer model used in BeAware Sensing Layer uses Babup (Beaware base station user protocol, Google Protocol buffers (GPB) based system) with normal HTTP as transport protocol. Both south- and northbound interfaces use the same protocol, while the northbound interface also offers additional java library that abstracts the HTTP(S) communication away.

Google protocol buffers offer a reasonably platform and language independent way of representing structural data. The main strength over XML is that GPB supports human readable form and also wireform, where all the data has been packed to a binary form, thus making transfer quicker and marshalling operations much lighter that with XML parsing.

The figure below shows the UML model of BABUP. Basically, due to protocol buffer internal workings, various message types are wrapped inside a BabupMessage. DataReply is the type of message base stations uses to send measurement data to data storage, while Service layer uses DeviceRequest, ChannelRequest and DataRequest to query the needed data.

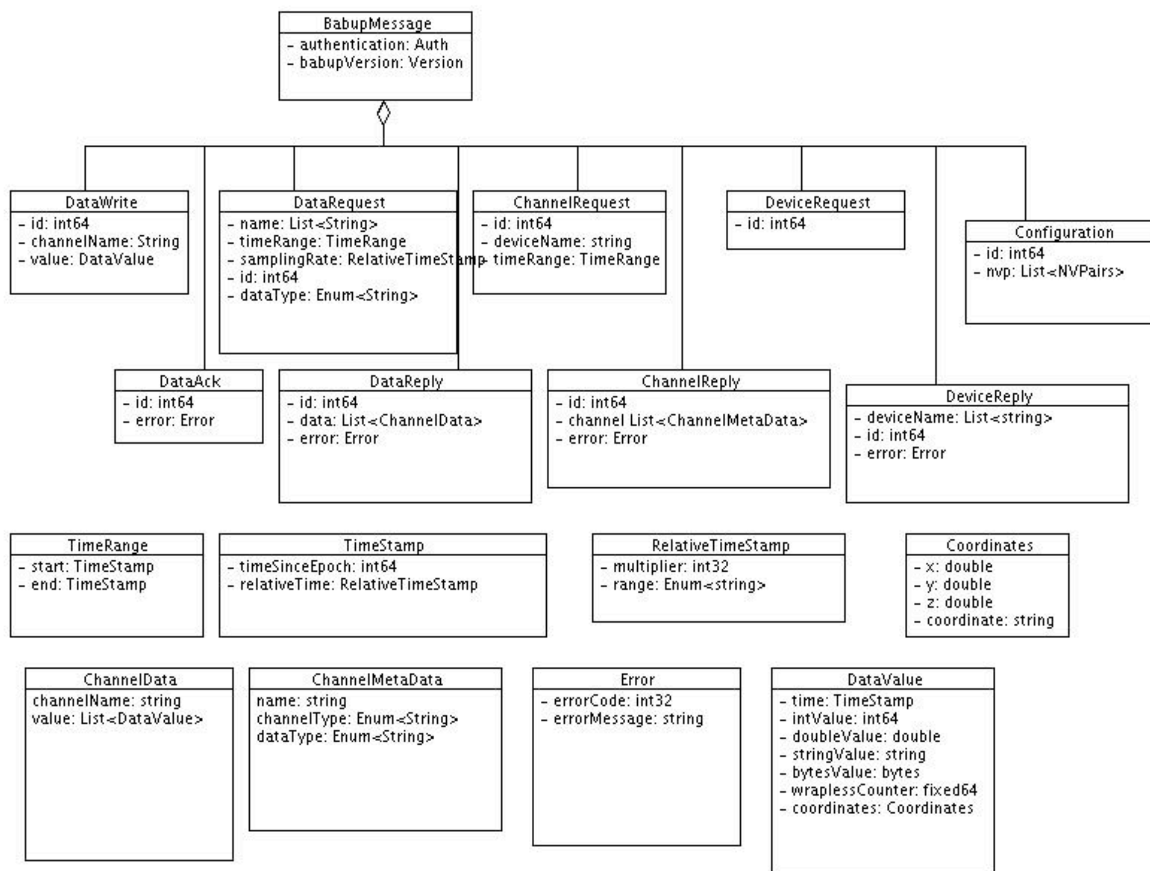


Figure 5 Babup sub messages

3.1.1 Using babup

As implementing a new sensor or base station that wants to interface with BeAware system via providing BABUP data is probably one of the most likely usage scenarios, details are covered here.

All communication between the client and the server is wrapped in the top-level *BabupMessage*:

```
message BabupMessage {
required Auth authentication = 1;
required Version babupVersion = 2;
optional DataWrite dataWrite = 1000;
optional DataRequest dataRequest = 1001;
optional ChannelReply channelReply = 1002;
optional ChannelRequest channelRequest = 1003;
optional DeviceReply deviceReply = 1004;
optional DeviceRequest deviceRequest = 1005;
optional DataReply dataReply = 2000;
optional DataAck dataAck = 3000;
}
```

The *authentication* and *babupVersion* fields need to be specified in each message.

Only one of the optional field types should be included in each BabupMessage. The most relevant types for regular client use are the *dataReply*, used to send measurement data from client to server, and the *dataAck*, which provides the server's response to this.

The *deviceRequest* is used by a client to request a listing of available 'devices', and the *deviceReply* is the server response. The *channelRequest* is used by a client to request a listing of channels within a device, and the *channelResponse* is the server response to this request.

The *dataWrite* and *dataRequest* are provisional types for server-to-client communication, and are not currently in use.

3.1.1.1 Auth

The *Auth* message is defined as follows:

```
message Auth {
optional string username = 1;
optional string credential = 2;
}
```

These fields are here specified for future usage where authentication cannot be handled in the transport protocol. For now, authentication **is** handled in the transport protocol.

3.1.1.2 Version

The *Version* message records the version of the protocol being used:

```
message Version {
optional fixed32 major = 1 [default = 1];
optional fixed32 minor = 2 [default = 0];
}
```

The current version number is 1.0.

3.1.1.3 DataReply

A DataReply carries information from the client to the server. The format of the *DataReply* message is:

```
message DataReply {
required fixed64 id = 1;
repeated ChannelData data = 2;
optional Error error = 3;
message ChannelData {
required string channelName = 1;
repeated DataValue value = 2;
}
}
```

The *id* field is reserved for future usage where the communication can be asynchronous and thus tracking what package has been processed is essential. For now, it can be set to any value.

Each *DataReply* can contain multiple *ChannelData* messages, and each *ChannelData* contains a *DataValue*, which can contain single or grouped measurements for a channel.

3.1.1.4 DataValue

The *DataValue* contains time, location, and measurements, either as a single entity (one time stamp and optional location, one measurement) or as a grouped measurement (one time stamp and optional location, with multiple measured values for this time/location):

```
message DataValue {
required TimeStamp time = 1;
optional sfixed64 intValue =2;
optional double doubleValue =3;
optional string stringValue =4;
optional bytes bytesValue =5;
optional fixed64 wraplessCounter = 6;
optional Coordinates coordinates = 7;
repeated GroupedData groupedData = 8;
}
```

The *time* and optional *coordinates* specify a data point; the value is defined by one of *intValue, doubleValue, stringValue, bytesValue*, or *wraplessCounter*. If multiple measurements are defined at the same time and location, the *groupedData* message should be used, instead.

Table 1 babup value types

Field	Content
intValue	Regular 64-bit integer value
doubleValue	Regular double value
stringValue	UTF-8 encoded string of arbitrary length
byteValue	Byte-encoded binary data of arbitrary length
wraplessCounter	Unsigned 64-bit integer value, used for pulse counting etc

3.1.1.5 Time

The *time* field records time stamp of the measurement. The format is:

```
message TimeStamp {
optional fixed64 timeSinceEpoch = 1;
optional RelativeTimeStamp relativeTime = 2;
}
```

One of the two fields must be specified. The field *timeSinceEpoch* records milliseconds since the Unix epoch (1.1.1970). If the client does not have an accurate clock, it can

choose one of the various other ways of recording time provided by the *RelativeTimeStamp* message:

```
message RelativeTimeStamp {
optional fixed32 multiplier = 1;
enum Range {
NOW = 1;
CURRENT = 2;
MINUTE = 3;
HOUR = 4;
MILLISECOND = 5;
SECOND = 6;
FOREVER = 7;
}
required Range range = 2;
}
```

The meanings of the values the *range* field can assume are explained in Table 1. For example, if one wants to communicate to the platform that the measurement was made three minutes ago, one sets the value of the *range* field to *MINUTE* and the value of the *multiplier* field to 3.

Table 2 Babup time values

Range keyword	Implied time
NOW	Receiving end will supply the timestamp when received.
CURRENT	Not applicable
MINUTE	*multiplier* minutes ago
HOUR	*multiplier* hours ago
MILLISECOND	*multiplier* ms ago
SECOND	*multiplier* seconds ago
FOREVER	Not applicable

3.1.1.6 Coordinates

A coordinates entry specifies the location of a measurement. The format is:

```
message Coordinates {
required double x = 1;
required double y = 2;
optional double z = 3;
required string coordinate = 4;
}
```

The *x* and *y* fields indicate the longitude and latitude respectively; the units depend on the selected coordinate system, as specified by the *coordinate* field. The currently supported coordinate value is "*gps*"; an updated list of supported systems will be provided in a new version of this document.

The optional *z* field indicates the height from sea level, in meters.

3.1.1.7 GroupedData

It is a common occurrence for multiple measures to be recorded for a specific time-location pair. The most effective method to encode and store this kind of related information is to use the *GroupedData* option, as opposed to specifying multiple *DataValue* messages. The format of *GroupedData* is:

```
message GroupedData {
required string subChannelName = 1;
optional sfixed64 intValue = 2;
optional double doubleValue = 3;
optional string stringValue = 4;
optional fixed64 wraplessCounter = 6;
optional Coordinates coordinates = 7;
}
```

Here the fields *intValue*, *doubleValue*, *stringValue*, and *wraplessCounter* are of the same format as in the *DataValue* message. The *coordinates* field is supplied for the case where a location is actually a measurement (e.g. when sending measurements from a radar or other position reading device). The *subChannelName* is the name of the particular quantity being measured, and it must always be specified.

Note that the *byteValue* field is not supported in *GroupedData*. This is because byte-encoded binary data is typically large, and requires special handling on the receiving end; the optimization performed with grouping would bring no benefit.

3.1.1.8 DataAck

When storage has received a babup message, it will reply with a DataAck.

The format of the *DataAck* message is:

```
message DataAck {
required fixed64 id = 1;
optional Error error = 2;
}
```

The platform sets the *id* to the same number that the client used in the *DataReply*. If errors occurred during handling the received message, the *error* field of the response is set. It has the format:

```
message Error {
required fixed32 errorCode = 1;
optional string errorMessage = 2;
```

```
}
```

The currently used error codes are listed in Table 2.

Table 3 Babup error codes

Error code	Explanation
1000	Received a null message
1001	Received a message that could not be parsed
1002	Received a message that could not be parsed
1003	Received a message that was not known to BABUP
1004	Unknown error

3.1.1.9 Transport

The transport protocol used to communicate the messages is standard HTTP, using the POST method.

The client is authenticated with HTTP basic authentication. The BABUP messages may be transmitted either as binary or as text messages, and the HTTP headers must be set accordingly:

Table 4 Babup HTTP headers

Content type	HTTP header	Header value
Binary-serialized Babup message	`Content-type`	`application/octet-stream`
	`X-Babup-Text`	`FALSE`
Text-serialized Babup message	`Content-type`	`text/plain`
	`X-Babup-Text`	`TRUE`

The payload of the query is a *BabupMessage*, which has its *dataReply* field set. In the payload of the platform's answer is another *BabupMessage* that has its *dataAck* field set. The format of the reply message is the same as that of the query message – if the original message contained binary-serialized data, the response is a binary message, as well.

3.1.1.10 Examples

The following Python snippet will generate a *BabupMessage* that contains measurements from two channels, *power* and *temperature.* In the power channel we have five integer values measured at consecutive seconds. In the temperature channel

we have one double value for which we are happy to accept the timestamp provided by the server.

```
msg = BabupMessage()

msg.authentication.username = "user"
msg.babupVersion.major = 1
msg.babupVersion.minor = 0

msg.dataReply.id = 1

data = msg.dataReply.data.add()
data.channelName = "power"
for i in range(5):
value = data.value.add()
value.time.timeSinceEpoch = 1225887748940 + i*1000
value.intValue = i

data = msg.dataReply.data.add()
data.channelName = "temperature"
value = data.value.add()
value.time.relativeTime.range = RelativeTimeStamp.NOW
value.doubleValue = 123456.789
```

On platforms where none of the Google protocol buffer implementations can be used, one might adopt the following scheme for generating messages. Since all data types being used in the definitions are of

fixed length, one can generate on one's desktop computer a template message, and then employ that on the client device only filling in those portions that are variable. Below, there are two hex dumps of

messages generated by the above snippet. In the second message the power values run from 0 to -4 contrary to the first message where they run from 0 to 4.


```
00000000 0a 06 0a 04 75 73 65 72 12 0a 0d 01 00 00 00 15 |....user........|
00000010 00 00 00 00 82 7d a0 01 09 01 00 00 00 00 00 00 |.....}?.........|
00000020 00 12 75 0a 05 70 6f 77 65 72 12 14 0a 09 09 4c |..u..power.....L|
00000030 93 9a 6c 1d 01 00 00 11 00 00 00 00 00 00 00 00 |..l.............|
00000040 12 14 0a 09 09 4d 93 9a 6c 1d 01 00 00 11 01 00 |.....M..l.......|
00000050 00 00 00 00 00 00 12 14 0a 09 09 4e 93 9a 6c 1d |...........N..l.|
00000060 01 00 00 11 02 00 00 00 00 00 00 00 12 14 0a 09 |................|
00000070 09 4f 93 9a 6c 1d 01 00 00 11 03 00 00 00 00 00 |.O..l...........|
00000080 00 00 12 14 0a 09 09 50 93 9a 6c 1d 01 00 00 11 |.......P..l.....|
00000090 04 00 00 00 00 00 00 00 12 1e 0a 0b 74 65 6d 70 |............temp|
000000a0 65 72 61 74 75 72 65 12 0f 0a 04 12 02 10 01 19 |erature.........|
000000b0 c9 76 be 9f 0c 24 fe 40 |?v?..$?@|

00000000 0a 06 0a 04 75 73 65 72 12 0a 0d 01 00 00 00 15 |....user........|
00000010 00 00 00 00 82 7d a0 01 09 01 00 00 00 00 00 00 |.....}?.........|
00000020 00 12 75 0a 05 70 6f 77 65 72 12 14 0a 09 09 4c |..u..power.....L|
00000030 93 9a 6c 1d 01 00 00 11 00 00 00 00 00 00 00 00 |..l.............|
00000040 12 14 0a 09 09 4d 93 9a 6c 1d 01 00 00 11 ff ff |.....M..l.....??|
00000050 ff ff ff ff ff ff 12 14 0a 09 09 4e 93 9a 6c 1d |??????.....N..l.|
00000060 01 00 00 11 fe ff ff ff ff ff ff ff 12 14 0a 09 |....????????....|
```

```
00000070 09 4f 93 9a 6c 1d 01 00 00 11 fd ff ff ff ff ff |.O..l.....??????|
00000080 ff ff 12 14 0a 09 09 50 93 9a 6c 1d 01 00 00 11 |??.....P..l.....|
00000090 fc ff ff ff ff ff ff ff 12 1e 0a 0b 74 65 6d 70 |????????....temp|
000000a0 65 72 61 74 75 72 65 12 0f 0a 04 12 02 10 01 19 |erature.........|
000000b0 c9 76 be 9f 0c 24 fe 40 |?v?..$?@|
```


The alternative to the binary format is the text format. The following listing is the text format encoding of the message described above:


```
authentication {
username: "user"
}
babupVersion {
major: 1
minor: 0
}
dataReply {
id: 1
data {
channelName: "power"
value {
time {
timeSinceEpoch: 1225887748940
}
intValue: 0
}
value {
time {
timeSinceEpoch: 1225887749940
}
intValue: 1
}
value {
time {
timeSinceEpoch: 1225887750940
}
intValue: 2
}
value {
time {
timeSinceEpoch: 1225887751940
}
intValue: 3
}
value {
time {
timeSinceEpoch: 1225887752940
}
intValue: 4
}
}
data {
channelName: "temperature"
value {
time {
relativeTime {
range: NOW
}
```

```
}
doubleValue: 123456.789
}
}
}
```

This encoding is quite inefficient, both in handling and in transport, but it is easy to produce and parse. The text encoding may be the only option in some cases, e.g. if implementing a browser-based Javascript client.

The following example illustrates the message format in the case of an agent collecting several measurement quantities associated to a time and location, called grouped measurements. We collect a single sample of the quantities *temperature* and *power.*

Listing 1 is a Python example of the process; Listing 2 performs the same message building in Java. Listing 3 is the resulting message in text format.

Note: both the Python and Java code require understanding of the language in question, and of the Google Protocol Buffers generated code for that language. Neither listing accounts for library or module dependencies and imports.

```
msg = BabupMessage()

msg.authentication.username = "user"
msg.babupVersion.major = 1
msg.babupVersion.minor = 0

msg.dataReply.id = 1

data = msg.dataReply.data.add()

data.channelName = "ExampleDevice"
value = data.value.add()
value.time.timeSinceEpoch = 1225887748940
value.coordinates.x = 24.946604
value.coordinates.y = 60.167497
value.coordinates.z = 0.0
value.coordinates.coordinate = “gps”
group = value.groupedData.add()
group.subChannelName = “power”
group.doubleValue = 1.23
group = value.groupedData.add()
group.subChannelName = “temperature”
group.doubleValue = 42.0
```

Listing 1: Python code for building a simple grouped message

```
BabupMessage.Builder msg = BabupHelper.createEmptyBabupMessage( "user", 1,
0 );
DataReply.Builder reply = DataReply.newBuilder();
reply.setId( 1 );

TimeStamp.Builder time = TimeStamp.newBuilder();
time.setTimeSinceEpoch( 1255589326436L );

DataValue.Builder data = DataValue.newBuilder();

ChannelData.Builder cd = ChannelData.newBuilder();
cd.setChannelName( "ExampleDevice" );


Coordinates.Builder c = Coordinates.newBuilder();
c.setX( 24.946604 );
c.setY( 60.167497 );
c.setZ( 0.0 );
c.setCoordinate( "gps" );
data.setCoordinates( c );
data.setTime( time );
GroupedData.Builder gdb = GroupedData.newBuilder();
gdb.setSubChannelName( "power" );
gdb.setDoubleValue( 1.23 );
data.addGroupedData( gdb );
gdb = GroupedData.newBuilder();
gdb.setSubChannelName( "temperature" );
gdb.setDoubleValue( 42.0 );
data.addGroupedData( gdb );

cd.addValue( data );
reply.addData( cd );
msg.setDataReply( reply );
```

Listing 2: Java code for building a simple grouped message

```
authentication {
username: "user"
}
babupVersion {
major: 1
minor: 0
}
dataReply {
id: 1
data {
channelName: "ExampleDevice"
value {
time {
timeSinceEpoch: 1225887748940
}
coordinates {
x: 24.946604
y: 60.167497
z: 0.0
coordinate: "gps"
}
groupedData {
subChannelName: "power"
doubleValue: 1.23
}
groupedData {
subChannelName: "temperature"
doubleValue: 42.0
}
}
}
}
```

Listing 3: the produced message

3.2 Southbound data receiver interface

Southbound interface towards base station is purely for data receiver via Babup, but it also can receive data from other sources. The BABUP interface takes care of authorizing the incoming request, processing the data for storage and it guarantees that the base station only gets ACK message when data actually has been stored.

Data receiver interface only accepts DataReply messages, as the base station is not authorized to read any data from this interface, nor it is authorized to write any control data. (If base station needs to read data, it can access the northbound interface, which supports both read and write.)

The Figure 6 shows the activity when a babup message is received. First AuthenticationFilter inside the receiver servlet authenticates the request, then a BabupFilter processes the payload and finally the DataStorage creates a DataAck message, which is sent as part of the servlet HTTP response.

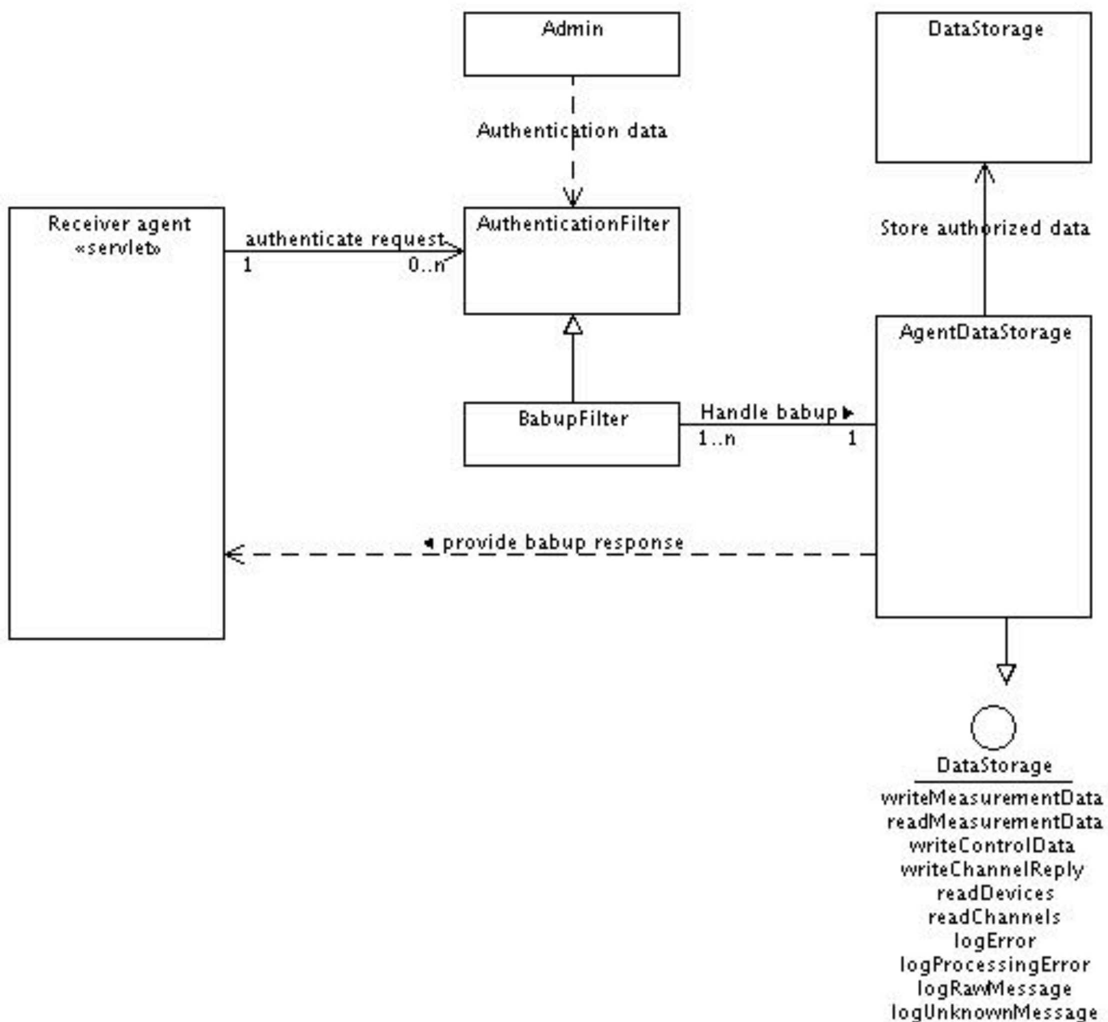


Figure 6 Data receiver interface

3.3 Northbound client interface

The Data Storage provides a similar BABUP based servlet access point for higher layers. In practice, this is hidden it the Service Layer's library BAProtocols.jar, which contains the following functionality:

- BabupClient abstract the Babup over HTTP communication to java API
- BeawareBabupClient extends the client with BeAware specific functionality

This code is available in the BeAware repository, so user is advised to read the documentation therein for workings of the java part.

The underlying babup receiver is similar to above but as this interface provides also read and control access, in addition to authenticating the request, system also

authorizes it by checking that user is allowed to perform the desired operation and is allowed to access the requested data.

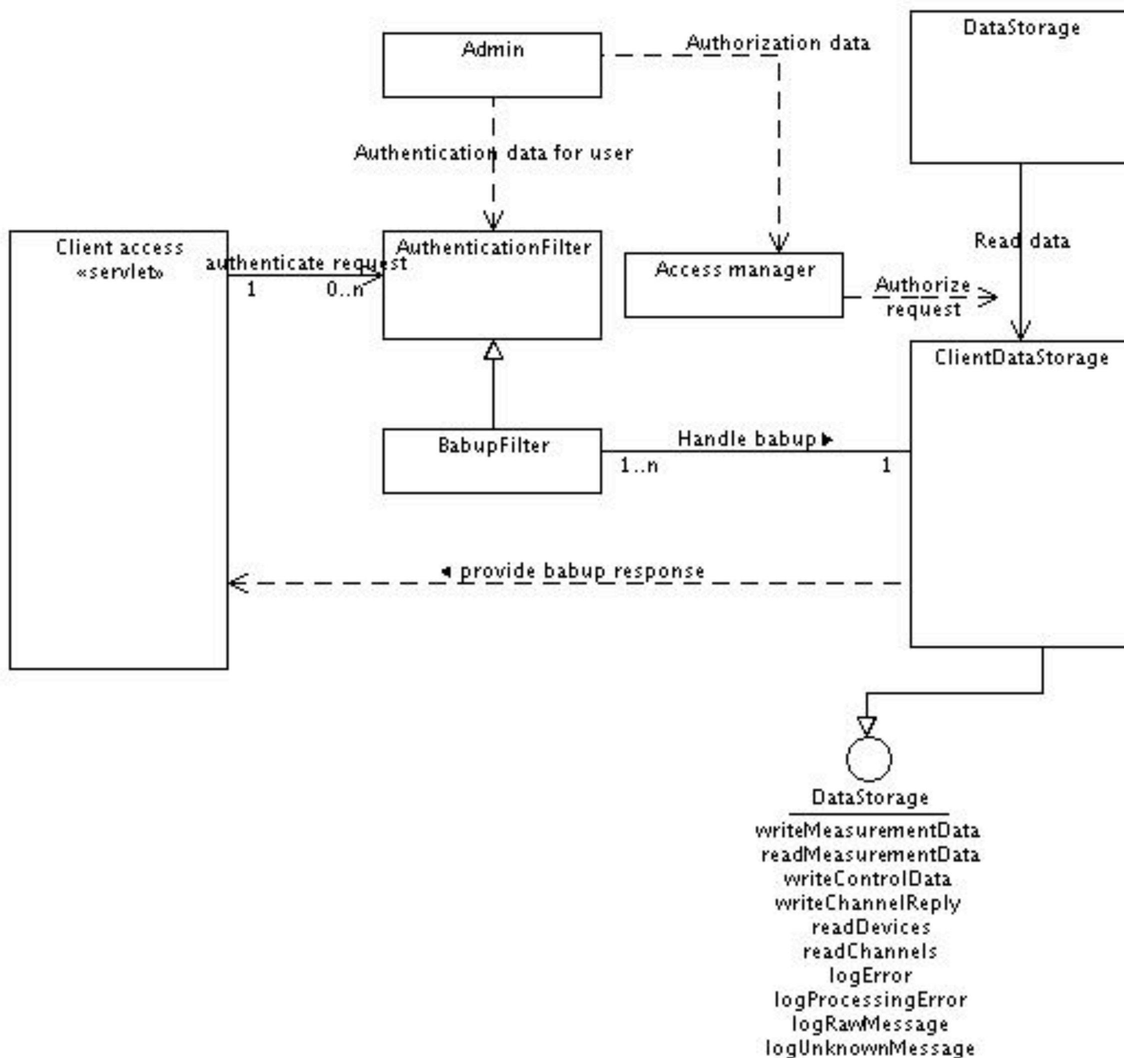


Figure 7 Client Interface

In general, unless the available data is known, northbound interface is accessed in 3 steps, as shown in Figure 8. First user checks what available devices (base stations) he has via DeviceRequest, then he can check what channels (name, type) are available each device via ChannelRequest and finally he can request data via DataRequest. Usually Devices and Channels do not change much, so those values are prime candidates for caching in upper layers.

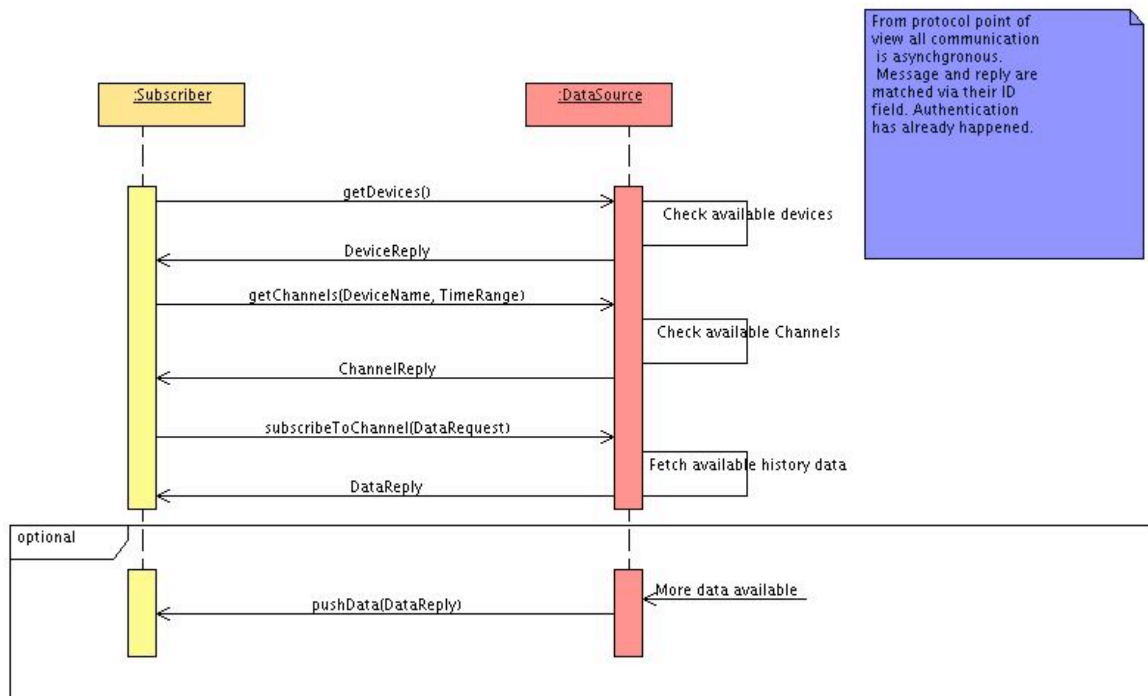


Figure 8 Data retrieval process

3.4 Other supported protocols

3.4.1 Base station

Base station capabilities have been explained in further detail in D3.7, so this chapter presents just a brief overview of possibilities.

- BASP is the BeAware Sensor Protocol, a byte level protocol for reading sensor data and transmitting AI specific instructions. As with any lightweight embedded system, resources in sensor are very limited with harmonics analysis taking most of the available CPU time, so protocol has been tailored to fit the sensor requirements.
- Pulse outputs - base station is able to act as a pulse counter via serial port.
- SNMP is supported in both base station and data receiver, as it is very widely supported protocol. As an example, Moxa R202 pulse concentrator has a reader support in base station software
- Plugwise sensors we used as an alternative to BeAware sensors, and thus Plugwise protocol 1.0 is supported. Protocol did chance in version 2.0 and support for that has not been added.

3.4.2 Data receiver

Unlike base station, data receiver has been written is Java and is part of the BaseN platform proper. Java has a very good support of various industrial protocols, so these are in turn supported by the data receiver and functionality has been also used in early BeAware prototypes, when various approaches to interface with meters was tested. Currently other supported protocols are

- SNMP: de facto industry standard for monitoring anything network related. SNMP has been tested very thoroughly, so combined with IPv6 would probably be the best choice for truly massive scale deployment of smart meters.
- KNX: one of the high-end, partially open home automation protocols. (See www.knx.org) Supports a variety of meters and provides excellent functionality, but unfortunately both devices and configuration software for the KNX network suffer from high cost. Can be interfaced via TCP/IP adapter, so very easy to work with.
- M-bus and ModBus are both older industrial standards for industry and home automation. They support a wide variety of devices and particularly m-bus is on the verge of becoming most probable standard for local bus meter reading in EU. (See www.modbus.org, www.m-bus.com)
- XML, CSV and binary files over SMTP, FTP or HTTP

4 Analysis capabilities

The underlying engine was previously tested with minute resolution data, and as expected the BeAware data (second resolution) pinpointed some bottlenecks, most obvious of these being efficiently handling the raw data analysis for GUI baselines calculations, which need several months worth of data.

Apart of the very obvious analysis functions like period averages, median smoothing, etc., all these methods are also available as part of the BeAware open source codebase.

4.1 Underlying system

Even as the system itself does both real time analysis to arriving data and on request analysis to stored data, the system used for both is the same, only the data source beneath is changed between in-memory and long-term storage. All arriving data is kept in memory from 1 minute to 15 minutes, depending on per-channel configuration, with default being 15 minutes.

The calculations, as explained in 2.3.3 is based on chains of configurable filters, each performing a subtask. A collection of filters can be either configured in the request by user, or from ready-made templates.

4.2 Usage calculations

The usage calculations are performed on raw data from the Power channel. BeAware sensor does have also cumulative energy available, but as Plugwise POL protocol only provides power and sensors are mixed transparently, cumulative energy is calculated from integral of power for BeAware sensor too. For beaware purposes, the current power display uses time period of last 5 minutes of data with the average of last minute shown to user. (5 minutes, as the inherent network delays and data being sent at 30s intervals sometimes mean that the last minute is not available, so minute before that can be used.)

Error from integrating over power for a period of day is less than 1‰ when integration result was compared against energy measurement, when available.

Method to use BabupClient::getChannelData

4.3 Baseline

Baseline (how much is the comparison usage against which the current usage is reflected) is one of the algorithms that changed a lot during the project. Initially used 3 previous months average meant that particularly in Nordic climes with direct electrical heating, it was completely impossible to save on the household level when weather was getting colder and similarly almost impossible not to save, when weather against got warmer, as baseline would always lag behind. So currently the system gives 3 options for baseline

- Previous period against current period (currently previous full week's average against last days average)
- This calendar month's average consumption from last year, if available. Otherwise this month's average consumption this far.
- Predefined constant for this month. (Usually from bill.)

For a more complete system, baseline should have a thermal correction available, but due to resource constraints that was skipped to allow for more research in fingerprinting.

Methods to use BeawareBabupClient::getBaselineWeek for first option with weeks default, BeawareBabupClient::getBaseline for other options with parametrizable time period.

4.4 Standby detection

As reducing device standby consumption is one of the easiest places to save energy, detecting such state was added as an own functionality. Standby is currently defined as lowest detectable steady state that is at least 30s long, separates valid on states, and consumption is between 0.1W to 10W. This unfortunately leaves out large TV sets and such where even standby consumption can be in range of tens to hundred watts, but

raising the limit much above 10W would mean that many multistate devices would be incorrectly flagged as on standby.

A better solution, now at least partially possible is via fingerprinting. Fingerprinting can detect both the device type and then the different operational modes to some degree, so standby detection could be done via separate fingerprints.

Method to use is BeAwareBabupClient::getStandbyLevel for estimated standby level and BeAwareBabupClient::getEvents

4.5 Usage cycles

For device with cyclic characteristics (washing machine, refrigerators compressor etc) the system provides the possibility to count the number of cycles (defined as power over 0.1W) within a given period, plus the length and energy used for each cycle. Even this simple cycle detection algorithm can give us information how user's habits regarding the device are changing and also warning if the device behavior is changing (for example if energy used changes, something is probably broken, unless it is a multistate device. Of if the frequency of cycles gets higher with freezer, it probably needs defrosting.)

Method to use is BeawareBabupClient::getEvents

4.6 Alerting user

The real time analysis system has the capability of pushing alert ("Event X has happened") to users. In BeAware this functionality was tested in two separate cases:

Admin users monitored each test site for availability, both on the base station level and on sensor level. This gave us ability to immediately see how the system was working and if some household should be contacted.

As an example of an early ultra smart advice, fridge door opening/closing was monitored via detecting the changes in steady states matching the internal lamp consumption and alerting the user via sms if door remained open over 5 minutes. This functionality was only piloted in Finnish internal trial site as a standalone feature.

In the future, alerting system could be connected to a service layer servlet, which would generate a message to user every time a trigger condition is met - this would allow system to push advice user as events happen, instead of periodical batch checks.

Alerts are configured via BaseN internal system, as they were not connected to EnergyLife. In production version of EnergyLife, user should be able to do configuration via EnergyLife app itself.

4.7 Advice

Engaging users via tailored advice based on their individual consumption habits was one of the main points of BeAware. This functionality is a shared effort between the Service and Sensing layer. BeAware support 3 levels of advice:

- *Normal advice* is based on informing users of expected energy usage of appliances and how to lower it, and informing users of common bad and good habits. No measurement data is needed.
- *Smart advice* depends on minute level measurements and advises people on their normal usage patterns: how long a device was used per week, how long it was on standby, how much power it consumed, etc.
- *Ultra-smart advice* requires more frequent measurements (1 Hz range) and power quality information. The data is used to advise users on device specific issues such as suitable power levels, or on usage anomalies, e.g. an unclosed refrigerator door. This class of advice can also import other information, such as outside temperature and indoors humidity to enable the system to produce better information of HVAC and heating related systems.

Normal and Smart advice are part of the current EnergyLife applications, whereas Ultra-smart advice has been demonstrated in lab and internal trial tests, as they require the fingerprinting functionality.

5 Load fingerprinting

5.1 Overview

One of the advantages of the BeAware system is its' ability to produce highly detailed data at high granularity, and to store it all in non-processed format, allowing for later in-depth analysis. One of the tasks of the data storage was to apply analysis on the power quality data, to check if device type and state can be detected from this collected data, either in real time or from history. The term ‘appliance fingerprinting’ was conceived to express this solution due to the fact that it needs to make use of electrical characteristics (or ‘load signatures’) that each appliance uniquely possesses.

The detailed information collected regarding the electrical attributes of various residential loads, such as harmonics contents, is presumably a key input in devising ways to tackle power quality issues, which are on a steady rise due to the increasing use of power electronic devices and other non-linear loads in industrial, commercial and domestic applications. In addition to its relevance in energy consumption tracking and power quality control, appliance fingerprinting also helps to realize the following scenarios:

- Utilities can improve planning and operation; for example, one way is to re-schedule larger loads by offering time-dependent rates to consumers so that they will be encouraged to utilize high-energy appliances at low-rate times.

- Equipment manufacturers can improve quality and compliance, thus providing more energy efficient products to the market.
- Aging and abnormally operating appliances that consume higher amounts of energy can be spotted and remedial action can be taken.
- Monitoring of individuals or systems with specific needs based on their detailed electricity usage patterns; for example, seniors living alone or remotely operated mission-critical equipments.
- Switching off non-essential loads such as air conditioners in case of emergencies if the power system is in danger of collapse.

In BeAware, the fingerprint analysis is based on steady state harmonics analysis, where the harmonic frequencies of the input current are combined with the fundamental frequency. This allows for better identification of power electronic loads. For more details, see D3.8.

Due to variations in design philosophy among manufacturers producing the same type of appliance, the impact of power factor correction and harmonic mitigation techniques, and specific operating mode of the appliance at a given instant of time, fingerprinting of exact device type seems not to be feasible.

Based on the findings of the measurement work, it is possible to categorize the majority of household appliances into the following three main classes.

5.1.1 Resistive load

This category encompasses appliances that are mainly used for heating and lighting purposes such as panel heaters and incandescent lamps. Heating elements of other appliances like washing machines and dishwashers also belong to this category.

Their reactive power consumption is comparatively quite small, which means they regularly operate close to unity power factor (both FPF and total PF). Their harmonic content is usually negligible and in general THDI does not exceed 5%. Results of the measurement work also show that the crest factor of appliances in this group normally falls in the range 1.38 to 1.44, which is close to the crest factor value of a pure sinusoidal current.

5.1.2 Power electronic load

Majority of modern electronic equipments use switched-mode power supplies (SMPS). These differ from older units in that the traditional combination of a step-down transformer and rectifier is replaced by direct-controlled rectification. The benefits are improved load efficiency, better controllability, and reduced size, weight and cost. However, the undesirable effect is an increase in the propagation of harmonic currents back to the utility grid, with amplitudes at times exceeding that of the fundamental frequency current.

Appliances such as computers, television sets and compact fluorescent lamps belong to this category. As can be seen in Figure 3.2, these appliances contain significant levels of harmonic distortion. For instance, THDI values of 230% (laptops) and 175% (next generation LEDs) were recorded during the measurement work.

Due to the high level of harmonic content and hence the existence of distortion power, total PF is notably smaller than FPF.

5.1.3 Motive (inductive) load

This class consists of motor-driven, pump-operated and other inductive loads such as refrigerators, microwave ovens and fluorescent lamps (without PFC). The operation of these loads results in the production of substantial reactive power and it also causes harmonic distortion but at a lesser level as compared to electronic appliances.

Due to significant reactive power consumption, such loads do not operate close to unity power factor. However, the addition of a PFC circuit, as observed in the case of fluorescent lamps, improves the condition. In this group of loads, the 3rd harmonic current is dominant over the other harmonic orders.

5.2 Fingerprints and load library

The fingerprint used consists of the following variables:

- Active power (P)
- Fundamental power factor (FPF)
- Total power factor (TPF)
- Total harmonic distortion (THDI)
- Crest factor (CF)
- 3rd harmonic current
- 5th harmonic current
- 7th harmonic current

These variables are explained in more detail in D3.8. All values are the average of variable over the whole steady state event. Device can have several such fingerprints attached, if it is multimodal.

Load library will be available as part of the BeAware website later. A whole load library entry consists of

- Device name
- Unique id (assigned by BeAware team, basically a counter)

- Fingerprint data, with at least one decimal accuracy, preferably more
- Type of load (electronic|resistive|motor)
- Usage category

5.3 Fingerprinting process

The fingerprinting process is shown in Fig 9.

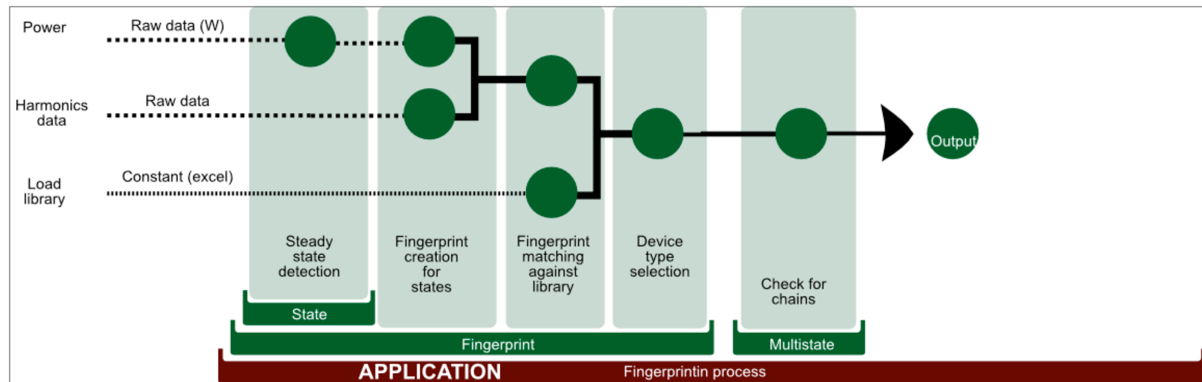


Figure 9 Fingerprinting process

- The steady states from the Power channel are detected.
- Other power quality values for each steady state period are read and fingerprint is constructed
- Fingerprint is matched against library
- If from same sensor system gets multiple devices, those are matched against known multistate device chains to check if this is a know multistate device

5.4 Steady state detection

The steady state algorithm is explained in[1], but basically follows the following:

- Filter the samples with a median filter if necessary
- Filter unsteady states, where the change to last measurement is more than the sample buffer standard deviation. If this happens, state has been exited.

---

[1] Algorithms for Event Detection and Fingerprinting of Electrical Appliances,

Arto Meriläinen

Helsinki Institute for Information Technology

Available at project website

- Check for state transition. If value is outside the confidence interval calculated for the sample buffer, a state has been exited.
- If system is currently in unstable state, but change was inside standard deviation and confidence interval and sample buffer has more samples that state minimum length, enter a new stable state.

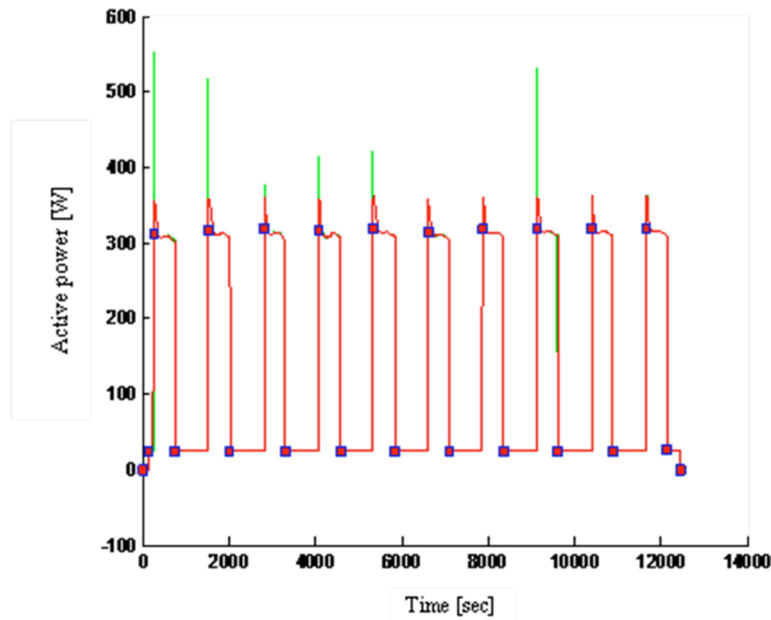


Figure 10 Steady state detection

The figure above shown an example of the algorithm applied to a refrigerator. Green line is the unfiltered measurement series, red is the media filtered and violet squares are the detected state transition points. The algorithm performs very well, but it has the noted shortcoming of not being able to detect loads, value of which falls in to the standard deviation of previous loads. So in effect a couple of large loads (heaters, sauna, oven) will cause a situation where the system is not able to detect very small loads, like phone chargers. A more adaptive system would be needed for this, warranting more research.

5.5 Main type detection

As described in D3.8, the harmonics based fingerprinting system is able to detect 3 main classes of loads:

- Resistive
- Power electronic
- Motor inductive

Unfortunately the variation inside the groups is such, that at least with limited size load library, exact subtype of device (vacuum cleaner inside motor inductive group, or

computer inside power electronic) cannot be detected with high enough certainty that it could be used.

The detection system is based on K-nearest neighbor clustering, where the distance between fingerprints x and y is given by the function, where j is the number of attributes in fingerprint (see D3.8) and σ is the standard deviation for attribute j.

$$d(x,y) = \sqrt{\sum_{j=1}^{n} \left(\frac{1}{\sigma_j^2}\right)(x_j - y_j)^2}$$

and the final class label by unweighted voting by the K nearest fingerprints.

5.6 Device subtype

Unfortunately the current fingerprinting process, while much more accurate than one just based on active power, still has problems coping with exact device types within the main type. This is mainly due to few facts:

- Time and usage slowly alter the device fingerprint, so a brand new and 5 years old device might produce very different fingerprints, even is maker and mark are the same.
- Slight variations in operation conditions (dish washers, washing machine loads, software running on computer) do produce slightly different fingerprints
- Same type of device from different manufacturers use different parts, internal circuits etc, so their fingerprints are different.

So the approach chosen produced roughly 60% hit rate on known devices and for completely new devices just gives the main type correctly. So further research on this subject is needed - using transitional signatures with higher resolution sensor, taking on-cycle time into account etc.

5.7 Multistate devices

Many household devices do not have a single operation mode, but produce varying results - fan with a heating option might show as motor or resistive, whereas a dishwasher will alternate between resistive heating, motor etc. To detect these signatures and to get some estimate on what the exact operation mode is, a Markov Chain based approach was chosen.

Markov chains is a term for a system with a set of states and transition probabilities between states, where the transition from state to next is not dependant on the history of states. This allows the system to calculate what probability any given list of transitions has for each specified markov chain.

As the state analysis is based on steady states, it was necessary to add two new kinds of states to system. Undefined for periods between valid states where power is consumed, but state is not stable. And Switchoff for ending a chain when power consumption goes to 0.

Figure 11 shows an example of dishwasher on 65C degree program, with Figure 12 and Figure 14 showing the associated power quality and detected steady states respectively.

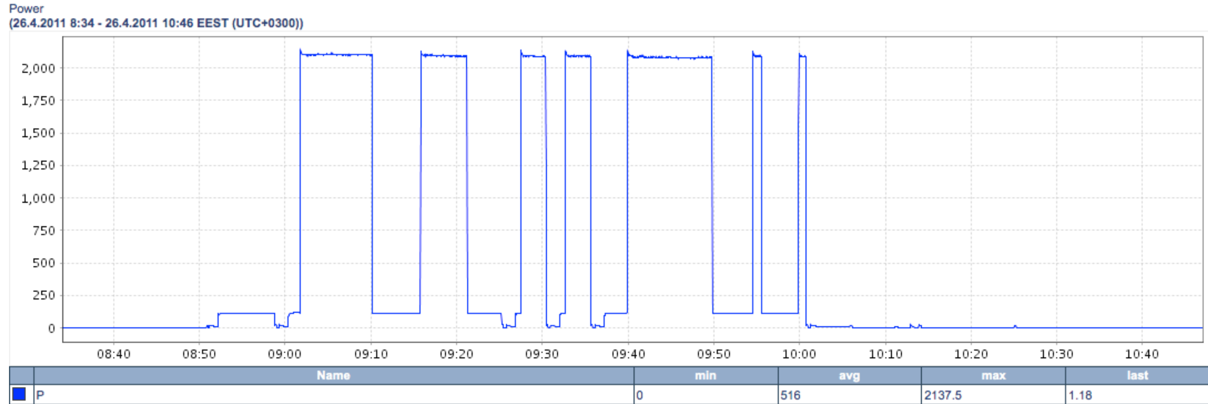


Figure 11 Raw power measurements

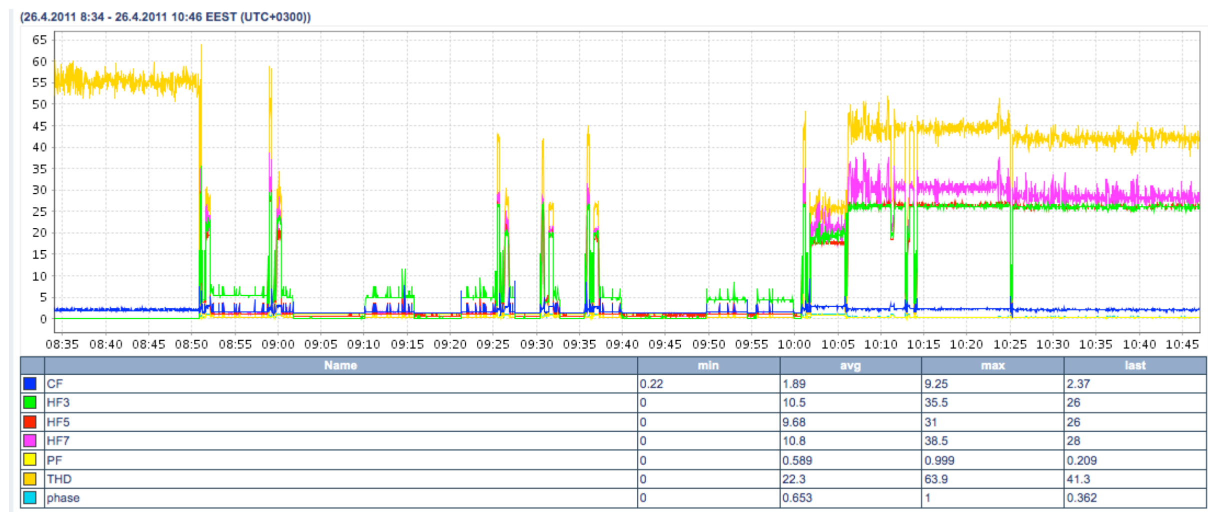


Figure 12 Power quality measurements

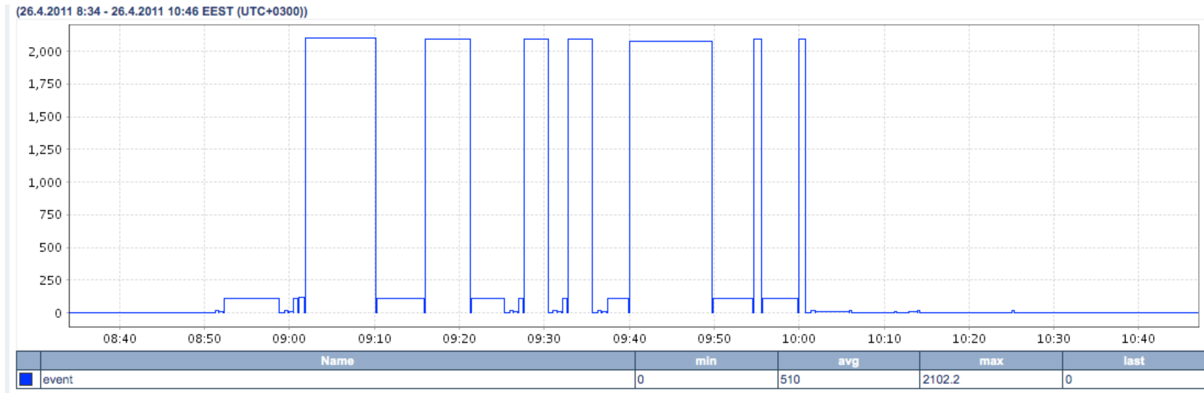


Figure 13 Detected steady states

These states are then analyzed via main type clustering to following:

25.4.2011 12:28 - 26.4.2011 12:28

26.4.2011 9.26.27 EEST	result	electronic
26.4.2011 9.25.57 EEST	result	electronic
26.4.2011 9.21.32 EEST	result	motor
26.4.2011 9.16.02 EEST	result	resistive
26.4.2011 9.10.22 EEST	result	motor
26.4.2011 9.02.02 EEST	result	resistive
26.4.2011 9.01.12 EEST	result	motor
26.4.2011 9.00.37 EEST	result	motor
26.4.2011 9.00.02 EEST	result	electronic
26.4.2011 8.59.32 EEST	result	motor
26.4.2011 8.52.27 EEST	result	motor
26.4.2011 8.51.52 EEST	result	electronic
26.4.2011 8.51.02 EEST	result	electronic

Returned 39 rows.

prev next 24 hours Channels: ^result$ Data: Filter log

Figure 14 Detected sub events

So the chain of transitions is electronic-electronic-motor-motor-electronic..

From a test set of these a probability of each set of transitions can be calculated. In the initial version where each separate state was considered, it very quickly became obvious that number of transitions in normal device during full operation cycle was usually so large that detection started to vary too much on random variables; whether start of device for example had enough noise to cause 2 separate electronic event or not. Basically the probabilities of longer chains became so low as to be meaningless.

To account for this, first a system where adjacent state of same type was combined was tried. So the chain above would become electronic-motor-electronic-motor-resistive. This alone produced much better results, but number of transitions was such that particularly detecting different modes of same device was hard, as they had roughly the same transitions, just numbers differed.

So the final iteration of system different a bit from pure Markov chain, as a combination of basic states was considered as "super state". For example in tested dish machine,

water heating cycle looped over "motor-resistive-electronic", with eco mode having only one or two such, whereas 65C long mode having several. Thus for eco-mode chain model transition could read:

start: motor

transition: resistive-electronic

Probability: 50%

Whereas for 65C it could be

start: motor

transition: resistive-electronic-motor-resistive-electronic

Probability: 50%

Where in both cases the state actually described is "transition over the heating cycle". Also, with this system of "super states" it is much easier to detect unknown chains that do not match any known chain fingerprint.

The chain fingerprint library will be available as the project public deliverables along with load library. Format is

DeviceModeName;StartState1:transition-transition-..-end:probability;..;StartStateN:transition-end:probability

Where at least one transition should end in "switchoff". All state names are in lower case and options are:

- resistive
- electronic
- motor
- switchoff
- undefined

Due to small number of test devices the system for which project had such access that we could record several full operation runs for several modes, the system has not yet been excessively tested but at least preliminary it gives good results and tends to err to side of "Unknown" instead of false positive, as most errors are due to Unknown state, which does no match a super type template. As the data from trials is still under analysis when this is written, we cannot yet give an estimate how accurate chain fingerprints from machine are when applied to another device from same category, same manufacturer or same make and type. Also, the overall chain fingerprint should have at least lower and probably lower limit on needed states to prevent false positives.

5.8 Load disaggregation

In the ideal case, load disaggregation (separation of loads from combined signal) would be done at sensor level, as it has access to actual current at frequencies where the waveforms are apparent. The case below shows how combined load waveform is created.

The pictures below show 2 simulated load waveforms, which different peak amplitudes and their harmonic frequencies, and the resulting combination. As can be seen, if one waveform and the combination are known, the other can be calculated via simple reduction operation.

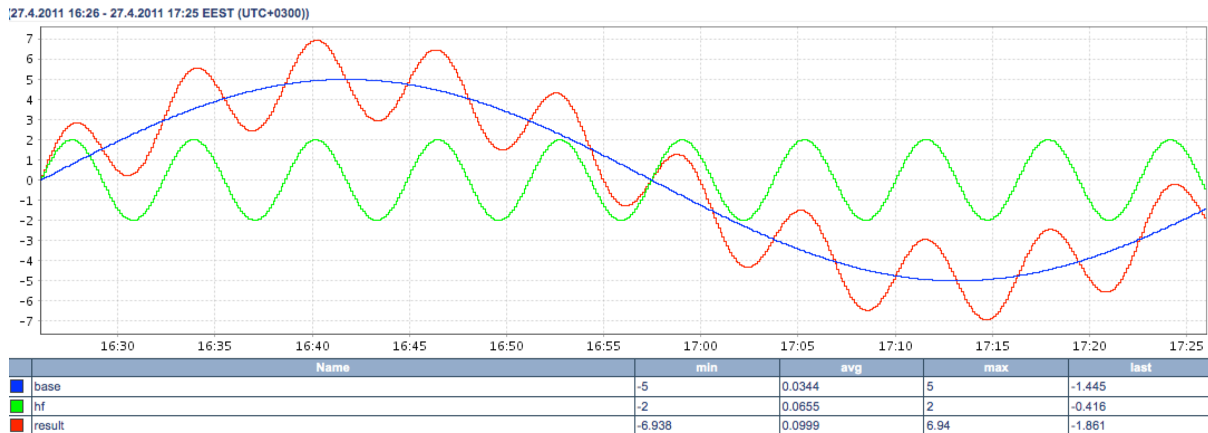


Figure 15 Waveform 1

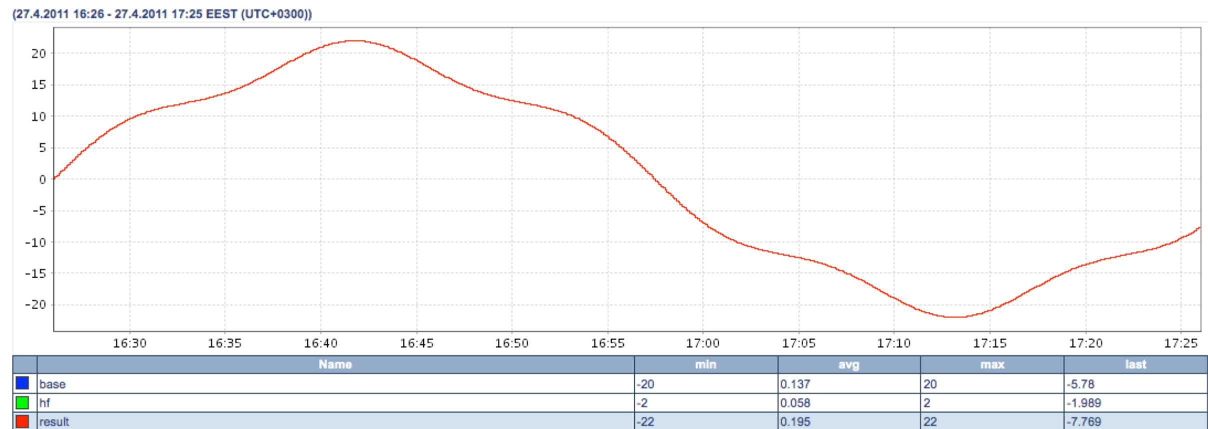


Figure 16 Waveform 2

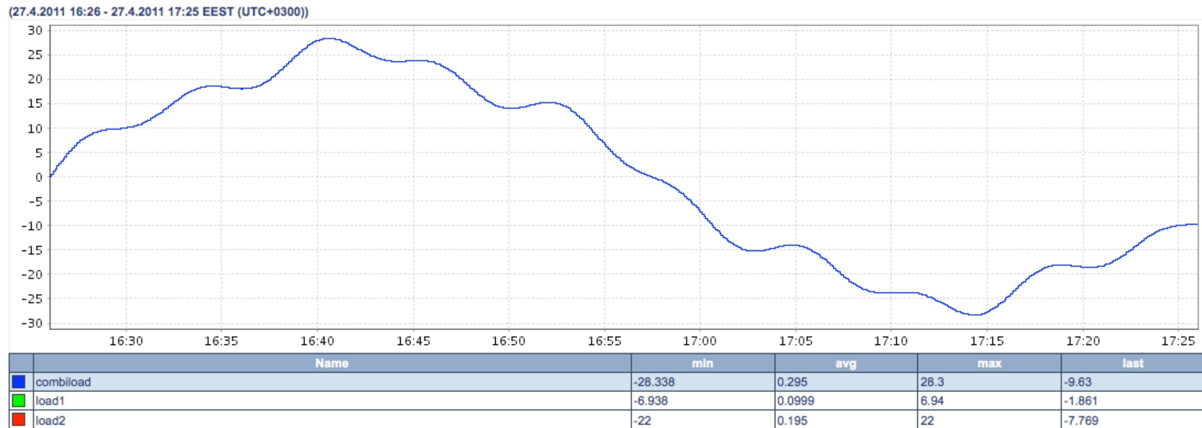


Figure 17 Combined waveforms

Unfortunately the data sent to storage is just a snapshot of summary values -with sensor doing calculations at 1kHz, data storage gets data at 1/2 Hz frequency. This in particularly means, that for example 2 loads that both have harmonic components at 3rd frequency can end up in a situation where those waves end up canceling each other, sum looking like the system did not have much of HF3, even if the case would be detectable at wave level. Thus the above simple system is not available and different approaches were tried.

From the sensor data, we can also calculate apparent, reactive and disruptive power. Their relative values tell us, what kinds of components the system is having., allowing us to at least separate resistive loads (no disruptive or reactive components to speak of) but still leaving the question of separating power electronics from the rest.

After several tries, a simple system that tries to account for aggregated loads with a combination of harmonic analysis, using load components and heuristics was suggested. It is based on the following observations:

Harmonic components HF3,HF5 and HF7 are relative to apparent power (ratio), thus when waveforms do not cancel each other out in ideal case, we could for each harmonic frequency use

$HF_{sum} = \frac{P_1 * HF_1 + P_2 * HF_2}{P_1 + P_2}$, where P_2 is average apparent power of load 2 and HF_2 is average harmonic frequency ratio of load 2. Thus when new steady state is detected, its' power is $P_{dev} = P_{new} - P_{old}$ and from the equation above, the harmonic distortion for each frequency can be estimated to be $HF_{dev} = \frac{(HF_{new} * P_{new} - HF_{old} * P_{old})}{P_{dev}}$.

In cases where this would mean that HF component of new load is negative, we must assume that waveforms are canceling each other out.

We also know that resistive loads have no noticeable harmonic content, in motor-inductive cases HF3 dominates whereas for power electronics there is significant component in HF3,HF5 and HF7.

Thus these rules combined when a new load is added:

- If relative disruptive and reactive power goes down (or stays at near zero), load was resistive
- Check each harmonic frequency for following:
    - If there is relative increase, flag HFx as having increased.
    - If relative value stayed same, flag as HFx having increased
    - If relative value dropped more that above equation would suggest for resistive load with HFx==0, flag HFx as having increased
- If only HF3 was flagged or if HF3 is much higher than HF5 and HF7 combined, load was motor inductive, otherwise electric.

In each case, staying same or increasing is controlled by a configurable parameter.

As explained in the chapter 5.4, current steady state detection is based on standard deviation, so very small loads compared to current overall consumption cannot be detected with this system, as they would not trigger a new state. They unfortunately do impact somewhat on harmonics, but fortunately impact is proportional to their power, so in usual cases effect to average will also be small. Also, as this system is based on estimating the harmonic components from the whole, it degrades as more loads with large harmonic components are added. It must also be noted that using only harmonic components to this analysis does not give as good results as full fingerprinting, as for example some fingerprints categozied as resistive do contain some harmonics - particularly dishwasher and wasing machine in heating cycle, when machine is doing other things, too. Thus it cannot yet be considered a production quality analysis but first steps towards such a solution.

A completely different approach would have been load disaggregation from power data just based on typical load patterns, basically answering the question "which combination of known loads would produce this result". This approach was not chosen, as we felt it more important to see what options the power quality data would give us.

6 Sensing infrastructure changes

As also the sensing infrastructure has had some changes since D3.7, they are covered briefly here.

Old base station, Via Artigo had two serious problems - very noisy fans and energy consumption of almost 15W, alternatives were tried. FitPC2 was chosen, as it provided the same functionality, with energy consumption reduced to 8W and wholly passive cooling system.

The initial version of BeAware sensor had some design flaws, so a new design was made and taken into use in trial 2. It basically has the same functionality as D3.6 explains, but has added the harmonic currents information for fingerprinting.

As the first version of Ambient interface was found to be too intrusive, 2 more versions were designed. AmbientInterface v2 is basically the same design, but supports a more configurable protocol interface (detailed in D.6) and Wattlite Twist is based on wholly new ideas, for Wattlite Twist, see D4.5

J2SE based base station was planned, but in the end resources were switched to load fingerprinting, as it was decided that just switching programming language for one part of the system would not benefit the project.

7 Results

7.1 Advances in state of the art

The main advance in state of the art for BeAware wp3 are twofold.

- Ability to process 1 Hz level data from multiple sources without need for custom build system. DEHEMS (www.dehems.eu) was tackling a similar problem of massive scale and solved it simultaneously with a different approach (http://www-01.ibm.com/software/success/cssdb.nsf/CS/STRD-84XL7P) - full detail are not available, but the BeAware system was already used to read similar quantities of minute based measurements at project start, so pushing the limit to 1/s for each measurement node gives us a possibility for load fingerprinting and more detailed advice.
- Load fingerprinting based on the measurement data was demonstrated both in the laboratory and in pilot sites. While full 1 sensor covers whole house approach was not possible with the current solution, fingerprinting in NIALMS system was shown possible in real time and some preliminary solutions for multistate and disaggregated analysis where developed.

As shown in D3.8 exact device identification had 60% hit rate inside the known devices, so was not sufficient for real world applications. Fortunately the main type detection has over 90% success rate on single mode devices with most problematic devices being motor-inductive with high HF5/HF7 components. Also, system was over 95% correctly able to detect devices which exhibited multiple different states - only problems are small fluctuations harmonic content which sometimes trigger electronic event on small resistive/motor consumption.

Multistate mode/type detection had almost 100% hit rate on dishwasher and dishwasher mode which exhibited very steady operational cycles, whereas with washing machine type hit rate was roughly 80%, and mode detection within confirmed washing machine 60% with current system having problem with false positives. With washing machine, particularly higher temperature modes differ mainly in resistive cycle time, whereas rest of cycles are same. Also, with washing machine the amount of clothing in machine does affect the states slightly - this is particularly evident when washing machine starts its' spinning cycle, higher loads not reaching a steady state.

Single mode devices were tested against trial2 setups with sensors that were reading only one device, whereas multistate detection was only tested on 3 internal pilot

installation machines with a total of 5 fingerprinted unique states, and rest of state used for testing against false positives.

It must be noted that personal computers proved to be very hard devices to fingerprint, as their fingerprint changes so much with usage pattern - how much CPUs are used, how much GPUs are used, how fans are running etc - they can basically change between several states based completely on how machine is used.

With the current trend of customers becoming producers of electricity, there is also an increasing need of easily available power quality analyzer for detecting what amount of noise and distortion are introduced to the grid. Currently the commercially available examples (www.electrix.fi) start from 2000e and upwards, so BeAware sensor could be seen as a viable alternative. So if BeAware sensor is commercialized, it could be seen as advancement in state of art for openly available such sensor.

7.2 Challenges encountered

- Amount of data: The initial challenge for wp3 was the immense amount of data available. Single BeAware equipped household produces almost 130 million measurements annually. so handling that needed some redesigns - fortunately these problems were identified in the first prototype trials conducted inside the project, so did not impact actual customer trials
- Sensor instability: The first version of sensor design had to be abandoned due to insufficient hardware specifications and the second version was built on top of hardware which still had some bugs in it. Due to this, a physical filter had to be attached to sensors measuring fridges, as the startup spikes would occasionally cause the A/D converter to malfunction. Also, as the harmonic current frequencies were added only to latest version of sensor, THD calculation was needlessly complicated as is tried to filter out the noise from measurement - with the HF3-HF7, this filtering would not have been necessary.
- Data format: the whole field in unfortunately plagued with the lack of de facto standard like SNMP is on the telecom side. In Italy, COMSEL is seen as most probable standard, whereas in Finland industry was lobbying for wireless m-bus and several commercial players like Plugwise are using their own standards over ZigBee. BeAware transfers relatively lot of data, so needed very efficient protocol and thus chose to implement its' own.
- Customizable analysis: Initially it was thought that the iterative software process would give enough feedback the algorithms coded to the system would remain relatively stable in customer trials. It was quickly noticed that in fine-tuning the game, algorithms needed to be both parameterized and even changed quickly. Thus BABUP protocol now gives the option for parametrizable content.
- Physically BeAware servers are located in Italy and Finland, with measurement sites also in Sweden. This means that for family in Sweden, data is first

transmitted to Finland to data storage, from there to Service layer in Italy and then back to Sweden. Transmission delays between data centers will accumulate and thus the responsiveness of system is degraded. In commercial solution, all servers should reside in same place, with at least some geographical mirroring available.

- Originally the system was designed so that each measurement site would have a public ip for base station for ease of upgrading and debugging. Unfortunately this was not available in Italy, so upgrading software needed manual intervention. In production system, this should be replaced by automated update process, which periodically checks for updates.
- Steady state analysis, cornerstone of our fingerprinting, is based on standard deviation as one of the criteria for a steady state. This creates a situation where larger loads will dominate the system hiding smaller ones. For load disaggregation systems, this should be changed to something more adaptive. Similarly, for a more general-purpose system, it should take into account non-linear states, like regular waveforms etc.

8 Future

It is quite clear that for any sort of household energy measurement system to become truly widespread, customers need to be able to hook their own sensors to it, necessating a common protocol, which can handle both large amounts of real time data and infrequent large batch data transfers. Also, a local bus to wide area net converters cost money and add a new layer of potential faults to system, so most probably a system, where main meter would communicate with a common standard would be technically best - SNMP over IPv6 is already a such standard in telecom, so would probably meet these needs too. Main meter should have an interchangeable communications module for at least Ethernet, GPRS/3g and PLC, with the protocol being the same. Main meter could also act as the local bus gateway if needed, keeping the needed equipment to minimum. EU standardization mandate M/441 EN aims for at least local bus standardization, but the initiative is already late and it remains to be seen if a real de facto standard emerges.

As stated in 7.1, power quality from small-scale suppliers might become an issue with widespread use of local solar panels and other such equipment. Having too much distortion in power causes degradation in both equipment and causes transfer losses, so it very nondesirable. So meters having just the basic capability of THD+noise analysis are probably needed at much lower cost that they nowadays are available.

BeAware trials proved that having household with multiple measurements per second and analyzing that stream of data is easily feasible with modern equipment, even if power companies are currently hard pressed to provide even previous 1 hour value to their customers. Having a real time data available allows for engaging the user and providing him with timely information, like advice that are pushed to him immediately,

generated via fingerprinting and advice system like in BeAware. Having similar system also in water and gas pipes would also allow immediate detection of leaks, which at least in Finland amount for tens of percent loss for water companies and can cause major damage to private owners, if not detected early enough.

BeAware will open the algorithm, fingerprint and multistate device chain libraries for open public, allowing for other projects to further research that field. If load disaggregation was developed even bit further, it would allow for metering the household with 3 meters connected to mains, instead of having several meters, allowing for much cheaper installation while keeping the ability to separate at least main loads.