



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) | |


| | |
|---|---|
| **Programme Name:** | ICT |
| **Project Number:** | 224557 |
| **Project Title:** | Boosting Energy Awareness with Adaptive Real-time Environments |
| **Partners:** | COORDINATOR: TKK (FI)<br><br>CONTRACTORS:<br>Helsinki University of Technology, TKK<br>BaseN Corporation, BaseN<br>Interactive Institute II AB, II<br>Engineering Ingegneria Informatica, ENG<br>University of Padova, UNIPD<br>Enel.si, ENELSI<br>IES Solutions, IES<br>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, Erik Bunn |
| **History:** | 0.9 Final draft - Topi Mikkola<br>1.0 Reviewed - Tatu Nieminen<br>1.1 Summary updated - Topi Mikkola<br>1.2 Language check - Erik Bunn |
| **Availability:** | [public] |

# Table of Contents

# Executive Summary

This document covers the second part of wp3, namely Data Storage, which 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 the whole BeAware system. Also a brief overview of the underlying cloud system is given. A single BeAware meter produces over 126 000 000 measurements per year and one household has upwards of 10 sensors, so the amount of raw data coming to system is non-trivial.

The second part documents the available public interfaces and services available through them. Also, the chosen transmission protocol BABUP, a Google Protocol Buffer based system, is described and details on how to use it are provided. The decision to develop a new protocol was made because the amount of data means that message processing must be as efficient as possible. While BABUP was created for load quality data, it can be used with any other measurements as well.

Another main task of Data Storage is the analysis of incoming data. It is analyzed in real time for any trigger conditions such as power dropping to 0 or no data from sensor, and administrative users are alerted if needed. All normal statistical analysis tools (averaging, data aggregation, comparisons) are provided, but in addition BeAware Data Storage can apply so called load fingerprinting, or load identification, to any given sensor measurement. This system allows identification of the type of measured load by comparing it against known loads. This fingerprinting is also available is real time, and can alert users if, for example, a device is malfunctioning. Current fingerprinting can identify load type category (resistive, motor or electronic) and we also demonstrate algorithms to use that in identifying multimodal devices (washing machine) and lod disaggregation (multiple loads behing one sensor).

Load fingerprinting and power quality analysis are an important part of the future Smart Grid where anyone can not only use power but also feed it back to the main grid. An approach like BeAware's low cost sensor and scalable analysis approach can easily meet those challenges.

# 1 Introduction

The BeAware sensing layer consists of 2 logically separate parts. D3.7 describes the physical Sensing Infrastructure; this document concentrates on the Data Storage and simply presents an update to new functionality available to the 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 the Service Layer, interfaces for fetching data.

Data Storage has been built to support high granularity data from multiple sources so that all 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 also presents a brief overview of the internal workings of the cloud - all the interfaces and protocols are public. The 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 document D3.1, which in case derives requirements and use cases from D4.1 and D5.1. The base station use cases have been covered in previous documents. Data Storage needs to cover the following use cases:

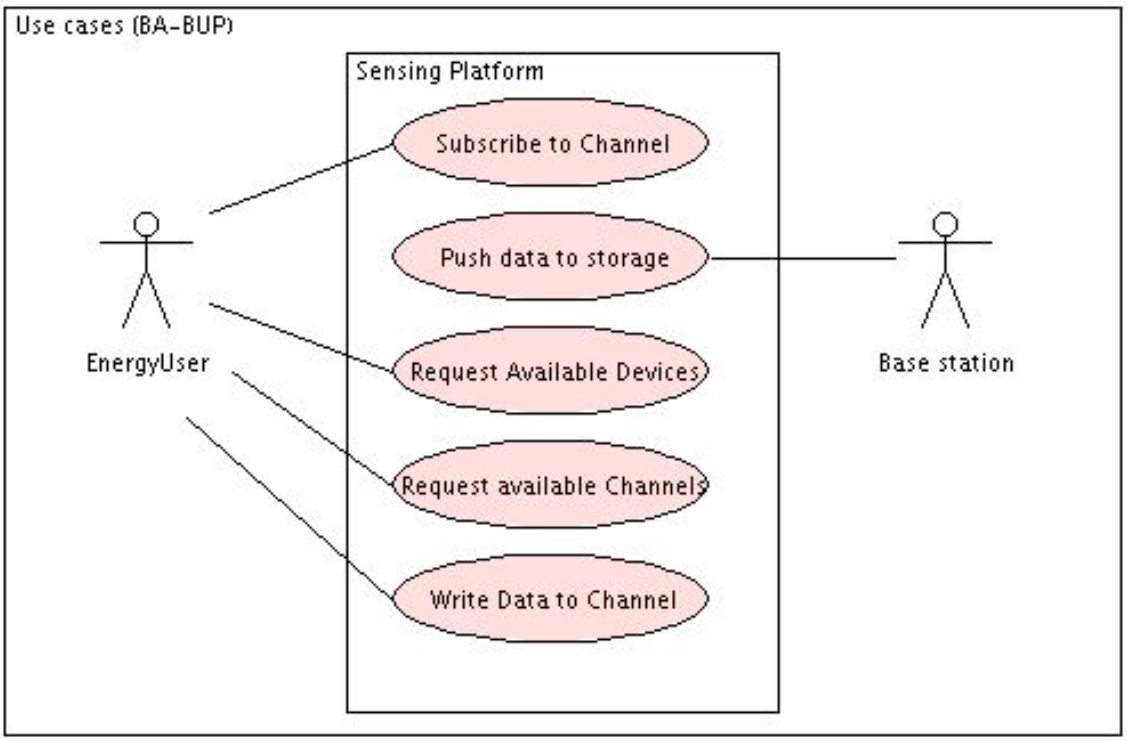



*Figure 1 Supported use cases*

- Subscribe to channel: BeAware was initially thought to work by an asynchronous subscribe-publish pattern , but was later changed to synchronous reading via method calls.
- Push data to storage: Data Storage supports data writes for 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 under each base station.
- Write data to channel: EnergyUser is allowed to write both measurement data and configuration data to a configurable set of channels.

## 1.2 Design principles, system requirements and licensing

With cloud and grid computing, the system can be optimized for several factors. 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 levels.

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

1. The system must be able to handle measurements at 1Hz level
2. The system must be able to update the User Interface within a minute of a measurement value being received.
3. Algorithms must be configurable from an external source, without need to recompile

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

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

# 2 Architecture

In the layered BeAware architecture (see Figure 2), the Sensing layer forms the two lowest levels. The physical sensing infrastructure consists of sensors and base stations 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 numerous available services. The following chapters explain the logical parts of the sensing platform.


*Figure 2 BeAware overview*

## 2.1 Sensing platform

The base station and Data Storage parts of the sensing platform are completely decoupled, all communication happens via BABUP (see **Virhe. Viitteen lähdettä ei löytynyt.**) over HTTP. So the base station itself can be considered just an example of how to implement a home sensor 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.

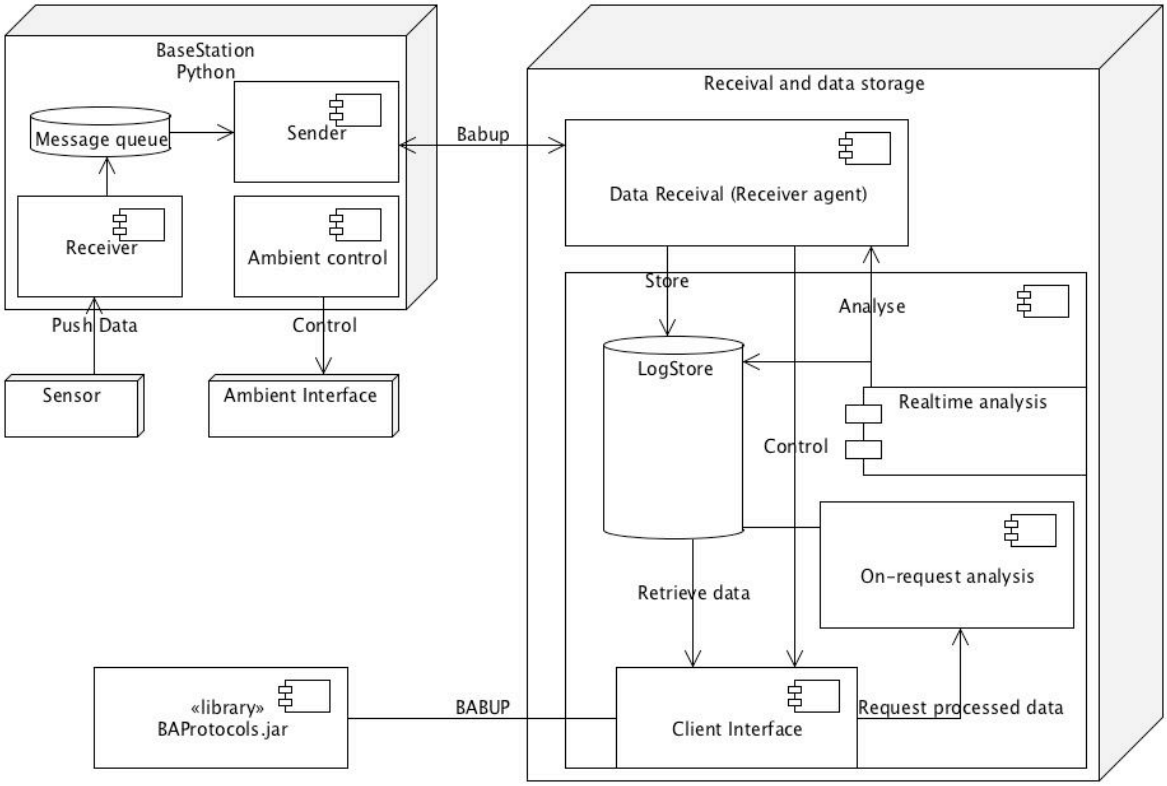



*Figure 3 Data Storage components*

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 at a rate of 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 64 bits, and is tagged also with a 64 bit timestamp and a varying length sensor identifier tag. For production purposes the identifier should also be 64 bits (for completely unique sensor id) plus 8 bits (data channel identifier - power, energy, total harmonic distortion (THD), etc) In trials a longer sensor identifier has been kept mostly for debugging purposes, even if it does waste 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 designed from the start to scale easily.

- The base station is able to cache data for a sufficient time period (in fact, several years) to survive network bottlenecks or outages, and to dequeue the data after connections are re-established.
- The number of data receiver agents can be increased if more processing power is needed. Basically base stations can periodically check which receiver agents they send data to and load balancing can be thus done dynamically.

- The employed service discovery architecture means that new processing power can be added on the fly. All services are logical and each physical computer can run multiple services.
- In the case of BeAware, the more static data like baselines can be precalculated and cached. Cache hits on these cases is close to 100% (last weeks consumption etc.) and cached data is neglible in size.
- Both Data Receiver and analysis are configurable by configuration templates, so adding new measurement points or new analysis options for all users is easy.

The computation cloud itself is protected by firewalls and access control lists, and only the data receiver agents and client, export and visualization interfaces are available to public internet through authenticated 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 description of the actual Data Receiver entity. In the case of BeAware 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 in a project lasting only a couple of years, this highlighted how important it is not to have a single point of failure, as malfunctioning hardware resulted in a several day outage ofthe BeAware test environment Data Receiver. Of course, data caching in base stations meant that no data was lost, but real time functionality was lost for that period.

## 2.3 Data Storage

Data Storage is a computation cloud, handling 3 main tasks: data collection, analysis in both real time and on request, and visualization. As BeAware visualization is done on an upper layer, for BeAware purposes Data Storage collects data and allows various view to it. All data is tagged with the data path and it is associated with a timestamp, which allows the system to distributethe data into the cloud storage and then retrieve it based on the above two keys.

The architecture itself is based on distributed software services coordinated by a service announcement and discovery system. All services are logical and have multiple copies running. Each service instance advertises itself in the cloud network with information of its available features and available data. Every service listens to these advertisements and when it needs a service, it contacts the best matching advertised instance. A simple sequence might be:

1. A data request comes in through the User Interface
2. The User Interface  locates and selects a data visualizer
3. The data visualizer locates and selects an on request data analyzer
4. The analyzer locates and selects 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 performing immediate analysis on it.

Long-term storage mirrors the data to at least two physically separate storage devices andacknowledges receipt only when the data has been stored to the physical devices. Once saved into storage, data cannot be modified. (Even administrative users can only remove data, not change its contents.)

Real time analysis keeps track of most recent data (usually the 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 are met, automatic alerts are triggered and delivered through  an export API (usually email).

### *2.3.2 Outbound data*

The outbound (aka northbound) data interface has two modes of operation: push and pull.

In push mode, the real time analysis (or any scheduled export, such as a monthly summary report) automatically sends informationto the user when a trigger condition is met.  It can be delivered via email, SMS, HTTP request to a predefined address, or just displayed as an alert indication in the User Interface. A trigger condition can be either an upper or lower limit breached by the raw data, missing data, or a predefined result from a filter chain (see next chapter) applied to the data.

In the pull mode the user requests a data sequence with a specified filter chain applied. The outbound interface first authenticates the user and then authorizes his data request, checking that it does not contain entries the user is not permitted to see. After that the filter chain is parameterized with user provided parameters:

1. What data is needed (channels)
2. What period is needed
3. What is the averaging time window, if needed
4. Any additional parameters for filter functions

The filter chain produces a numeric result (usually a time series), which is then converted to the format the user requested - an image or time series data in Excel or BABUP format.

### *2.3.3 Data filtering*

One of the main problems in BeAware is the sheer amount of data. Even a simple power channel produces 30 data points per minute, and a typical smart phone screen can not accommodate even 30 minutes of raw information, so some kind of filtering is necessary. BeAware has several different uses for the same data, all of which use some basic common analysis components, which can be reused. The project also highlighted the fact that for quick prototyping of different Graphical User Interface options, the underlying analysis engine must be reconfigurable without recompiling and restarting 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 subchains coming from ready-made templates. 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 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 gaussian filtering, joining channels with mathematical operations etc.
- Data analysis includes the final stages of analysis where more complex and specialized analysis is needed. This includes future estimates, state detection, fingerprint matching, etc.

### *2.3.4 Available services*

1. Data reception in the cloud accepts data from the receiver agents, and sends it both to long-term storage and the event and alert service for real time analysis. All data path map to unique hash codes and are stored according to them on at least two separate long term storage instances. Data Receiver agent receives an acknowledgement that submitted data is stored only when it actually has been stored to disk in long-term storage.
2. Long-term storage handles storing and retrieving of data. For security reasons, data is always offered as read only to prevent any tampering attempts.
3. The event and alert service analyzes the incoming data stream for any trigger conditions and performs an alert via the specified export service if any of the pre-specified conditions are met. All data is kept in memory for quick access and

analysis via the same filter chains as in on request data filtering. By default, the last 15 minutes worth of data is kept in memory for each path.

4. Data filtering applies a requested filter chain to data. Whereas the event and alert service works with in-memory data, data filtering fetches data from long term storage and can thus handle much longer periods, with practical limit of a few years.

5. Export pushes the requested data to a 3rd party via a selected protocol. BABUP is one example of this, email/SMS alerts of missing sensor datais another.

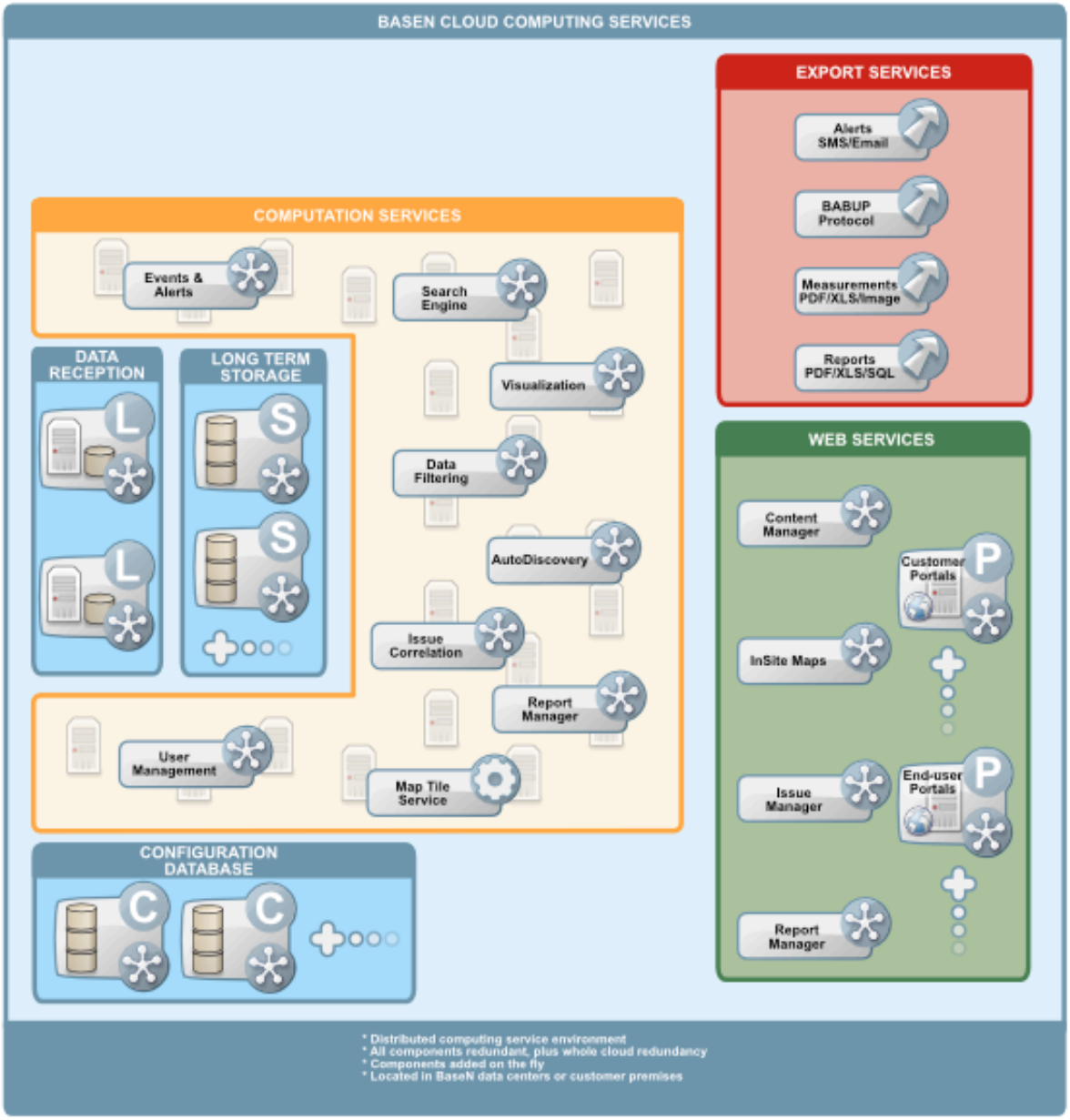



*Figure 4 Data Storage services*

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

The basic data transfer model used in BeAware Sensing Layer uses BABUP (BeAware Base station User Protocol, a Google Protocol Buffers (GPB) based description) with normal HTTP as transport protocol. Both south- and northbound interfaces use the same protocol, while the northbound interface also offers an 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 as well as wire form, where all the data has been packed to binary format, thus making transfer quicker and marshalling operations much lighter than XML parsing.

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

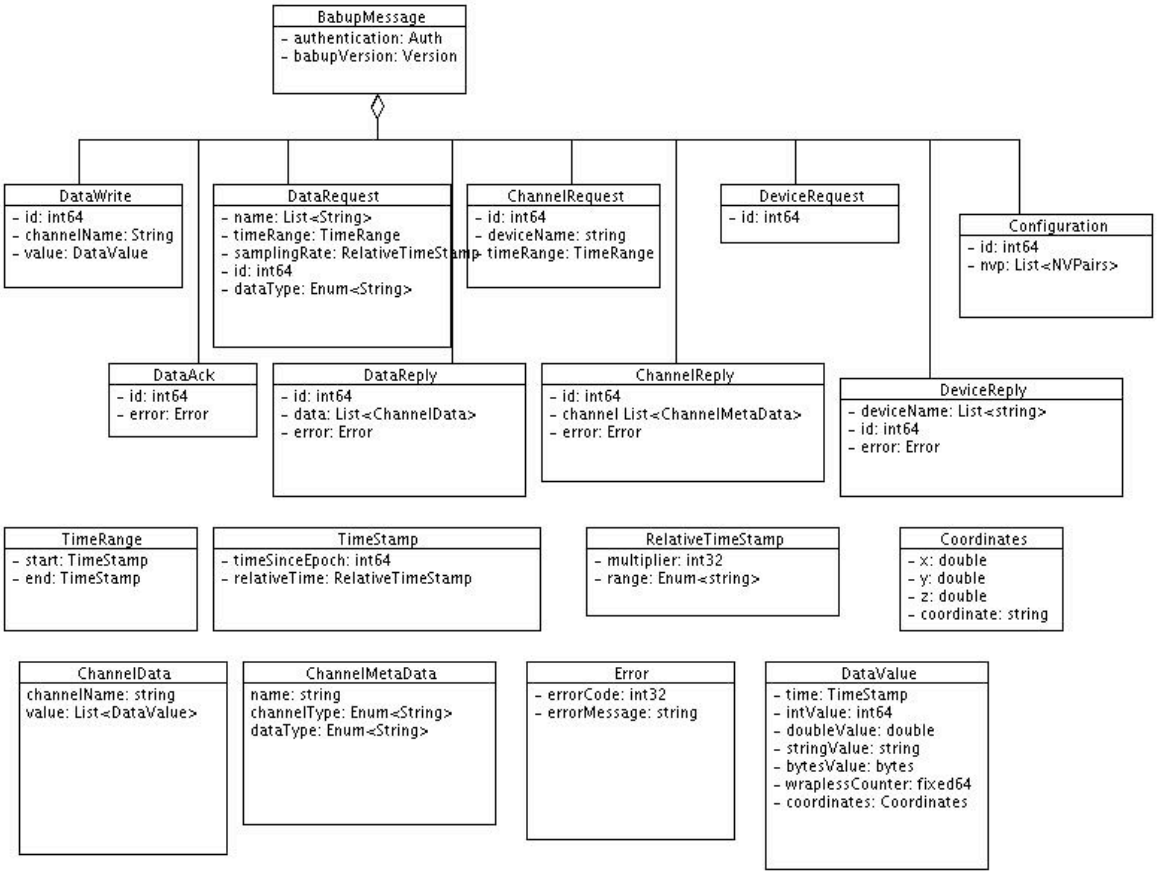



*Figure 5 BABUP sub messages*

### *3.1.1 Using BABUP*

As implementing a new sensor or base station that wants to interface with BeAware system by providing BABUP data is 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 must 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 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 measurements 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. 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 less efficient 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 receival via BABUP, but the Data Storage itself can also receive data from other sources if needed The BABUP interface takes care of authorizing the incoming request and processing the data for storage, and it guarantees that the base station only gets an ACK message when the data has actually 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 a base station needs to read data, it can access the northbound interface, which supports both read and write operations.)

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 HTTP response.

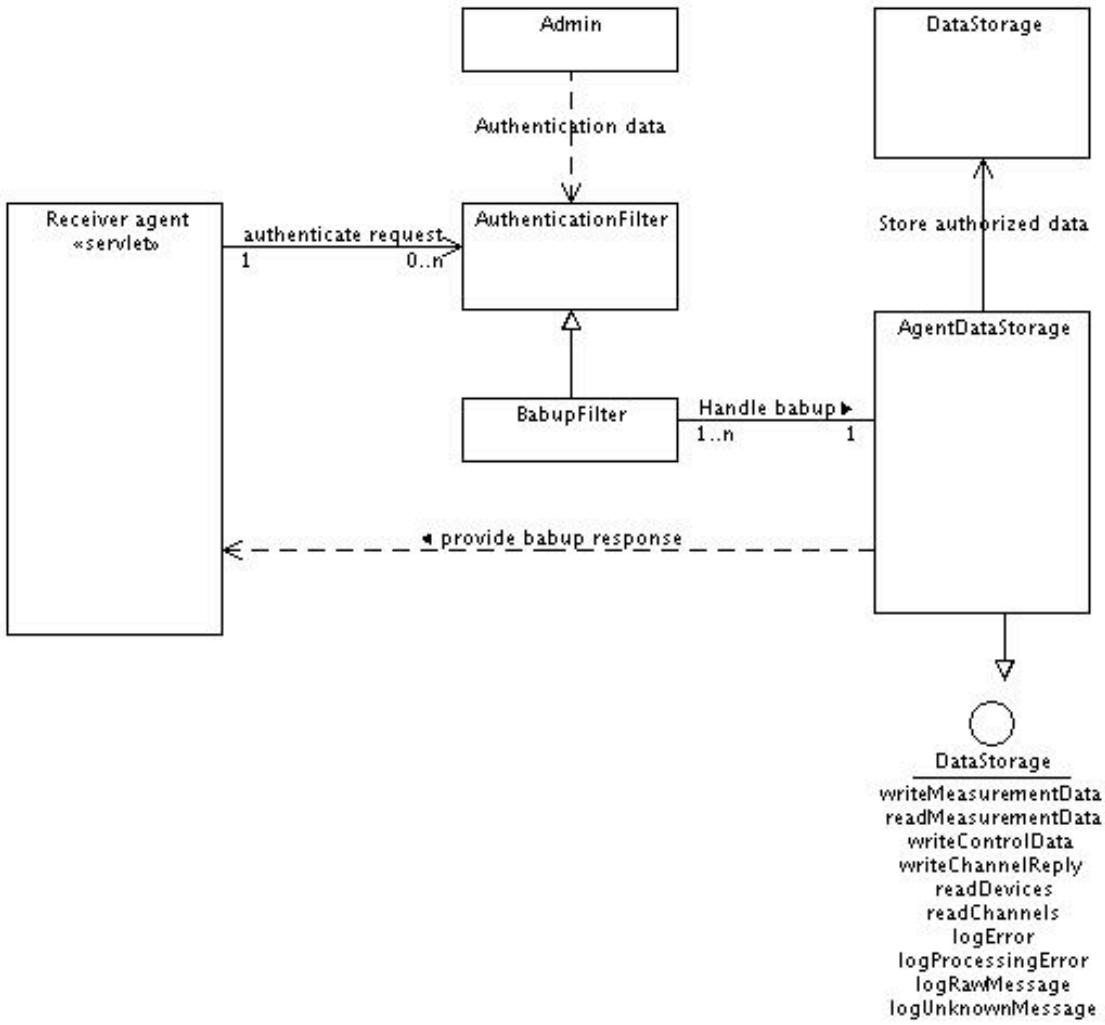



*Figure 6 Data receiver interface*

## 3.3 Northbound client interface

Data Storage provides a similar BABUP based servlet access point for upper application layers. In practice, this is hidden in the Service Layer library BAProtocols.jar, which contains the following functionality:

1. BabupClient abstracts the BABUP over HTTP communication into a Java API
2. BeawareBabupClient extends the client with BeAware specific functionality

This code is available in the BeAware repository. The user is advised to read the documentation therein for the workings of the java library.

The underlying BABUP receiver is similar to 3.2, but as this interface provides also read and control access, in addition to authenticating the request, the system also authorizes

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

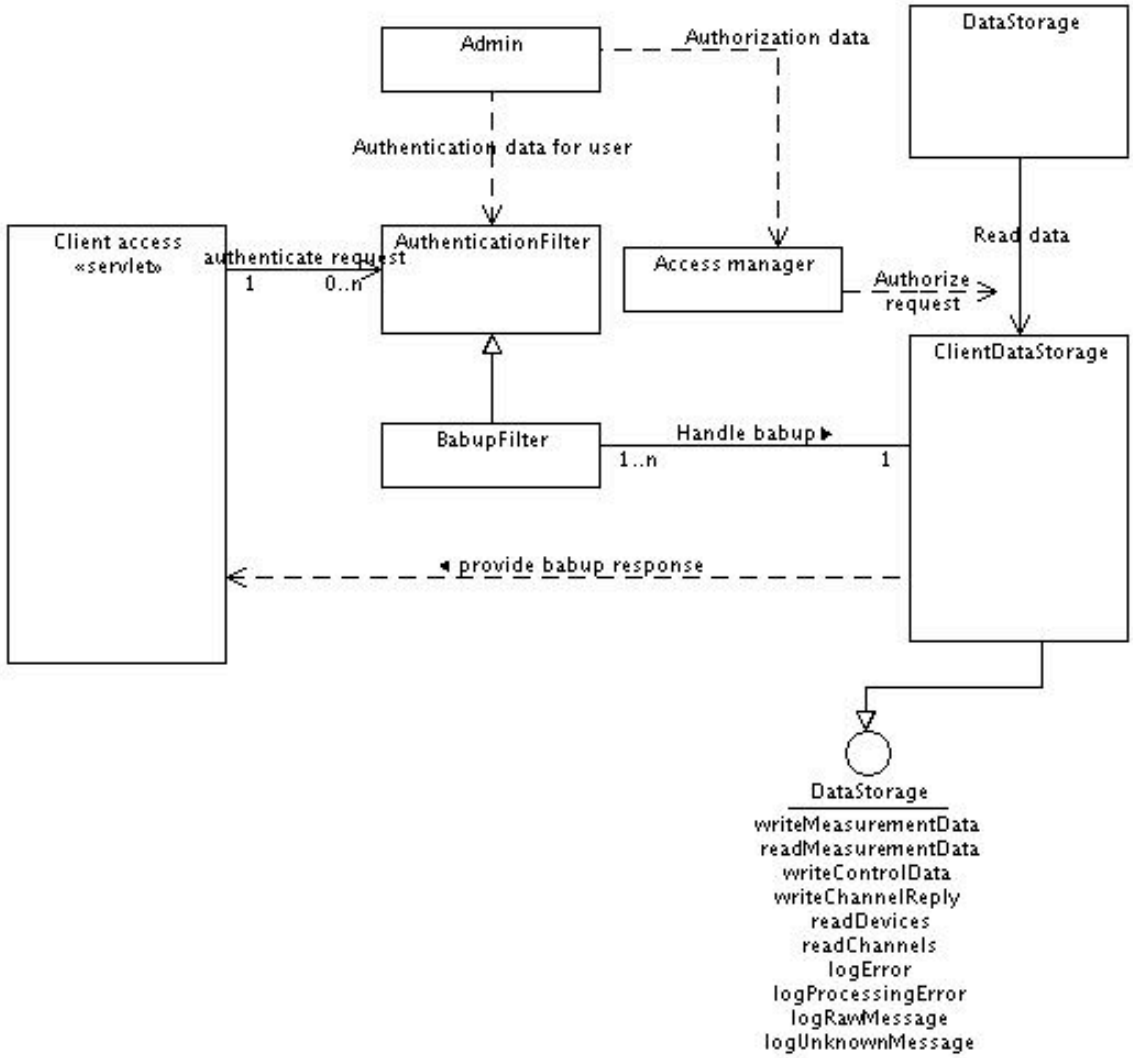



*Figure 7 Client Interface*

In general, unless the available data is known, the northbound interface is accessed in 3 steps, as shown in Figure 8. First the user checks what available devices (base stations) he has via DeviceRequest, then he can check what channels (name, type) are available in 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.

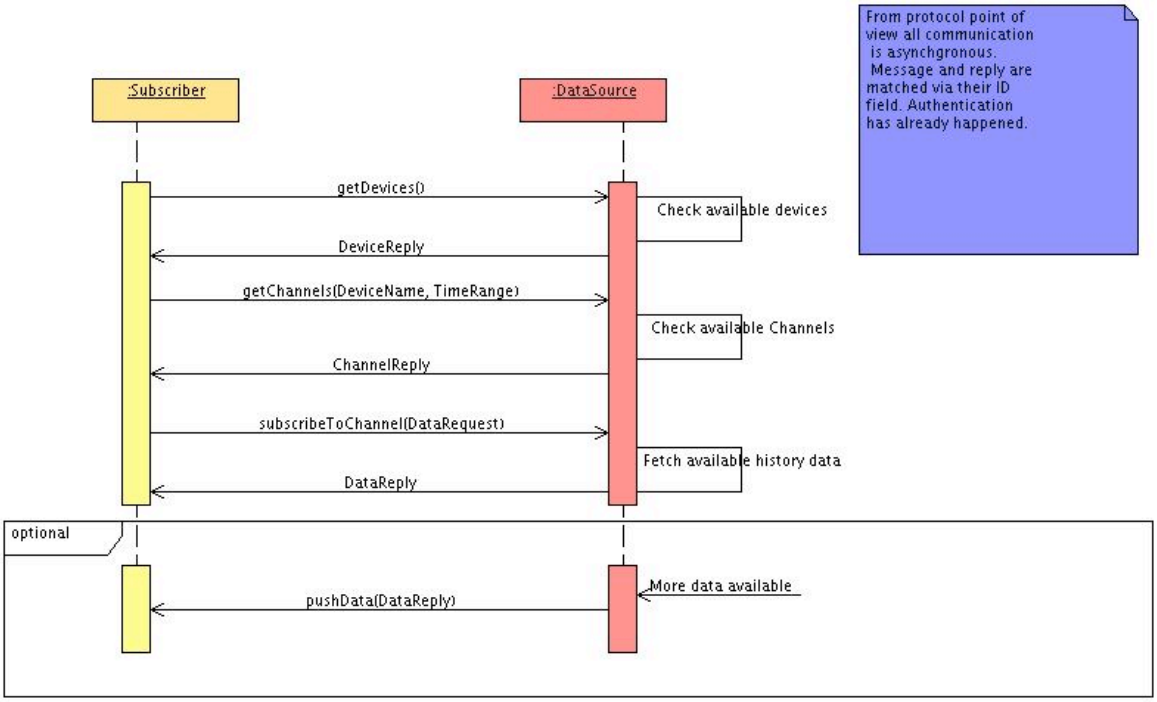



*Figure 8 Data retrieval process*

## 3.4 Other supported protocols

### *3.4.1 Base station*

Base station capabilities have been explained in further detail in D3.7 This chapter presents just a brief overview.

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

### *3.4.2 Data receiver*

Unlike the base station, Data Receiver has been written is Java and is part of the BaseN Platform proper. Java has very good support of various industrial protocols, so these are in turn supported by 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:

1. SNMP: a 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.
2. 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.
3. 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 the EU. (See www.modbus.org, www.m-bus.com)
4. XML, CSV and binary files over SMTP, FTP or HTTP.

# 4 Analysis capabilities

The underlying engine was previously tested with one minute resolution data, and as expected the BeAware data (one second resolution) pinpointed some bottlenecks, the most obvious being efficient handling of the raw data analysis for user interface baseline calculations, which require 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

While the platform performs the two tasks of real time analysis to arriving data and on request analysis to stored data, the system used for both tasks is the same, and only the data source beneath is swapped 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 the default being 15 minutes.

The calculations, as explained in 2.3.3, are based on chains of configurable filters, each performing a computational 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. The BeAware sensor does providecumulative energy, but as the Plugwise POL protocol only provides immediate power, and as sensors are mixed transparently, cumulative energy is calculated from the integral of power in the case of the BeAware sensor as well. For the purposes of BeAware, the current power display uses a time period of the last 5 minutes of available measurements. (Lst 5 minutes is used, as due to transmission delayes we cannot always guarantee that data from previous minute is available.)

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

In the java interface method to use is BabupClient::getChannelData

## 4.3 Baseline

Baseline (a reference figure to compare own usage against) is one of the algorithms that changed a lot during the project. The initially used "3 previous months' average" meant that particularly in the Nordic climes with direct electrical heating, it was completely impossible to meet the baseline (achive savings) at household level when the weather was cooling and, similarly, almost impossible not to exceed the baseline when the weather warmed, as the baseline would always lag behind. So currently the system gives 3 options for baseline:

1. Previous period against current period (currently previous full week's average against last day's average)
2. This calendar month's average consumption from last year, if available, otherwise this month's average consumption thus far.
3. A predefined constant for this month. (Usually from bill.)

For a more complete system, an option for thermal correction of the baseline should be available, but due to resource constraints this feature was skipped in favor of more research in fingerprinting.

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

## 4.4 Standby detection

As reducing device standby consumption is one of the easiest ways to save energy, detecting such states was added as an independent functionality. Standby is currently defined as lowest detectable steady state that is at least 30s long, separates valid active 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 the range of tens to a 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 fingerprinting. As itcan detect both the device type and then the different operational modes to some degree, standby detection could be done by separate fingerprints.

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

## 4.5 Usage cycles

For devices with cyclic power 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, along with the length and energy used for each cycle. Even this simple cycle detection algorithm can give us information on how a user's habits regarding the device are changing, and warn the user if the device behavior is changing (for example, if consumed energy changes radically, something is probably broken unless it is a multistate device;or, if the frequency of a freezer's cycles grows, it probably needs defrosting.)

Method to use is BeawareBabupClient::getEvents

## 4.6 Alerting the user

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

Administrative users monitored each test site for availability, both at the base station level and at sensor level. This gave us the ability to immediately see how the system was working and whether 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 with tailored advice based on their individual consumption habits was one of the main goals of BeAware. This functionality is a shared effort between the Service and Sensing layers. BeAware support three levels of advice:

- *Normal advice* is based on informing users of expected energy usage of appliances and how to lower it, and on 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 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 analyze the power quality data to check whether 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:

1. 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.

2. Equipment manufacturers can improve quality and compliance, thus providing more energy efficient products to the market.
3. Aging and abnormally operating appliances that consume higher amounts of energy can be spotted and remedial action can be taken.
4. 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.
5. 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 third harmonic current is dominant over the other harmonic orders.

## 5.2 Fingerprints and load library

The fingerprint used consists of the following variables:

1. Active power (P)
2. Fundamental power factor (FPF)
3. Total power factor (TPF)
4. Total harmonic distortion (THDI)
5. Crest factor (CF)
6. 3rd harmonic current
7. 5th harmonic current
8. 7th harmonic current

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

The load library will be available as part of the BeAware website at a later stage. A full load library entry consists of

- Device name
- Unique id (assigned by the 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.

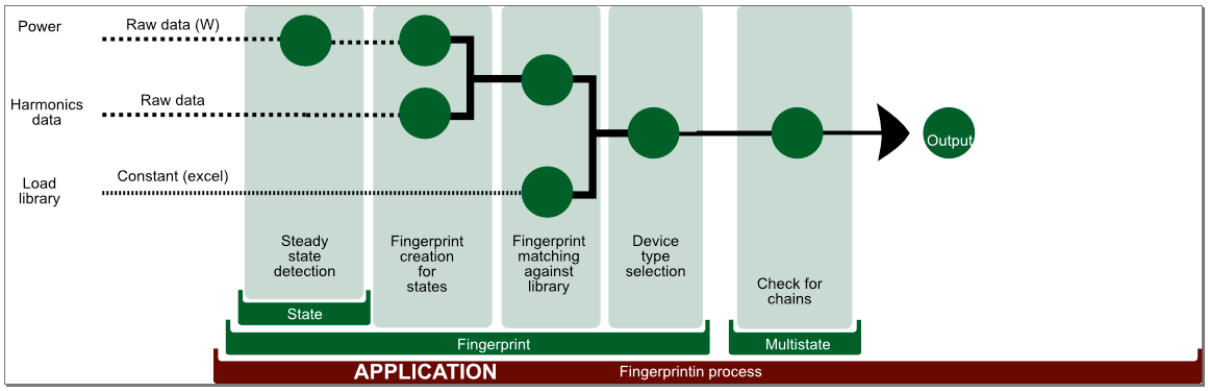



*Figure 9 Fingerprinting process*

- The steady states from the Power channel are detected.
- Other power quality values for each steady state period are read and a fingerprint is constructed
- The fingerprint is matched against library
- If analysis detects that data from single sensor contains several different fingerprint results, a sequence of these is matched against multistate device fingerprint.

## 5.4 Steady state detection

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

1. Filter the samples with a median filter if necessary

---

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

Arto Meriläinen

Helsinki Institute for Information Technology

Available at project website

2. Filter unsteady states, where the change to the last measurement is more than the sample buffer standard deviation. If this happens, the state has been exited.
3. Check for state transition. If the value is outside the confidence interval calculated for the sample buffer, the state has been exited.
4. If the system is currently in an unstable state, but measurement change was withinstandard deviation and confidence intervals and the sample buffer has more samples tha state minimum length, enter a new stable state.

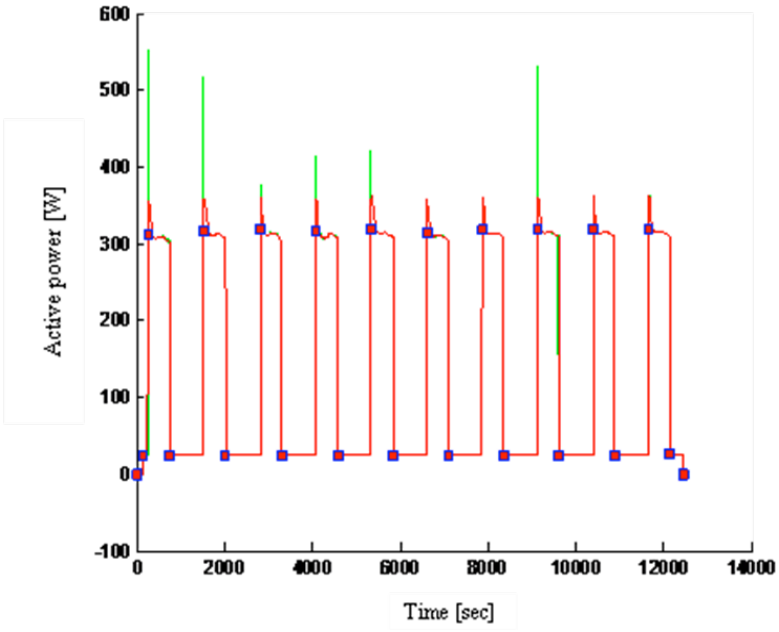



*Figure 10 Steady state detection*

The figure above demonstrates the algorithm applied to a refrigerator. The green line is the unfiltered measurement series, red is the median filtered and the 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, the value of which falls within the standard deviation of previous loads. 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 Mains type detection

As described in D3.8, the harmonics based fingerprinting system is able to identify three classes of main load types:

1. Resistive
2. Power electronic
3. Motor inductive

Unfortunately the variation inside the groups is such that at least with a limited capacity load library, the exact subtype of device (vacuum cleaner inside motor inductive group, or computer inside power electronic) cannot be detected with high enough certainty that the algorithm 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 d(x,y), where j is the number of attributes in fingerprint (see D3.8) and $\sigma$ is the standard deviation for attribute j.

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

The final class label is decided by unweighted voting by the K nearest fingerprints.

## 5.6 Device subtype

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

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

The approach chosen produced roughly 60% hit rate on known devices; in the case of completely new devices it resolves the main type correctly, but can not distinguish the subcategory. Further research on this subject is needed, using transitional signatures with higher resolution sensors, taking on-cycle time into account etc.

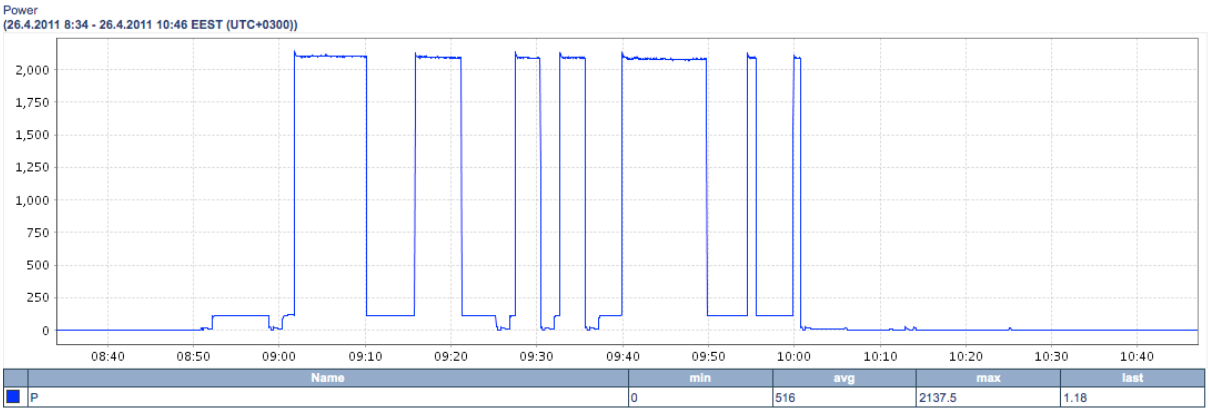
## 5.7 Multistate devices

Many household devices do not have a single operation mode, but produce varying results: a fan with a heating option might show as a motor or resistive load, 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.

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

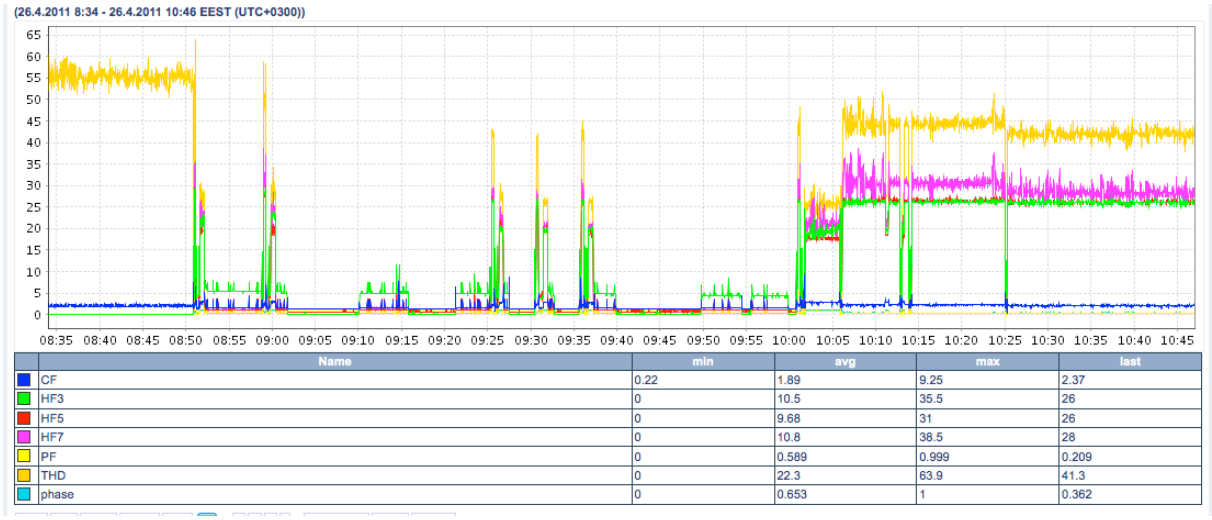
As the state analysis is based on steady states, it was necessary to add two new kinds of states to the system: *undefined* for periods between valid states where power is consumed, but the state is not stable, and *switchoff* for ending a chain when power consumption goes to zero.

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




| Name | min | avg | max | last |
|---|---|---|---|---|
| P | 0 | 516 | 2137.5 | 1.18 |

*Figure 11 Raw power measurements*




| Name | min | avg | max | last |
|---|---|---|---|---|
| CF | 0.22 | 1.89 | 9.25 | 2.37 |
| HF3 | 0 | 10.5 | 35.5 | 26 |
| HF5 | 0 | 9.68 | 31 | 26 |
| HF7 | 0 | 10.8 | 38.5 | 28 |
| PF | 0 | 0.589 | 0.999 | 0.209 |
| THD | 0 | 22.3 | 63.9 | 41.3 |
| phase | 0 | 0.653 | 1 | 0.362 |

*Figure 12 Power quality measurements*

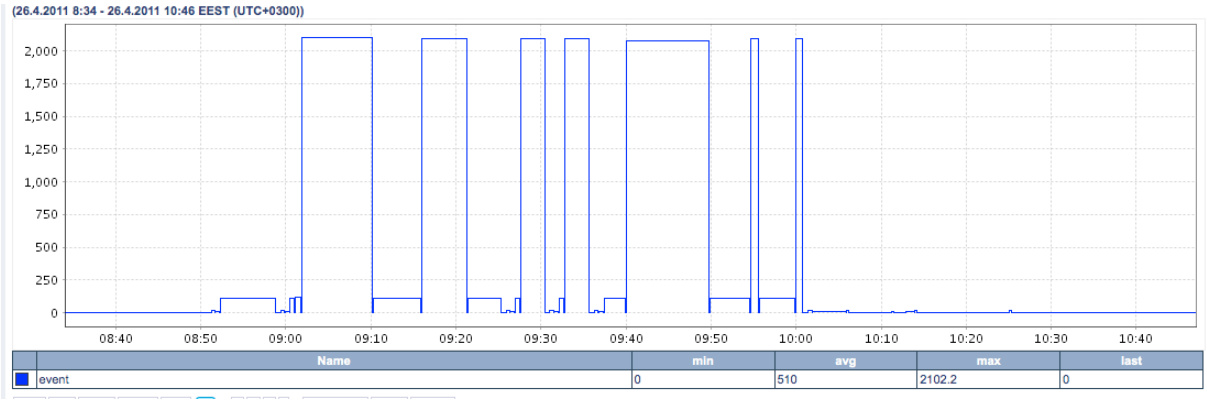



*Figure 13 Detected steady states*

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

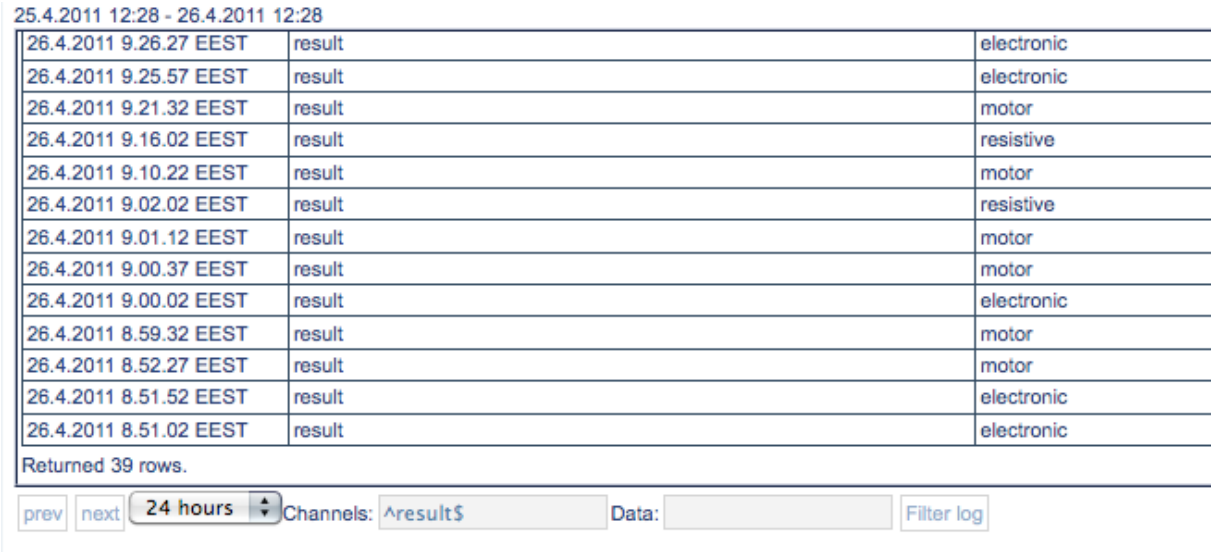



*Figure 14 Detected sub events*

The chain of transitions is thus electronic - electronic - motor - motor - electronic.

From a test set of this data 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 the number of transitions in a normal device during a full operation cycle was usually so large that detection started to vary too much due to random variables: for example, whether the start-up of the device was noisy enough to cause two separate 'electronic' events or not. Basically the probabilities of longer chains became so low as to be meaningless.

To account for this, first a system where an adjacent state of same type was combined was tried. Thus, the previous chain would become electronic - motor - electronic - motor - resistive. This alone produced much better results, but identifying different modes of device was still problematic, as single device tends to have same sequence of transitions regardless of mode - just the amount of sequence repetition changes. The final iteration of the system differed slightly from a pure Markov chain, as a combination of basic states was considered a "super state". For example, in a tested dishwasher, the water heating

cycle looped over "motor-resistive-electronic", with eco mode looping only once or twice, whereas the 65C long mode looped multiple times. Thus for eco-mode the 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 the load library. Its 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 possible values are:

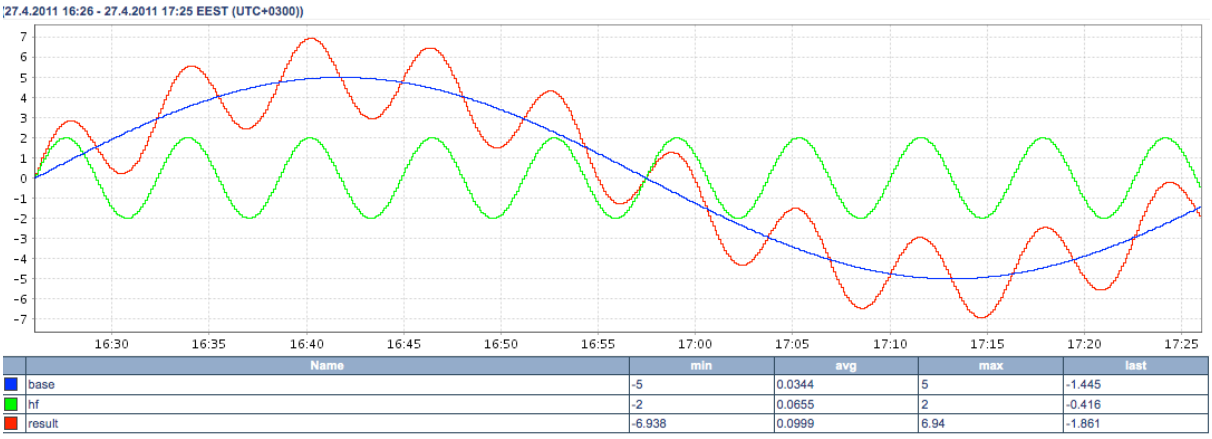
- resistive
- electronic
- motor
- switchoff
- undefined

Due to the small number of test devices the project had sufficient access to for full recording, the system has not yet been excessively tested but at least preliminary it gives good results and tends to err on side of "Unknown" rather than false positives, as most errors are due to an unknown state that does not match a super type template. As the data from trials is still under analysis at this writing, we cannot yet estimate how accurate chain fingerprints from one machine are when applied to another device of the same category, same manufacturer, or same make and type. Also, the overall chain fingerprint should have at least low and probably high limit on needed states to prevent false positives.

## 5.8 Load disaggregation

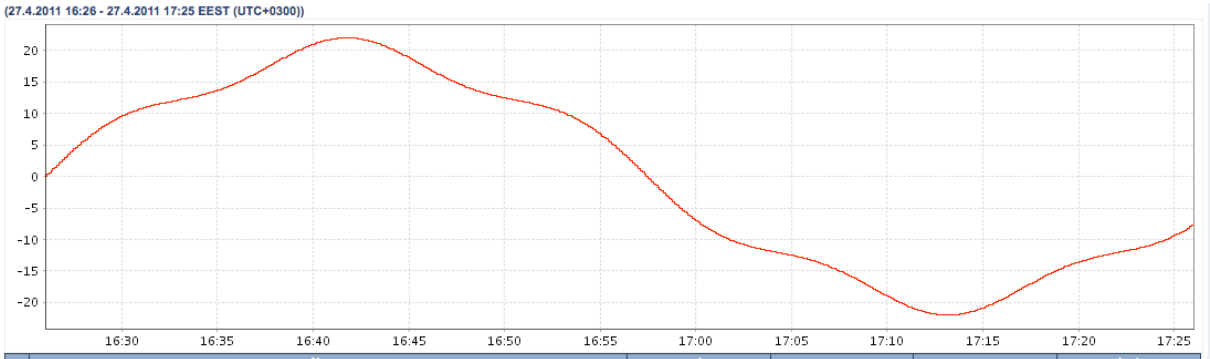
In the ideal case, load disaggregation (separation of individual loads from a combined signal) would be done at the sensor level, as it has access to actual current at frequencies where the waveforms are apparent. The case below shows how combined load waveform is created.

The pictures below show 2 simulated load waveforms, with different peak amplitudes and 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.



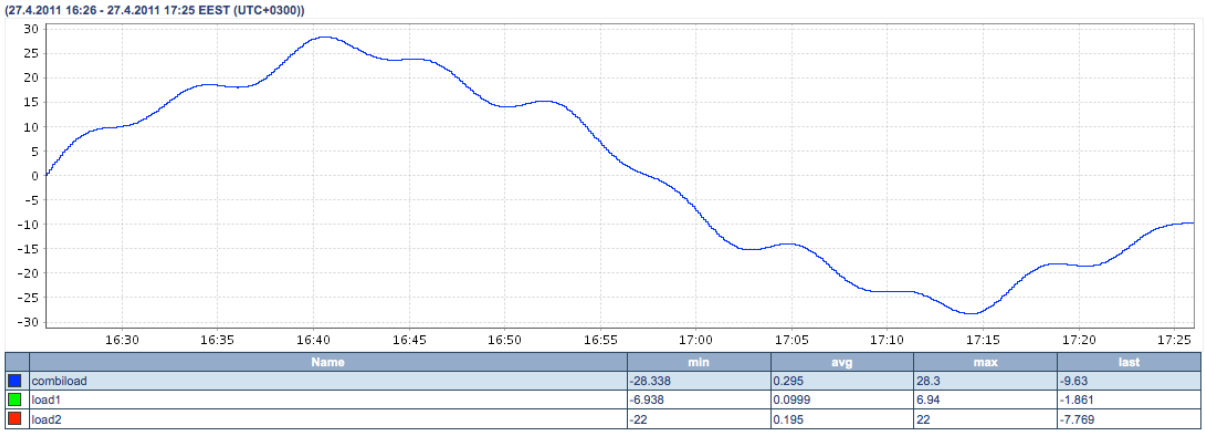

| Name | min | avg | max | last |
|---|---|---|---|---|
| base | -5 | 0.0344 | 5 | -1.445 |
| hf | -2 | 0.0655 | 2 | -0.416 |
| result | -6.938 | 0.0999 | 6.94 | -1.861 |

*Figure 15 Waveform 1 with base, harmonic and result*




| Name | min | avg | max | last |
|---|---|---|---|---|
| base | -20 | 0.137 | 20 | -5.78 |
| hf | -2 | 0.058 | 2 | -1.989 |
| result | -22 | 0.195 | 22 | -7.769 |

*Figure 16 Waveform 2*




| Name | min | avg | max | last |
|---|---|---|---|---|
| combiload | -28.338 | 0.295 | 28.3 | -9.63 |
| load1 | -6.938 | 0.0999 | 6.94 | -1.861 |
| load2 | -22 | 0.195 | 22 | -7.769 |

*Figure 17 Combined waveforms*

Unfortunately, the data sent to storage is just a snapshot of short interval summary values –with the sensor performing at 1kHz, Data Storage receives data at 1/2 Hz. In addition to aliasing effects, this also makes it too probable to receive readings of two signals with a shared harmonic frequency in such a manner that a phase shift obscures the harmonics. 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 contains, 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 attempts 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 $P_i$ is average apparent power load i and $HF_i$ is average harmonic frequency ratio of said load. 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 in a negative HF component for a new load , 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 are significant components HF3, HF5 and HF7.

Combining these observations, we formulate a rule for detecting a new load:

- If relative disruptive and reactive power goes down (or stays at near zero), the load is resistive
- Check each harmonic frequency for the following:
    - If there is a relative increase in harmonic content of frequency x, flag HFx as having increased.
    - If the relative value stayed the same, flag as HFx not having increased
    - If the relative value dropped more than the 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, the load was motor inductive, otherwise electric.

As explained in chapter 5.4, currently steady state detection is based on standard deviation, so very small loads compared to current overall consumption cannot be detected with this system – they would not trigger a new state. Unfortunately, they do have a small impact on harmonics, but fortunately it is proportional to their power, so in usual cases their effect on the average will also be small. Further, 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 in this analysis does not produce as good results as full fingerprinting. For example, some fingerprints categorized as resistive do contain some harmonics - particularly dishwashers and washing machines in heating cycle, when the machines perform other simultaneous tasks. 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

Changes in the sensing infrastructure since D3.7 are covered briefly here.

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

The initial version of the 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 described in D3.6, but has added harmonic currents information for fingerprinting.

As the first version of the Ambient interface was found to be too intrusive, two more versions were designed. Second version of Ambient Interface is basically the same design, but supports a more configurable protocol interface (detailed in D.6) and, and

the Wattlite Twist device is based on wholly new ideas.See D4.5 for further information on Wattlite Twist.

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

# 7 Results

## 7.1 Advances in the state of the art

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

1. The ability to process 1 Hz level data from multiple sources without a need for a custom built system. As an example of other EU FP7 project, DEHEMS (www.dehems.eu) tackled 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 details are not available, but the BeAware system was already used to read similar quantities of minute based measurements at the beginning of the project , so pushing the limit to 1/s for each measurement node gives us a possibility for load fingerprinting and more detailed advice.

2. Load fingerprinting based on measurement data was demonstrated both in the laboratory and in pilot sites. While the current solution did not allow full coverage of a household with a single sensor, real time fingerprinting in the non-intrusive load modelling (NIALMS) system was shown feasible and some preliminary solutions for multistate and disaggregated analysis where developed.

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

Multistate mode/type detection had almost a 100% hit rate on dishwasher and such dishwasher modes that exhibited very steady operational cycles, whereas with washing machine type loads the hit rate was roughly 80%. Mode detection within recognized washing machine category was 60%; thesystem had problems with false positives. Particularly the higher temperature washing modes differ mainly in resistive cycle time whereas other cycles are same. Also, the amount of clothing in a washing machine affects the states slightly; this is particularly evident when a washer starts its spinning cycle – heavier loads do not reach a steady state.

Single mode devices were tested against trial 2 setups with sensors that read only one device, whereas multistate detection was only tested on three internal pilot installation

devices with a total of five fingerprinted unique states, remaining states used for testing against false positives.

It must be noted that personal computers proved to be very hard fingerprint, as their fingerprint changes so much with usage pattern - how much CPU power is used, how much GPU power is used, how high the fans are running, etc. - they can basically fluctuate among several states depending on how the computer is used.

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

## 7.2 Challenges encountered

- The amount of data: the initial challenge for wp3 was the immense amount of data available. A single BeAware equipped household produces almost 130 million measurements annually. Handling this needed some redesigns, but fortunately these items were identified in the first prototype trials conducted withinthe project and did not impact actual customer trials
- Sensor instability: the first version of the 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. Due to this, a physical filter had to be attached to sensors measuring refridgerators, 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 the sensor, THD calculation was needlessly complicated as is attempted to filter noise out of measurements. With the HF3-HF7 solution, this filtering would not have been necessary.
- Data format: the whole field is unfortunately plagued with the lack of de facto standards comparable to e.g. SNMP on the telecom side. In Italy, COMSEL is seen as the most probable upcoming standard, whereas in Finland the industry has lobbied for wireless m-bus, and several commercial players such as Plugwise are using their own standards over ZigBee. As BeAware transfers relatively high amounts of data, it needed a 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 and the algorithms coded into the system would remain relatively stable in customer trials. It was quickly noticed that in fine-tuning the EnergyLife game, algorithms needed to be both parameterized and quickly replaceable. Thus the BABUP protocol now provides the option of parametrizable content.

- BeAware servers are physically located in Italy and Finland, with measurement sites also in Sweden. This means that for a family in Sweden, data is first transmitted to Finland to Data Storage, from there to the Service Layer in Italy, and then back to Sweden for user presentation. Transmission delays between data centers accumulates and thus the responsiveness of the system is degraded. In a commercial solution, all servers should reside close to each other, with at least some geographical mirroring available.
- The system was originally designed so that each measurement site would have a public base station IP address for ease of upgrading and debugging. Unfortunately this was not possible in Italy, so upgrading software required manual intervention. In a production system, this should be replaced by an automated update process that periodically checks for updates.
- Steady state analysis, the cornerstone of our fingerprinting solution, 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, such as 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 that can handle both large amounts of real time data and infrequent large batch data transfers. Local Bus to wide area network converters are expensive and add a new layer of potential faults to the system. Bypassing such added complications with mains meters that are able to communicate directly to the WAN over a common standardized protocol would be the technically best solution. SNMP over IPv6 is already a such standard in telecom, and would probably meet the requirements of power meter communications, as well. Mains meters should have an interchangeable communications module for at least Ethernet, GPRS/3g and PLC, with the protocol being the same. A mains meter could also act as the local bus gateway if needed, limiting required hardware to a 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. Too much distortion in power causes degradation in both equipment and causes transfer losses, so it very undesirable. Meters with just the basic capability of THD+noise analysis will be needed, at a much lower cost than currently available.

BeAware trials proved that a household with multiple measurements per second and the analysis of that stream of data is easily feasible with modern equipment, even while

power companies are currently hard pressed to provide even the previous hour's information to their customers. Making real time data available allows engaging the users and providing them with timely information, such as immediate advice generated by fingerprinting and an advice system as in BeAware. A similar system forwater and gas pipes would allow immediate detection of leaks, which at least in Finland amount for tens of percents of loss to water companies and can cause major damages to private owners if not detected early enough.

BeAware will open the algorithm, fingerprinting, and multistate device chain libraries to the public, allowing other projects to further research this field. Very little added development of the load disaggregation system will allow metering a household with three meters connected to mains (instead of meter at each socket), achieving much cheaper installation costs while retaining the ability to separate at least mains loads.