Deliverable 3.2

Project Title	Next-Generation Hybrid Broadcast Broadband
Project Acronym	HBB-NEXT
Call Identifier	FP7-ICT-2011-7
Starting Date	01.10.2011
End Date	31.03.2014
Contract no.	287848
Deliverable no.	3.2
Deliverable Name	Design and Protocol (High Level Architecture): User ID, Profile, Application Reputation Framework
Work package	3
Nature	Report
Dissemination	Public
Author	Félix Gómez Mármol, Ginés Dólera Tormo (NEC), Mark Gülbahar (IRT), Sebastian Schumann (ST), Gregor Rozinaj, Ivan Minárik (STUBA)
Contributors	Juraj Kačur, Matej Féder, Jozef Bán, Marek Vančo, Ondrej Lábaj, Alexandra Posoldova (STUBA)
Due Date	30.09.12
Actual Delivery Date	05.10.12

Table of Contents

Executive Summary **3**
1. Introduction **5**
2. Multimodal User Identification / Authentication, Multilevel Authorization **6**
2.1. Speaker Identification 6
2.1.1. Data Model 6
2.1.2. Design 7
2.1.3. Interface 8
2.1.4. Conclusion 9
2.2. Face Recognition 10
2.2.1. System Design 10
2.2.2. Internal Implementation 12
2.3. 3D Face Recognition 14
2.3.1. Algorithm 14
2.3.2. Conclusion 15
2.4. Section Conclusion 15
3. Trust and Reputation Module **16**
3.1. Data model 16
3.1.1. Reputation bundle 17
3.1.2. Reputation 18
3.1.3. Subject 20
3.1.4. Score 21
3.1.5. Date 21
3.1.6. Feedback bundle 22
3.1.7. Feedback 23
3.1.8. Issuer 24
3.1.9. Comment 25
3.1.10. Preference 26
3.1.11. Device capabilities 26
3.1.12. System conditions 28
3.1.13. Example 29
3.2. Design 33
3.2.1. Device capabilities & System conditions 34
3.2.2. User's profile 34
3.2.3. Feedback collector 34
3.2.4. Feedback storage 34
3.2.5. Reputation computation engine 34
3.2.6. Reputation scores storage 35
3.2.7. Reputation-based decision making engine 35
3.2.8. Reputation information visualizer 35
3.3. Interface 35
3.3.1. Internal Implementation 35
3.3.2. External interfaces 45
3.3.3. U.027 – Download app from app-store 51
3.3.4. U.031 – Mark app as trusted 53
3.3.5. U.032 – Using user identity and security mechanisms 55
4. Identity Management Module **57**
4.1. Data model 57

4.1.1. User 57
4.1.2. Device 59
4.1.3. Context 60
4.2. Design 62
4.2.1. Database implementation 62
4.3. Interface 65
4.3.1. Internal Implementation 65
4.3.2. API 65
4.4. Sequence Diagrams 71
5. Profile Management Module 74
5.1. Data model 74
5.1.1. User Profile 74
5.1.2. Service Profile 75
5.2. Design 75
5.3. Interface 76
5.4. Sequence Diagrams 78
6. Security Manager 79
6.1. Scope 79
6.1.1. Solution Overview 82
6.1.2. Management 83
6.1.3. Hardware architecture 83
6.1.4. Authorization model 83
6.1.5. PKI architecture 84
6.1.6. Use case 87
6.1.7. Sequence Diagram 89
6.1.8. API description 91
7. Conclusion 94
8. References 95
9. Abbreviations 96
10. Annex A 97

Executive Summary

This document is the second deliverable of WP3 of the HBB-NEXT project. It presents designs and first prototype implementations related to personalisation of next-generation hybrid-broadcast-broadband television. It provides a current (September 2012) view of the status of the designs, prototypes, integrations and demonstrations towards a central use case. It also includes an outlook to future work.

Use case: The principal use case of WP3 is described in following scenario.

- Generic app store
- User is entering the room, basic identification recognizes him/her
- Personalized app store will appear
- User tries to install/buy/validate(feedback) the app
- According user multilevel authentication and behaviour user is allowed/not allowed to do that

Design: A preliminary system design is presented, based on the following components:

- Multimodal interface
- Identity and Security Manager
- Reputation Framework

Prototypes: The following initial prototypes have been realised: Face Recognition (STUBA), Gesture Recognition (STUBA), Speaker identification (STUBA), Integration System (STUBA), Application reputation system (NEC), Early prototype for cloud offloading (NEC). The first four prototypes are a part of multimodal interface made by STUBA, last 2 applications from NEC represents reputation framework and cloud offloading.

Integration: Within WP3 we have presented the first successful integration of the two following components: gesture recognition and remote control of the TV system using gestures.

Demonstration: Demonstration of face recognition, speaker identification, as well as reputation framework has been presented.

Future work: Among other the following work is planned for WP3 for the next year: improving the multi-modal interface, combining face recognition with voice recognition and enabling the identification of multilevel user identification; 3D face recognition and iris recognition. Improving of the reputation framework should result in a possible integration of all WP3 functionalities into one integrated demo.

1. Introduction

This deliverable provides first designs and protocols for the following components:

- Multimodal user identification
- Security manager
- Reputation framework.

Multimodal interface has been addressed by speaker identification and face recognition. The modules are designed to cooperate together and to demonstrate the integration on a simple demo. A user enters the room, the system will recognize the user, the user open the AppStore application and the system allows him to choose, open, buy and install a desired application. For each activity or operation of the user, the system may ask multilevel authentication based on secure identification with satisfactory validation and security.

The structure of this document is as follows. The second chapter is devoted to multimodal interface and research areas, in which the project team implement applications for user identification. The third chapter describes trust and reputation framework. Identity management module, Security management module and Profile management module described in following 3 chapters together with multimodal interface module offer the background for secure and personalised functionality of the system.

2. Multimodal User Identification / Authentication, Multilevel Authorization

Multimodal User Identification is one of the key features of the HBB-NEXT project aiming at effortless utilization of the system's features while assuring that the system only performs commands which are properly authorized. The speaker identification tends to provide basic identification of the possible users located in the system installation area. The basic level would be suitable for not-so-crucial identification tasks, such as loading personal profile. The face detection approach aims to provide more reliable user identification based on users' faces which contain far more characteristics that can be parameterised in comparison to the voice identification approach. Additionally, the 3D face recognition further extends the possibilities of feature extraction in order to more precisely identify particular persons and can be thus used for the highest level authentication (and authorisation) for the most demanding applications (i.e. bank account login, etc.).

2.1. Speaker Identification

The aim of a speaker identification system is to decide the identity of a speaker upon an utterance, regardless of what he or she said. As any speaker identification system consists of two main parts, namely: feature extraction and classification method, the designer has to select proper methods and their modifications for a given application which may differ depending on type and setting of particular task. For the purpose of our application we have chosen the k- nearest neighbour method (KNN) for classification and Mell frequency cepstral coefficients (MFCC) for speech parameterizations and their modifications. For the system functional overview see Figure 1 that will be explained in the Interface section.

2.1.1. Data Model

In order to perform speaker identification using the selected KNN method (weighted KNN) and MFCC, following internal data structures had to be introduced:

- Buffer of speech samples, which is a simple vector containing sequence of short numbers. As there are both recognition and training phases which are of the different lengths two such buffers of different sizes are used.

- Buffer of parameterised speech, i.e. vectors of MFCC. Theoretically it is a matrix where rows are different speech frames and columns are individual features (MFCC elements). However it is implemented as a vector. 2 buffers are used one for tested speech and the second one will store all vectors in the database (training MFCC vectors appended by indexes of speakers)
- Vectors for found neighbours (their indexes)
- Vectors for found neighbours (their distances)
- Vector of winners (their indexes)
- Vector of winners (their confidences)
- Structure for storing recognition settings that contains 10 important items controlling the recognition process, like: type of local distance, umber of neighbours, training and recognition times etc.
- Structure for storing feature extraction settings that contains 12 important items controlling the speech extraction process, like: frequency limits, frame length and its shift etc.
- Array of strings storing indexes of speakers and their names

2.1.2. Design

The application is a standalone console Windows application and as it should be fast and process lot of data it is coded in C/ C++ language. As it uses OS and hardware specific system calls (reading sounds from microphones via WINAPI 32) it is not portable to other OS systems unlike Windows 98 and higher versions.

The basic facilities of the applications are:

1. Recording a new speaker, transforming the wave samples into MFCC features according to configuration variables and storing both name and feature vectors in the database
2. Identifying unknown speaker (recording voice, producing MFCC features, finding nearest neighbours and taking decision) according to configuration variables.

3. Listing recorded users in the database.

4. Removing recorded users from the database

5. Change its functionality via properly setting decision and parameterisation configuration files.

2.1.3. Interface

Internally the application can be divided into several functionally distinctive and setting independent blocks which are: wave form reading, feature extraction (MFCC), KNN, final decision taking, I/O functions, data storage and user API. Their mutual cooperation is shown in Figure 1.

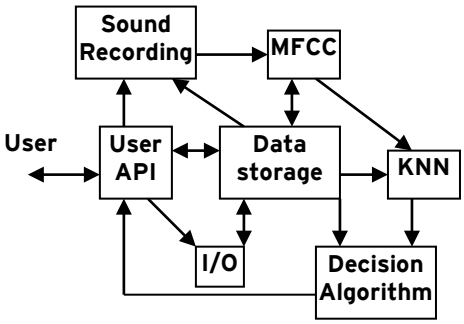

Figure 1: A functional block scheme of single user identification.

The API is divided (by functionality and access not by security) between designer/system administrator API and common user interface. The designer/ system administrator API is done via 2 configuration files (*config_knn.txt* –controlling recognition process and *config_param.txt* that is involved in the speech extraction, for examples of the configuration files please see Annex A) where almost all settings controlling the behaviour of an application can be set. It should be noted that in order the application function properly the parameters in both configuration files must be set by professionals. The user API is done via a console application menu whose outlook is as follows:

Speaker Identification Menu:

- Add and record new user press 1
- Identify unknown speaker press 2
- Display users in database press 3

- Remove a user from database press 4
- Exit press 5

Choice:

The operation flows are as follows. After choosing a desired option from the menu a particular code is invoked. When adding a new user first a name is require to be entered that must be unique. After that a speaker is prompted to speak for pre-set time given by a configurable parameter. There is also a minimal training time which must be met prior to saving new features. When an unknown individual is to be recognized, the recognition data are first updated and stored in operational memory. Then he or she is prompted to speak for a given time period (also set in the configuration file). If minimal length is not met warning is displayed. Then speech is transformed into MFCC and is passed to KNN algorithm which based on the stored training vectors chooses nearest neighbours and determines particular winners with their scores. Finally, all winners are considered in the concluding decision process whose output is provided to the user via a console window. The remaining options (listing and deleting users) are self-explanatory.

2.1.4. Conclusion

In the next step all implemented methods would be optimized (all free parameters and features will be set) in order to achieve the highest accuracy and robustness. It is possible to implement and test parametric decision taking algorithms like Gaussian mixture model in the case of lack of training vectors. Also additional signal processing techniques may be implemented and tested, like: cepstral mean subtraction, feature normalization, feature warping and mapping (to suppress session variability), etc. It is also possible to implement higher level features like pitch period to increase the accuracy and robustness.

2.2. Face Recognition

In this chapter we present a design and first prototype implementation of our face recognition module to application in the HBB-NEXT project. HBB-NEXT project defines a list of requirements for single local user identification based on a human face [1] that the systems must/should/may implement:

- The system shall identify user inside the room by his face, if he belongs to the local users group.
- The system should be able to recognize user and compare him to locally stored user profiles without internet access.
- The system may identify user by his face inside the room unknown to the local system (not listed in the local users group).
- System may recognize user based on face recognition even in dark.

HBB-NEXT requirements and methods which are able to work well in real-world conditions were included to our face recognition module.

2.2.1. System Design

In this section we describe architecture of the proposed system and workflow used in the HBB-NEXT project. Since the face recognition task requires processing of the input from a camera, we decided to use a pipeline model where each frame captured by the camera is processed by set of modules. These modules contains an implementation of the chosen face recognition method and, together with the camera, operates in a loop.

The main recognition process (the system loop) consists of following sub processes:

- ***Image acquisition*** - reads an image from the camera, converts it to the system format and pass it to the system pipeline.
- ***Face localisation*** - localizes the faces in the image on the image and associate found coordinates with the image. For the purpose of this project we use two different localisation implementation:, depending on the camera which is used:

- *OpenCV Cascade detector* - this is used with ordinary cameras with standardized interface (e. g. web cameras compatible with OpenCV library)
- *Kinect Toolkit Face detector* - this detector is used with the Kinect Camera[1]
  - ***Pre-processing of localised faces*** - copies regions of the image containing faces into new samples, converts the regions into the appropriate resolution and the appropriate colour format.

- ***Feature extraction*** - Extracts features from preprocessed faces. For the purpose of this project we decided to use LBP histograms as features. LBP histograms are considered as one of the best features for recognizing faces ([2], [3]) even when only a limited number of samples is available [4] and can be easily computed in the real time [5].
- ***Classification of faces*** - For the classification of features extracted from faces we propose to use two methods depending on the number training images and number of identities which is to be used within the system:
  - Support Vector Machines - is used when only relatively small number of identities is considered in the system. Main disadvantage of this method is the time-consuming training of the model when large number of samples is used.
  - Nearest neighbour distance matching (with the use of Chi-square distance) - this algorithm can be easily parallelised and used in distributed system. The training is done simply by inserting features into the database.
- ***Face tracking*** - In our implementation we localize only frontal faces in the image because the vast majority of face recognition methods is reliable only with use of frontal face images. Once the face has been recognized, it is tracked, what

[1] Face tracking SDK is a part of Kinect for Windows Developer Toolkit v1.5 - http://www.microsoft.com/en-us/download/details.aspx?id=29865

significantly saves computational resources and can follow the subject even after changes in pose. In our system we use TLD tracker proposed by Zdenek Kalal [6].

- ***Temporal filtering*** - localisation of faces is not always reliable due to changes in environment, poses or imperfection of cameras. To prevent the system from triggering events in case of every detection or tracking failure we propose a method for filtering the most frequent types of failure. For instance to filter out short focus-losing windows in tracking.

- ***Event trigger*** - this module sends a notification of user state changes to the rest of HBB-NEXT applications.

The face recognition module sends XML output to a server. The XML describes the identity of subject (person), probability of correct classification, presence of a person in the room (present, idle, quit) and it contains all the identities that were trained. The example of using the XML structure is shown in Figure 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<application>FACE_RECOGNITION</application>
<users>
      <user>
            <name>John</name>
            <probability>10</probability>
            <presence>PRESENT</presence>
      </user>
      <user>
            <name>George</name>
            <probability>75</probability>
            <presence>IDLE</presence>
      </user>
      <user>
            <name>Bill</name>
            <probability>60</probability>
            <presence>QUIT</presence>
      </user>
</users>
```

Figure 2: The example of XML structure

2.2.2. Internal Implementation

The core of the face recognition system is written in C++ with using OpenCV library. We decided to use an open source project Biosandbox developed at STUBA. In this project a large number of required algorithms was already implemented in form of dynamic modules.

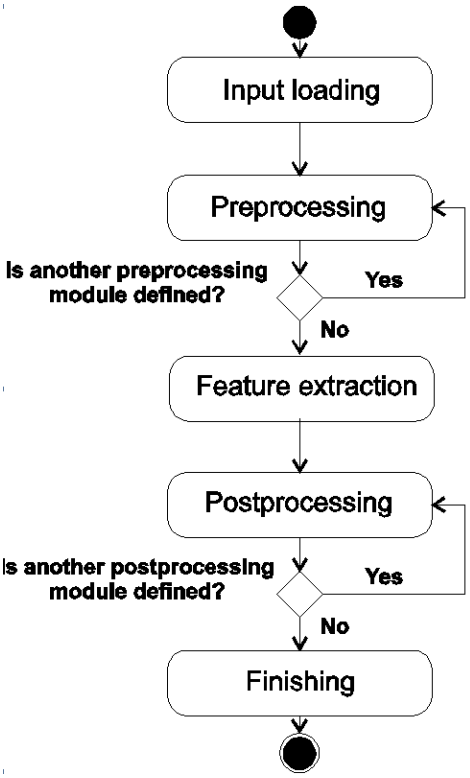

Figure 3: Schematic illustration of processes sequence during system execution.

The basic concept of the Biosandbox system is shown in Figure 3. The composition of modules is defined in an XML configuration file that can be created in a graphical editor that is also part of this project. For the purpose of the project we developed 3 additional modules:

- Kinect Input module - reads images captured by Kinect camera.
- Tracking module based on TLD [6] - once a face is localized in the image, this modules track its position even when the pose of the subject has changed.
- Temporal filtering module combined with event trigger - checks presence of the subject in the scene with considering detection results and previous results in the time.

We would like to highlight that other existing modules has been also modified to meet all requirements of the project.

2.3. 3D Face Recognition

HBB NEXT project is an idea where is very important to keep trust because of users will use many applications which will be used for various purposes. For example: logging to trust application, verification for paying or buying products, etc. We have included to HBB 3D face recognition for High level verification that could increase the security of some used applications.

2.3.1. Algorithm

The software is programmed in .NET platform and it uses Kinect libraries for communication with Kinect sensor. The main principle is to find the nearest point of face from sensor that is a nose tip. Firstly a person has to have a face in active zone that is from 400mm to 850mm from sensor. If the face is in inactive zone (red zone), won't be measured. In the active zone will be detected the nearest point (nose tip). The nose tip will be our reference value from which other pixels of face are measured. Next we have to set a distance of measured zone that usually is about 50mm (white zone). Then we get a set of facial points that are represented by relative distances. The relative distances are processed 30 times per second because the position of measured head is continually changed.

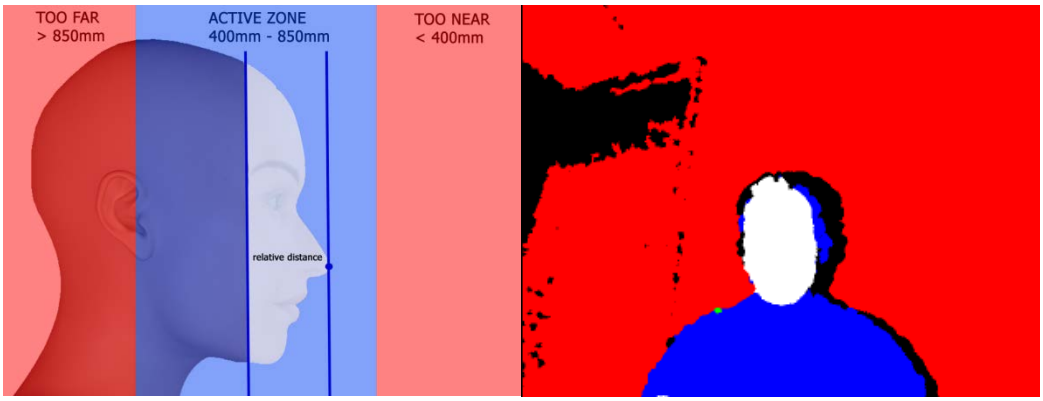

Figure 4: a) Principle of base measuring and b) scanned face from distance 900mm.

Algorithm in steps:

1. Obtain raw data from Kinect sensor – The sensor obtains data and sends as 16 bit number. This number contains 13 bits for depth data and 3 bits for player segmentation data.

2. Calculate distances from raw data – In this step it is required pick 13 bits of distance and compute distance in millimetres.

3. Find the nearest point (nose tip) – This algorithm finds the closest area of points, because a nose has the smallest distance from camera among all points of a face.

4. Set width of the measured zone – It is still needed to set an active zone of face. It is a distance from nose tip to the deepest point of face area.

5. Get data from zone – Now it possible discern face points from other data. This data can be normalized and saved to database.

6. Normalize data and convert their to RGB picture – Normalization of face data according some parameters.

2.3.2. Conclusion

In the future we want to normalize data for saving to a database and create some parameters for comparing in the database. Firstly, we want to try to use some principles which were described in Deliverable 3.1 and add some of ours.

2.4. Section Conclusion

As it was mentioned in the previous sections the presented solutions only exist in the form of early demos. All of them, however, are already capable of interaction with the rest of the multimodal system designed by STUBA, even though there is still room for improvements, particular to each modality. Further work in the field of multimodal user identification and authorisation will include, apart from improving the algorithms themselves, gradual integration of the modalities' outputs to enhance the quality of proper identification. Additionally, a goal exists to create multi-stage user identification, consistent with the project's Work Package 5.

3. Trust and Reputation Module

As stated in D3.1, one of the challenges of the HBB-NEXT project is to offer trustworthy HBB services and the rise of users' trust when interacting with the HBB-NEXT platform. WP3 tackles this challenge by developing a trust and reputation management framework for service providers and applications developers. The resulting solution will also be intended to guarantee accurate handling of security threats as well as isolation of potential malicious services, e.g. avoiding fraudulent or untrustworthy applications, or determining the trustworthiness of an entity requesting some users' profile attributes to deliver a service.

The content of this chapter is organized as follows: Section 3.1 will present the set of data structures, represented as XML schemas, used to transfer reputation information between the internal modules of the enabler and between this enabler and external ones. The aforementioned internal modules will be presented and described in Section 3.2, showing their functionality as well as their interactions in the whole picture of the enabler architecture. In turn, Section 3.3 will explain the interfaces required to interact and make use of both each internal module, as well as the trust and reputation management enabler itself. Finally, Section 3.4 will show some sequence diagrams depicting the main operations to be performed through this trust and reputation management enabler. The actual algorithms to compute the final reputation scores, as well as many other implementation details of the trust and reputation enabler will be shown and explained in D3.3 "Design and Protocol: Intermediate User ID, Profile, Application Reputation Framework".

3.1. Data model

In order to ease the communication and transfer of reputation information, both within the internal modules constituting the trust and reputation enabler and towards external modules, we have adopted a set of data structures inspired in the specification done by OASIS ORMS (Open Reputation Management Systems [7]).

3.1.1. Reputation bundle

The <ReputationBundle> element can contain one or more <Reputation> elements to optionally make a group of reputation instances. Within the context of HBB-NEXT, this data structure can be used, for instance, by the AppStore to retrieve the <Reputation> elements of a bunch of applications, all in once.

The following schema fragment defines the <ReputationBundle> element and its ReputationBundleType complex type:

```
<element name="ReputationBundle" type="ReputationBundleType"/>
<complexType name="ReputationBundleType">
<sequence>
<element ref="Reputation" minOccurs="1" maxOccurs="unbounded"/>
</sequence>
<attribute name="id" type="String" use="optional"/>
</complexType>
```

The <ReputationBundle> element will be used, within the context of HBB-NEXT, to represent the reputation score given to a set of HBB-NEXT applications.

One example of use of the <ReputationBundle> element would be the following one:

```
<ReputationBundle id=”rpb001”>
<Reputation id=”...”>
    ...
</Reputation>
<Reputation id=”...”>
    ...
</Reputation>
...
</ReputationBundle>
```

3.1.2. Reputation

The following schema fragment defines the <Reputation> element and its ReputationType complex type:

```
<element name="Reputation" type="ReputationType"/>
<complexType name="ReputationType">
<sequence>
<element ref="Subject" minOccurs="1"/>
<element ref="Score" minOccurs="1"/>
<element ref="Date" minOccurs="1"/>
<element ref="FeedbackBundle" use="optional"/>
</sequence>
<attribute name="id" type="anyURI" use="optional"/>
</complexType>
```

The <Reputation> element will be used, within the context of HBB-NEXT, to represent the reputation score given to a particular HBB-NEXT application. It might contain, optionally, a <FeedbackBundle> element containing the set of feedbacks provided by previous users regarding this specific HBB-NEXT application.

One example of use of the <Reputation> element would be the following one:

```
<Reputation id=”...”>
     <Subject>
          AngryPigeon
     </Subject>
     <Score>
          0.68</Score>
     <Date>
          2012-09-30T09:30:10+02:00
     </Date>
     <FeedbackBundle id=”...”>
     ...
     </FeedbackBundle>
</Reputation>
```

3.1.3. Subject

The <Subject> element contains a String value which identifies the entity evaluated by this document.

The following schema fragment defines the <Subject> element:

```
<element name="Subject" type="String"/>
```

The <Subject> element will be used, within the context of HBB-NEXT, to represent a particular HBB-NEXT application whose reputation needs to be computed or known. Moreover, it could be used as well to identify the developer or provider of a specific application requesting access to user's attributes. One example of use of the <Subject> element would be the following one:

```
<Subject>
    AngryPigeon
</Subject>
```

3.1.4. Score

The <Score> element contains a double value of a reputation score contained within a <Reputation> element.

The following schema fragment defines the <Score> element:

```
<element name="Score" type="double"/>
```

One example of use of the <Score> element would be the following one:

```
<Score>
    0.68
</Score>
```

3.1.5. Date

The <Date> element contains a time value which specifies the dates defined in the namespace of the <Context> element. The value MUST be expressed in UTC form and MUST NOT use fractional seconds.

The following schema fragment defines the <Date> element and its DateType complex type:

```
<element name="Date" type="DateType"/>
<complexType name="DateType">
<simpleContent>
<extension base="dateTime">
<attribute name="type" type="anyURI" use="required"/>
</extension>
</simpleContent>
</complexType>
```

One example of use of the <Date> element would be the following one:

```
<Date>
    2012-09-30T09:30:10+02:00
</Date>
```

3.1.6. Feedback bundle

The <FeedbackBundle> element can contain one or more <Feedback> elements to optionally make a group of feedback instances. This can be used, for instance, and within the context of HBB-NEXT, to retrieve all the feedbacks related to a concrete application in the AppStore.

The following schema fragment defines the <FeedbackBundle> element and its FeedbackBundleType complex type:

```
<element name="FeedbackBundle" type="FeedbackBundleType"/>
<complexType name="FeedbackBundleType">
<sequence>
<element ref="Subject" minOccurs="1"/>
<element ref="Feedback" minOccurs="1" maxOccurs="unbounded"/>
</sequence>
<attribute name="id" type="anyURI" use="optional"/>
</complexType>
```

The <FeedbackBundle> element will be used, within the context of HBB-NEXT, to represent the collection of feedbacks provided by previous users regarding a concrete HBB-NEXT application.

One example of use of the <FeedbackBundle> element would be the following one:

```
<FeedbackBundle id=”fbb001”>
     <Subject>
          AngryPigeon
     </Subject>
     <Feedback id=”fb001”>
     ...
     </Feedback>
     <Feedback id=”fb002”>
     ...
     </Feedback>
     ...
</FeedbackBundle>
```

3.1.7. Feedback

The following schema fragment defines the <Feedback> element and its FeedbackType complex type:

```
<element name="Feedback" type="FeedbackType"/>
<complexType name="FeedbackType">
<sequence>
<element ref="Issuer" minOccurs="1"/>
<element ref="Subject" minOccurs="0"/>
<element ref="Score" minOccurs="1"/>
<element ref="Date" minOccurs="1"/>
<element ref="Comment" use="optional"/>
</sequence>
<attribute name="id" type="anyURI" use="optional"/>
</complexType>
```

The <Feedback> element will be used, within the context of HBB-NEXT, to represent the feedback provided by a specific previous user (<Issuer>) regarding a concrete HBB-NEXT application (<Subject>).

One example of use of the <Feedback> element would be the following one:

```
<Feedback id=”fb001”>
     <Issuer id=”Alice”>
     ...
     </Issuer>
     <Subject>
          AngryPigeon </Subject>
     <Score>
          0.8</Score>
     <Date>
          2012-09-28T10:22:30+02:00</Date>
     <Comment>
          Nice application, easy to use </Comment>
</Feedback>
```

3.1.8. Issuer

The <Issuer> element contains a String value which identifies the evaluating entity.

The following schema fragment defines the <Issuer> element:

```
<element name="Issuer" type="IssuerType"/>
<complexType name="IssuerType">
<sequence>
<element ref="Preference" maxOccurs="unbounded" use="optional"/>
</sequence>
<attribute name="id" type="String"/>
</complexType>
```

The <Issuer> element will be used, within the context of HBB-NEXT, to represent the specific end-user providing a feedback regarding a concrete HBB-NEXT application. Moreover, this end-user will have some defined preferences used to compute a customized reputation score.

One example of use of the <Issuer> element would be the following one:

```
<Issuer id=”Alice”>
     <Preference name=”price”>
     ...
     </Preference>
     <Preference name=”usability”>
     ...
     </Preference>
     ...
</Issuer>
```

3.1.9. Comment

The <Comment> element contains a string value which represents the optional opinion message left by an end-user when providing feedback about a specific HBB-NEXT application.

The following schema fragment defines the <Comment> element:

```
<element name="Comment" type="string"/>
```

One example of use of the <Comment> element would be the following one:

```
<Comment>
     Nice application, easy to use
</Comment>
```

3.1.10. Preference

The <Preference> element contains a String value which identifies a concrete user's preference regarding the applications provided in the AppStore to determine how important this feature is to the end user.

The following schema fragment defines the <Preference> element:

```
<element name="Preference" type="PreferenceType"/>
<complexType name="PreferenceType">
<attribute name="name" type="string"/>
<sequence>
<element ref="Score"/>
</sequence>
</complexType>
```

One example of use of the <Preference> element would be the following one:

```
<Preference name=”price”>
     <Score>
          0.8
     </Score>
</Preference>
```

3.1.11. Device capabilities

The <DeviceCapabilities> element is used to represent some of the features of the device used to (potentially) determine: i) the concrete reputation computation engine to apply, ii) how the feedback is collected and iii) how the reputation information is rendered to the end-users (as we will see in detail in Section 3.2).

The following schema fragment defines the <DeviceCapabilities> element:

```
<element name="DeviceCapabilities" type="DeviceCapabilitiesType"/>
<complexType name="DeviceCapabilitiesType">
<attribute name="id" type="String"/>
<sequence>
<element name="DeviceFeature" type="DeviceFeatureType"/>
</sequence>
</complexType>
<complexType name="DeviceFeatureType">
```

```
<attribute name="name" type="String"/>
<attribute name="value" type="String"/>
</complexType>
```

One example of use of the `<DeviceCapabilities>` element would be the following one:

```
<DeviceCapabilities id=”dc001”>
     <DeviceFeature name=”screen-resolution” value=”800X600” />
     <DeviceFeature name=”user-input” value=”keyboard” />
     ...
</DeviceCapabilities>
```

3.1.12. System conditions

The <SystemConditions> element is used to represent some of the features of the environment used to (potentially) determine: i) the concrete reputation computation engine to apply, ii) how the feedback is collected and iii) how the reputation information is rendered to the end-users (as we will see in detail Section 3.2).

The following schema fragment defines the <SystemConditions> element:

```
<element name="SystemConditions" type="SystemConditionsType"/>
<complexType name="SystemConditionsType">
<attribute name="id" type="String"/>
<sequence>
<element name="SystemCondition" type="SystemConditionType"/>
</sequence>
</complexType>
<complexType name="SystemConditionType">
<attribute name="name" type="String"/>
<attribute name="value" type="String"/>
</complexType>
```

One example of use of the <SystemCondition> element would be the following one:

```
<SystemCondition id=”sc001”>
     <SystemCondition name=”bandwidth” value=”1MB” />
     <SystemCondition name=”cpu-usage” value=”63%”” />
     ...
</SystemCondition>
```

3.1.13. Example

Next we present a comprehensive example of use of the `<ReputationBundle>` element containing all the previously shown elements:

```
<ReputationBundle id=”rpb001”>
    <Reputation id=”rep001”>
        <Subject>
            AngryPigeon
        </Subject>
        <Score>
            0.68
        </Score>
        <Date type=”xs:dateTime”>
            2012-09-30T09:30:10+02:00
        </Date>
        <FeedbackBundle id=”fbb001”>
            <Feedback id=”fb001”>
                <Issuer id=”Alice”>
                    <Preference name=”price”>
                        <Score>
                            0.4
                        </Score>
                    </Preference>
                    <Preference name=”usability”>
                        <Score>
                            0.8
                        </Score>
                    </Preference>
                </Issuer>
                <Score>
                    0.8
                </Score>
                <Datetype=”xs:dateTime”>
                    2012-09-28T10:22:30+02:00
                </Date>
                <Comment>
```

```
                        Nice application, easy to use
                  </Comment>
            </Feedback>
            <Feedback id=”fb002”>
                  <Issuer id=”Bob”>
                        <Preference name=”price”>
                              <Score>
                                    0.6
                              </Score>
                        </Preference>
                        <Preference name=”usability”>
                              <Score>
                                    0.6
                              </Score>
                        </Preference>
                  </Issuer>
                  <Score>
                        0.4
                  </Score>
                  <Datetype=”xs:dateTime”>
                        2012-09-27T07:26:54+02:00
                  </Date>
                  <Comment>
                        It becomes hard from level 15
                  </Comment>
            </Feedback>
      </FeedbackBundle>
</Reputation>
<Reputation id=”rep002”>
      <Subject>
            freeMaps
      </Subject>
      <Score>
            0.8
      </Score>
      <Date type=”xs:dateTime”>
```

```
                2012-09-30T09:30:10+02:00
          </Date>
          <FeedbackBundle id=”fbb001”>
                <Feedback id=”fb011”>
                      <Issuer id=”Alice”>
                            <Preference name=”price”>
                                  <Score>
                                        0.4
                                  </Score>
                            </Preference>
                            <Preference name=”usability”>
                                  <Score>
                                        0.8
                                  </Score>
                            </Preference>
                      </Issuer>
                      <Score>
                            0.8
                      </Score>
                      <Datetype=”xs:dateTime”>
                            2012-09-28T10:22:30+02:00
                      </Date>
                      <Comment>
                            I really recommend this app.
                      </Comment>
                </Feedback>
          </FeedbackBundle>
     </Reputation>
</ReputationBundle>
```

In this example the <ReputationBundle> element contains, in turn, two <Reputation> elements. The first one contains the reputation of an application named AngryPigeon (within <Subject>). The reputation value is represented as a double value between 0 and 1, which in this case is 0.68 (specified in <Score>). This element also specifies the date where the reputation has been computed (<Date>).

In addition to that, the <Reputation> element contains a <FeedbackBundle> element, which includes set of <Feedback> elements. In this case, there are two <Feedback> elements. The first one describes the feedback which the user identified as Alice has given about AngryPigeon. It specifies the score she gives to the application (within the <Score> element) date when she provides the feedback (<Date> element), some additional comments that Alice provided (<Comment> element) and it also includes a set of parameters defining Alice's preferences (within the <Preference> element).

The <Preference> element describes the importance that a user gives to a certain parameter. In this example, there are defined two parameters, namely price and usability. The former has a relevance of 0.4 for Alice, on a scale between 0 and 1, while the latter has a relevance of 0.8 for Alice.

The second <Feedback> element related to the application AngryPigeon describes the feedback given by Bob. Bob scored the application with a 0.4. After taking into account these feedbacks the reputation computed to the application AngryPigeon is 0.68.

In a similar way, the other <Reputation> element describes the reputation of an application named freeMaps. This application has a score of 0.8 out of 1, and contains a <Feedback> element representing the feedback given by Alice to this application.

3.2. Design

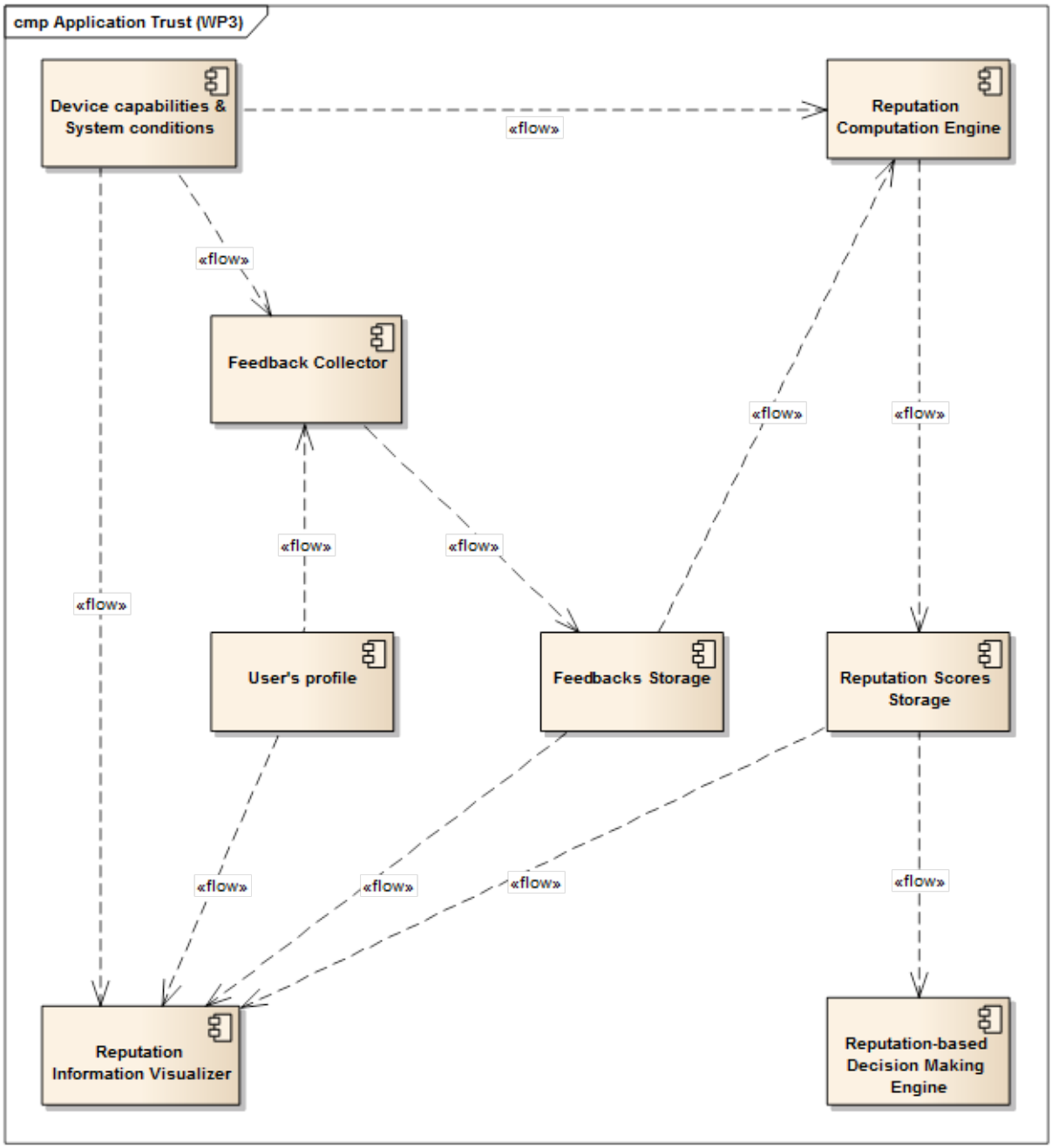

Figure 5: Trust and reputation internal modules interaction

As shown in Figure 5, the trust and reputation enabler consists of the following main internal modules (the description of the interfaces for each of these modules will be shown in Section 3.3):

3.2.1. Device capabilities & System conditions

This module will provide some information regarding the properties, capabilities and, in some cases, constraints owned by the device being used to interact with the trust and reputation framework. It also provides information about the current conditions of the environment which could mainly influence the behaviour of the framework. For instance, network or computer resources, number of users registered on the system could be provided to adapt the functionality of the enabler.

Both devices capabilities and system conditions might determine the way some operations are performed, like showing reputation information, requesting for feedback or calculating the reputation scores using one or another specific computation engine.

3.2.2. User's profile

The user's profile module does not actually belong to the trust and reputation enabler, but the latter is fed by some user's attributes from the former, in order to customize certain services or functionalities offered by the trust and reputation enabler.

3.2.3. Feedback collector

This module is responsible for gathering the end-users' recommendations/feedbacks regarding a particular consumed service (an app within the context of HBB-NEXT) in a user-friendly way, maybe depending also on the capabilities of the device in use or the system conditions for this purpose. It is also its duty to generate the feedback object that will be delivered to the feedback storage module.

3.2.4. Feedback storage

This module receives the feedbacks elements gathered by the feedback collector module and efficiently stores them in such a way that are easily retrievable when needed.

3.2.5. Reputation computation engine

This module probably constitutes the core of the trust and reputation enabler, since it is its responsibility to compute the actual reputation scores for a certain entity (an app in the context of HBB-NEXT), based on the feedbacks given by previous users and stored in the feedback storage module.

Furthermore, this module can decide whether to apply once concrete algorithm or another to compute the reputation scores, depending on the capabilities of the device in use and the system conditions, as well as other external system conditions. The final goal is to apply at each moment the most suitable reputation computation algorithm (the one obtaining most accurate results, while consuming the minimum resources). Finally, the computed reputation score is sent to the reputation scores storage module.

3.2.6. Reputation scores storage

This module receives reputation score elements from the reputation computation engine and stores them in an efficient way, so that they can be later accessed by other internal modules which use such scores as an input.

3.2.7. Reputation-based decision making engine

In order to enable the use case U.032 – "Using user identity and security mechanisms", where an application requires to access certain user's attributes in order to be actually installed and executed (see D2.1 [11]), there is the need to include this module which will be responsible for automatically deciding whether to allow the requested exchange of user's attributes with an app, based on the reputation score of the latter.

3.2.8. Reputation information visualizer

This module will take several inputs from a number of other internal modules, such as device capabilities and system conditions, user's profile or reputation scores storage. Its main responsibility consists of rendering the reputation scores and associated feedbacks for each app in the app-store in a user-friendly way.

3.3. Interface

3.3.1. Internal Implementation

This section shows the description of the main operations provided by each one of the internal modules of the trust and reputation management enabler depicted in Figure 5.

3.3.1.1. Device capabilities& System conditions

The Device capabilities & System conditions internal module offers two operations, namely, `getDeviceCapabilities()`, whose description can be found in Table 1 and `getSystemConditions()`, whose description is depicted in Table 2.

Operation name	`getDeviceCapabilities()`	
Operation description	Retrieves the specified features from the device in use	
Parameter name	**Parameter type**	**Parameter description**
`deviceFeatureName`	`DeviceFeatureType.name`	Name of the feature of the device to be retrieved
⋮	⋮	⋮
`deviceFeatureName`	`DeviceFeatureType.name`	Name of the feature of the device to be retrieved
Return type	**Return description**	
`<DeviceCapabilities>`	Returns a `<DeviceCapabilities>`element containing all the specified features that could be retrieved from the device. In case a specified feature could not be retrieved, the `<DeviceCapabilities>`element will just not contain it	

Table 1: Operation: getDeviceCapabilities()

Operation name	`getSystemConditions()`	
Operation description	Retrieves the current system conditions	
Parameter name	**Parameter type**	**Parameter description**
`systemConditionName`	`SystemConditionType.name`	Name of the system condition to be retrieved
⋮	⋮	⋮
`systemConditionName`	`SystemConditionType.name`	Name of the system condition to be retrieved
Return type	**Return description**	
`<SystemConditions>`	Returns a `<SystemConditions>`element containing all the specified conditions that define the current environment. In case a specified condition could not be retrieved, the <SystemConditions>element will just not contain it	

Table 2: Operation: getSystemConditions()

3.3.1.2. User's profile

Table 3 shows the only operation provided by the User's profile module: `getIssuerPreferences()`, which is responsible for retrieving the preferences of a given issuer, regarding the applications he/she consumes.

Operation name	`getIssuerPreferences()`	
Operation description	Retrieves the user preferences of a specified issuer	
Parameter name	**Parameter type**	**Parameter description**
`issuerId`	`IssuerType.id`	Identifier of the issuer (user) whose preferences need to be retrieved
Return type	**Return description**	
`<Issuer>`	Returns a `<Issuer>`element containing all the `<Preference>` elements representing, in turn, the user preferences of the specified issuer	

Table 3: Operation: getIssuerPreferences()

3.3.1.3. Feedback collector

The main operation offered by the Feedback collector module, named `collectFeedback()`, has been described in Table 4.

Operation name	`collectFeedback()`	
Operation description	Collects feedback from a specific issuer (user) about a concrete subject (HBB-NEXT application) taking into account the capabilities of the device in use	
Parameter name	**Parameter type**	**Parameter description**
`issuer`	`<Issuer>`	Issuer (user) providing the feedback
`subject`	`<Subject>`	Subject (HBB-NEXT application) to be evaluated by the issuer
`deviceCapabilities`	`<DeviceCapabilities>`	Capabilities of the device used to provide the feedback. Used to determine how to actually collect the feedback from the issuer
`systemConditions`	`<SystemConditions>`	Current system conditions. Used to tune how to collect the feedback from the issuer
Return type	**Return description**	

<Feedback>	Returns a <Feedback>element containing the score given by the specified issuer about the given subject, as well as a timestamp and, optionally, some opinion message (comment)

Table 4. Operation: collectFeedback()

3.3.1.4. Feedback storage

The Feedback storage module provides a couple of main operations in order to make use of it. The first one, storeFeedback(), has been described in Table 5, while the second one, called retrieveFeedbackBundle(), can be found in Table 6.

Operation name	storeFeedback()	
Operation description	Stores a given feedback within the Feedbacks storage module in such a way that it will be easily retrievable when needed afterwards	
Parameter name	**Parameter type**	**Parameter description**
feedback	<Feedback>	Feedback to be stored within the Feedbacks storage module
Return type	**Return description**	
boolean	Returns 'true' if the storage of the provided <Feedback> element was successful and 'false', otherwise	

Table 5. Operation: storeFeedback()

Operation name	retrieveFeedbackBundle()	
Operation description	Retrieves all the feedbacks provided to and stored in the Feedbacks Storage module, regarding a specified subject (HBB-NEXT application)	
Parameter name	**Parameter type**	**Parameter description**
subject	<Subject>	Subject (HBB-NEXT application) whose feedbacks are to be retrieved
Return type	**Return description**	
<FeedbackBundle>	Returns a <FeedbackBundle> element containing the <Feedback> elements which, in turn, represent each of the feedbacks provided regarding the specified subject	

Table 6. Operation: retrieveFeedbackBundle()

3.3.1.5. Reputation computation engine

The main operation of the Reputation computation engine module, which in turn is one of the key operations of the trust and reputation management enabler, has been described in Table 7 and it is called `computeReputation()`.

Operation name	`computeReputation()`	
Operation description	Computes a reputation score for a specified subject (HBB-NEXT application) based on the set of feedbacks provided by previous users regarding that specific subject. The concrete reputation computation engine applied will be selected dynamically depending on the capabilities of the device used to perform the computation of the reputation score	
Parameter name	**Parameter type**	**Parameter description**
`subject`	`<Subject>`	Subject (HBB-NEXT application) whose reputation is to be computed
`feedbacks`	`<FeedbackBundle>`	Set of feedbacks that previous users provided regarding the specified subject
`deviceCapabilities`	`<DeviceCapabilities>`	Capabilities of the device used to compute the reputation score of the specified subject
`systemConditions`	`<SystemConditions>`	Current system conditions, used to select the most suitable reputation computation engine
Return type	**Return description**	
`<Reputation>`	Returns a `<Reputation>` element containing the computed reputation score for the specified subject, as well as a timestamp and, optionally, the set of feedbacks provided by previous users regarding that specific subject	

Table 7: Operation: computeReputation()

3.3.1.6. Reputation scores storage

This internal module, Reputation scores storage, offers three main operations to the rest of internal modules within the trust and reputation management enabler, namely: i) `storeReputation()`, described in Table 8, ii) `retrieveReputationBundle()`, shown in Table 9, and iii) `retrieveReputation()`, explained in Table 10.

Operation name	storeReputation()	
Operation description	Stores a given reputation within the Reputation scores storage module in such a way that it will be easily retrievable when needed afterwards	
Parameter name	Parameter type	Parameter description
reputation	<Reputation>	Reputation to be stored within the Reputation scores storage module
Return type	Return description	
boolean	Returns 'true' if the storage of the provided <Reputation> element was successful and 'false', otherwise	

Table 8. Operation: storeReputation()

Operation name	retrieveReputationBundle()	
Operation description	Retrieves each reputation stored in the Reputation scores storage module, regarding each of the specified subjects (HBB-NEXT applications)	
Parameter name	Parameter type	Parameter description
subject	<Subject>	Subject (HBB-NEXT application) whose reputation is to be retrieved
⋮	⋮	⋮
subject	<Subject>	Subject (HBB-NEXT application) whose reputation is to be retrieved
Return type	Return description	
<ReputationBundle>	Returns a <ReputationBundle> element containing the <Reputation> elements which, in turn, represent each of the reputations stored regarding each of the specified subjects	

Table 9: Operation: retrieveReputationBundle()

Operation name	retrieveReputation()	
Operation description	Retrieves the reputation stored in the Reputation scores storage module, regarding the specified subject (HBB-NEXT application)	
Parameter name	Parameter type	Parameter description
subject	<Subject>	Subject (HBB-NEXT application) whose reputation is to be retrieved
Return type	Return description	

`<Reputation>`	Returns a `<Reputation>` element which represents the reputations stored in the Reputation scores storage module regarding the specified subject

Table 10. Operation: retrieveReputation()

3.3.1.7. Reputation-based decision making engine

The Reputation-based decision making engine module offers only one main operation named `allowAccessToIdentityAttributes()`, whose description can be observed in Table 11.

Operation name	`allowAccessToIdentityAttributes()`	
Operation description	Determines whether to allow or not a specified subject to have access to (some of) the identity attributes of a specified issuer, based on the reputation of the former from the perspective of the latter	
Parameter name	**Parameter type**	**Parameter description**
`issuer`	`<Issuer>`	End user who wants to consume the specified subject and whose identity attributes need to be retrieved
`subject`	`<Subject>`	Subject (HBB-NEXT application) requesting access to certain identity attributes of the specified issuer
Return type	**Return description**	
`{<Permit>|<Deny>}`	Returns either a `<Permit>` element indicating that the specified subject is trustworthy enough to have access to the identity attributes of the specified issuer, or `<Deny>` otherwise	

Table 11. Operation: allowAccessToIdentityAttributes()

3.3.1.8. Reputation information visualizer

Finally, the Reputation information visualizer provides the following four main operations:

- `renderReputationBundle()`, described in Table 12
- `renderReputation()`, described in Table 13
- `renderFeedbackBundle()`, described in Table 14
- `renderFeedback()`, described in Table 15

Operation name	renderReputationBundle()	
Operation description	Renders a given reputation bundle according to the capabilities of the device in use and the preferences of the user accessing the reputation information	
Parameter name	**Parameter type**	**Parameter description**
reputationBundle	<ReputationBundle>	Reputation bundle to be rendered by the Reputation information visualizer module
deviceCapabilities	<DeviceCapabilities>	Capabilities of the device used to render the specified reputation bundle
systemConditions	<SystemConditions>	Current conditions of the environment used to determine how to render the reputation information
issuer	<Issuer>	Issuer (user) accessing the reputation information
Return type	**Return description**	
HTML5	Returns an HTML5 template containing the rendering of the given <ReputationBundle> element according to the capabilities of the device in use and the current system conditions. Such HTML5 template can be further customized to be adapted to the desired style.	

Table 12. Operation: renderReputationBundle()

Operation name	renderReputation()	
Operation description	Renders a given reputation according to the capabilities of the device in use and the preferences of the user accessing the reputation information	
Parameter name	**Parameter type**	**Parameter description**
reputation	<Reputation>	Reputation to be rendered by the Reputation information visualizer module
deviceCapabilities	<DeviceCapabilities>	Capabilities of the device used to render the specified reputation element
systemConditions	<SystemConditions>	Current conditions of the environment used to determine how to render the specified reputation element

`issuer`	`<Issuer>`	Issuer (user) accessing the reputation information
Return type	**Return description**	
HTML5	Returns an HTML5 template containing the rendering of the given `<Reputation>` element according to the capabilities of the device in use and the current system conditions. Such HTML5 template can be further customized to be adapted to the desired style.	

Table 13. Operation: renderReputation()

Operation name	`renderFeedbackBundle()`	
Operation description	Renders a given feedback bundle according to the capabilities of the device in use and the preferences of the user accessing the reputation information	
Parameter name	**Parameter type**	**Parameter description**
`feedbackBundle`	`<FeedbackBundle>`	Feedback bundle to be rendered by the Reputation information visualizer module
`deviceCapabilities`	`<DeviceCapabilities>`	Capabilities of the device used to render the specified reputation bundle
`systemConditions`	`<SystemConditions>`	Current conditions of the environment used to determine how to render the specified feedback bundle
`issuer`	`<Issuer>`	Issuer (user) accessing the reputation information
Return type	**Return description**	
HTML5	Returns an HTML5 template containing the rendering of the given `<FeedbackBundle>` element according to the capabilities of the device in use and the current system conditions. Such HTML5 template can be further customized to be adapted to the desired style.	

Table 14: Operation: renderFeedbackBundle()

Operation name	`renderFeedback()`	
Operation description	Renders a given feedback according to the capabilities of the device in use and the preferences of the user accessing the reputation information	
Parameter name	**Parameter type**	**Parameter description**

feedback	<Feedback>	Feedback to be rendered by the Reputation information visualizer module
deviceCapabilities	<DeviceCapabilities>	Capabilities of the device used to render the specified feedback element
systemConditions	<SystemConditions>	Current conditions of the environment used to render the specified feedback element
issuer	<Issuer>	Issuer (user) accessing the reputation information
Return type	**Return description**	
HTML5	Returns an HTML5 template containing the rendering of the given <FeedbackBundle> element according to the capabilities of the device in use and the current system conditions. Such HTML5 template can be further customized to be adapted to the desired style.	

Table 15: Operation: renderFeedback ()

3.3.1.9. Summary

This section just lists, as a summary, all the operations offered by each one of the internal modules of the trust and reputation management enabler.

- Device capabilities
  - getDeviceCapabilities()→Table 1
- User's profile
  - getIssuerPreferences()→Table 3
- Feedback collector
  - collectFeedback()→Table 4
- Feedback storage
  - storeFeedback()→Table 5
  - retrieveFeedbackBundle()→Table 6
- Reputation computation engine

        - computeReputation()→Table 7

    - Reputation scores storage

        - storeReputation()→Table 8
        - retrieveReputationBundle()→Table 9
        - retrieveReputation()→Table 10

- **Reputation-based decision making engine**

    - allowAccessToIdentityAttributes()→Table 11

- **Reputation information visualizer**

    - renderReputationBundle()→Table 12
    - renderReputation()→Table 13
    - renderFeedbackBundle()→Table 14
    - renderFeedback()→Table 15

3.3.2. External interfaces

This section shows the description of the interfaces that other external modules or enablers need in order to interact with the trust and reputation management enabler. As shown in Figure 6, two external interfaces are defined in order to interact with the internal modules to resolve reputation-based request.

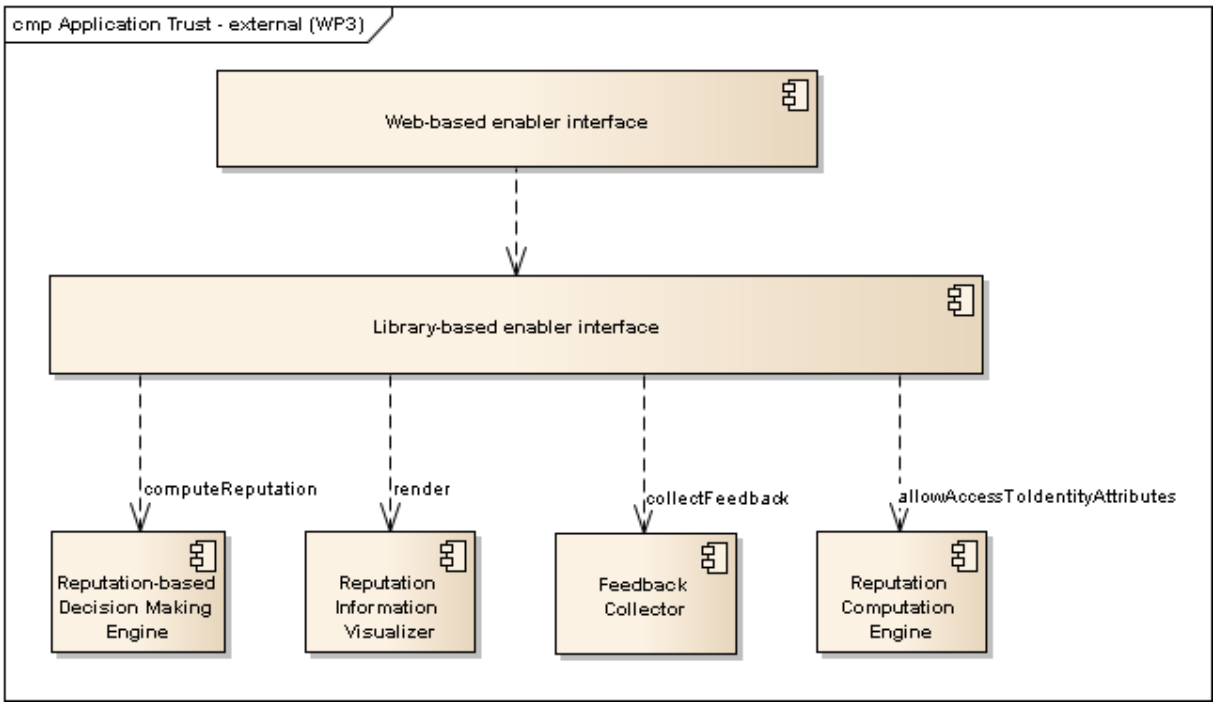

Figure 6: External interfaces design

3.3.2.1. Web-based enabler interface

Since the trust and reputation management enabler could be deployed in an external server, a communication process needs to be defined in order to request information to this enabler. The communication is achieved according to the REST constraints, in such a way that the requesters are able to send messages to this enabler using the HTTP method.

The enabler deploys a module in charge of processing HTTP requests and calling the necessary internal modules to process and answer the query.

Four main types of functions are defined to communicate with the enabler:

1. requestReputationInformation
2. allowAccessToIdentityAttributes
3. provideFeedback
4. render

Function:	Request Reputation Information		
Description:	See 3.3.2.2.1		
Method:	GET	**Resource:**	/reputation/*{subjectid}*

Parameters:	
subjectid	The id of the subject (e.g. HBB-NEXT application) whose reputation is to be retrieved

Function:	allowAccessToIdentityAttributes		
Description:	See 3.3.2.2.2		
Method:	GET	**Resource:**	/reputation/{subjectid}/issuers/{issuerid}
Parameters:			
subjectid	The id of the subject (e.g. HBB-NEXT application) whose reputation is to be retrieved		
issuerid	The id of the end user who wants to consume the specified subject		

Function:	provideFeedback		
Description:	See 3.3.2.2.3		
Method:	POST	**Resource:**	/reputation/{subjectid}
Parameters:			
subjectid	The id of the subject (e.g. HBB-NEXT application) whose reputation is being provided		
BODY	The body of the message shall contain reputation parameters [ref. this doc, sec. 3.1.2]		

Function:	render		
Description:	See 3.3.2.2.4		
Method:	GET	**Resource:**	/render/{renderelementid}/issuers/{issuerid}/devices/{deviceid}
Parameters:			
renderelementid	The id of the renderElement (ReputationBundle, Reputation, FeedbackBundle or Feedback) to be rendered		
issuerid	The id of the end user who will be shown the rendering to		
deviceid	The id of the device used to show the specified element		

3.3.2.2. Library-based enabler interfaces

The trust and reputation management enabler is also planned to be used as a library in order for other external enablers or modules to directly import its functionality. This library is used by the Web-based enabler interface to interact with the internal modules in order to process the received requests. The next subsections define the main operations exposed in that library.

3.3.2.2.1. Operation: Request Reputation Information

The trust and reputation management enabler exposes an operation named `requestReputationInformation()`, to be used by other external enablers or modules in order to retrieve reputation information about a specific subject (an HBB-NEXT application in the context of HBB-NEXT). Table 16 shows the description of this operation.

Operation name	`requestReputationInformation()`	
Operation description	Retrieves the reputation information owned by the trust and reputation management enabler regarding a specified subject (HBB-NEXT application)	
Parameter name	**Parameter type**	**Parameter description**
`subject`	`<Subject>`	Subject (HBB-NEXT application) whose reputation is to be retrieved
Return type	**Return description**	
`<Reputation>`	Returns a `<Reputation>` element which represents the reputation information owned by the trust and reputation management enabler regarding the specified subject	

Table 16: Operation: requestReputationInformation()

3.3.2.2.2. Operation: Request reputation-based decision

This enabler also exposes an operation named `allowAccessToIdentityAttributes()`, in order to obtain a decision regarding allowing or not a specific subject to have access to some issuer's attributes. The description of this operation can be observed in Table 17.

Operation name	allowAccessToIdentityAttributes()	
Operation description	Determines whether to allow or not a specified subject to have access to (some of) the identity attributes of a specified issuer, based on the reputation of the former from the perspective of the latter	
Parameter name	Parameter type	Parameter description
Issuer	<Issuer>	End user who wants to consume the specified subject and whose identity attributes need to be retrieved
Subject	<Subject>	Subject (HBB-NEXT application) requesting access to certain identity attributes of the specified issuer
Return type	Return description	
{<Permit>|<Deny>}	Returns either a <Permit> element indicating that the specified subject is trustworthy enough to have access to the identity attributes of the specified issuer, or <Deny> otherwise	

Table 17: Operation: allowAccessToIdentityAttributes()

3.3.2.2.3. Operation: Provide user feedback

This enabler exposes an operation named provideFeedback(), in order to allow other modules to present a feedback given by a user about a specific subject. The description of this operation can be observed in Table 18.

Operation name	provideFeedback()	
Operation description	Receive a feedback given by a user about a specific subject taking into account the capabilities of the device used by the user and the current system conditions. The feedback will be included in the feedback storage to be taken into account when computing future reputation values about the given subject.	
Parameter name	Parameter type	Parameter description
Issuer	<Issuer>	Issuer (user) providing the feedback
Subject	<Subject>	Subject (HBB-NEXT application) to be evaluated by the issuer

deviceCapabilities	<DeviceCapabilities>	Capabilities of the device used to provide the feedback. Used to determine how to actually collect the feedback from the issuer
systemConditions	<SystemConditions>	Current conditions of the environment. Used to determine how to actually collect the feedback from the issuer
Return type	**Return description**	
void		

Table 18. Operation: provideFeedback()

3.3.2.2.4. Operation: Render

In order to visualize the reputation and feedback information, the enabler offers an operation named `render()`. This operation accepts different attributes depending on the element it has to render. The description of this operation can be observed in Table 19.

Operation name	render()	
Operation description	Renders a given element (Reputation, ReputationBundle, Feedback or FeedbackBundle) according to the capabilities of the device in use, the current system conditions and the preferences of the user who the rendering will be shown to	
Parameter name	**Parameter type**	**Parameter description**
issuer	<Issuer>	Issuer (user) who will be shown the rendering to
deviceCapabilities	<DeviceCapabilities>	Capabilities of the device used to render the specified element
systemConditions	<SystemConditions>	Current conditions of the environment used to render the specified element
renderElement	{<ReputationBundle> |<Reputation> |<FeedbackBundle> |<Feedback>}	The element to be rendered
Return type	**Return description**	
boolean	Returns 'true' if the gathering and storage of the feedback was successful and 'false', otherwise	

Table 19. Operation: render() Sequence Diagrams

This last section presents and describes a set of sequence diagrams depicting the main operations and functionalities of the trust and reputation management enabler within the context of HBB-NEXT.

3.3.3. U.027 – Download app from app-store

The first sequence diagram, shown in Figure 7, represents the use case U.027 introduced in D2.1 [11], and entitled "Download app from app-store".

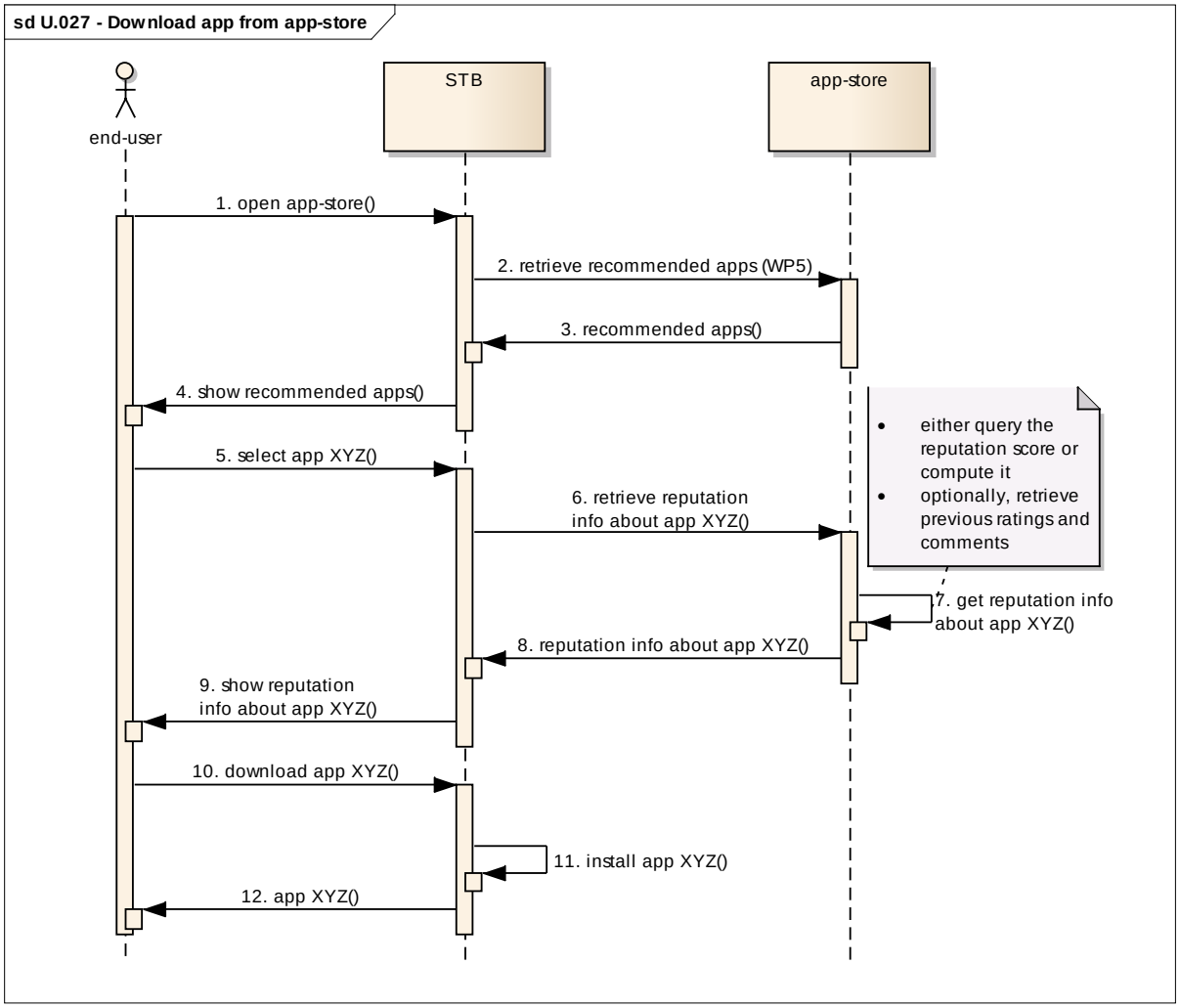

Figure 7. Sequence diagram for U.027 – Download app from app-store (see D2.1 [11])

Its main steps are as follows:

1. User accesses the app-store portal through the STB
2. The STB queries the actual app-store about the recommended applications for the user (or users) accessing the app-store
3. The app-store provides the recommended applications to the STB
4. The recommended applications are shown to the user
5. The user selects application XYZ
6. The STB queries the app-store about the reputation information regarding application XYZ (`requestReputationInformation()`,Table 16)

7. The app-store either computes a customized reputation score (`computeReputation()`, Table 7) or just retrieves a common one already stored (`retrieveReputation()`,Table 10). Optionally, it can also provide feedbacks given by previous users (`retrieveFeedbackBundle()`, Table 6).

8. The reputation information about application XYZ is given back to the STB (<Reputation> element, section 3.1.2).

9. The reputation information about application XYZ is rendered to the user (`renderReputation()`, Table 13).

10. The user, based on the reputation of the application XYZ, decides to download it.

11. The application XYZ is actually installed.

12. The user can now enjoy the application XYZ.

3.3.4. U.031 – Mark app as trusted

The second sequence diagram, shown in Figure 8, represents the use case U.031 introduced in D2.1 [11], and entitled “Mark app as trusted”.

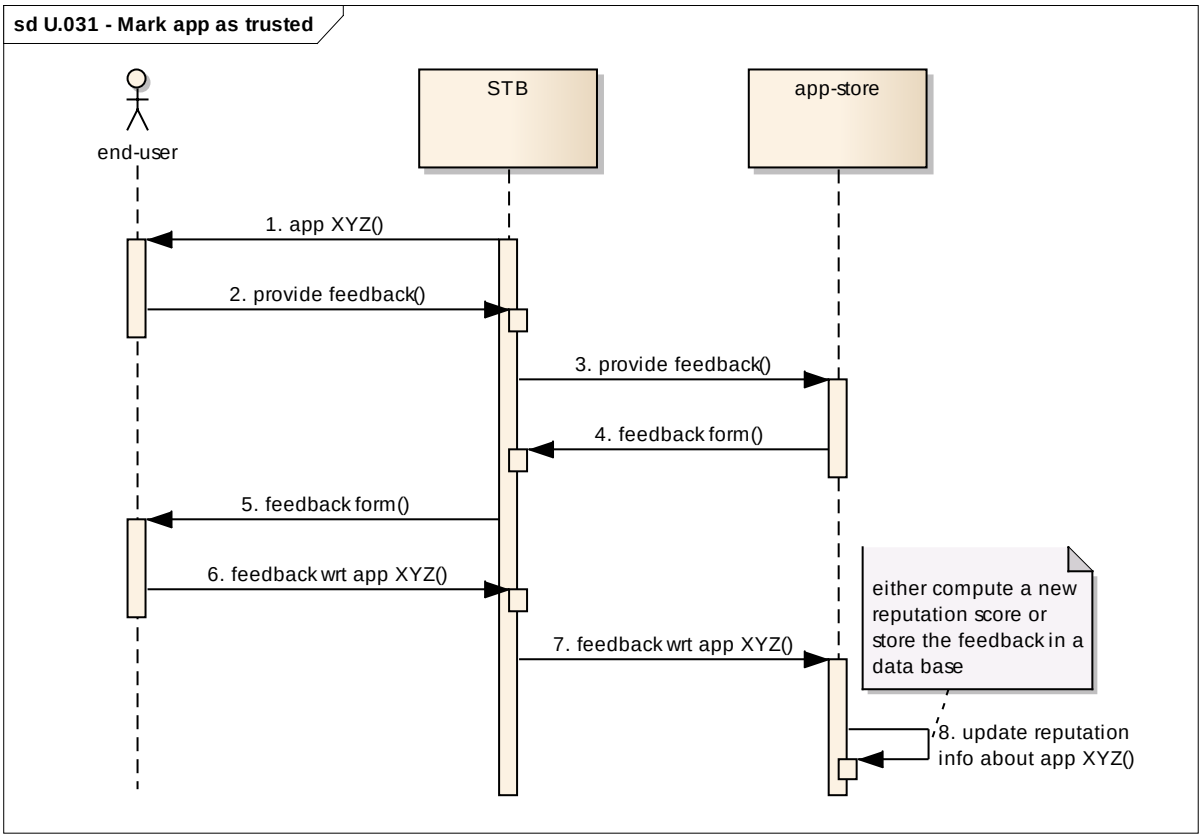

Figure 8. Sequence diagram for U.031 – Mark app as trusted (see D2.1 [11])

Its main steps are as follows:

1. The user has downloaded and installed the application XYZ (as shown in section3.4.1)
2. The user wants to provide a feedback regarding the application XYZ
3. The request to provide feedback regarding the application XYZ is forwarded to the app-store
4. The app-store delivers the feedback form to be used to collect the feedback from the user regarding application XYZ (`collectFeedback()`,Table 4)
5. The feedback form is rendered and presented to the user
6. The user fills in the feedback form with regards to the application XYZ and sends it back to the STB (`<Feedback>` element, section3.1.7)
7. The filled feedback form is forwarded to the app-store

8. The reputation information regarding the application XYZ is updated, which actually means that either the feedback is just stored (`storeFeedback()`,Table 5) or, in addition to that, a new and updated reputation score for application XYZ is also computed (`computeReputation()`,Table 7)

3.3.5. U.032 – Using user identity and security mechanisms

The third and last sequence diagram, shown in Figure 9, represents the use case U.032 introduced in D2.1 [11], and entitled "Using User Identity and Security Mechanisms".

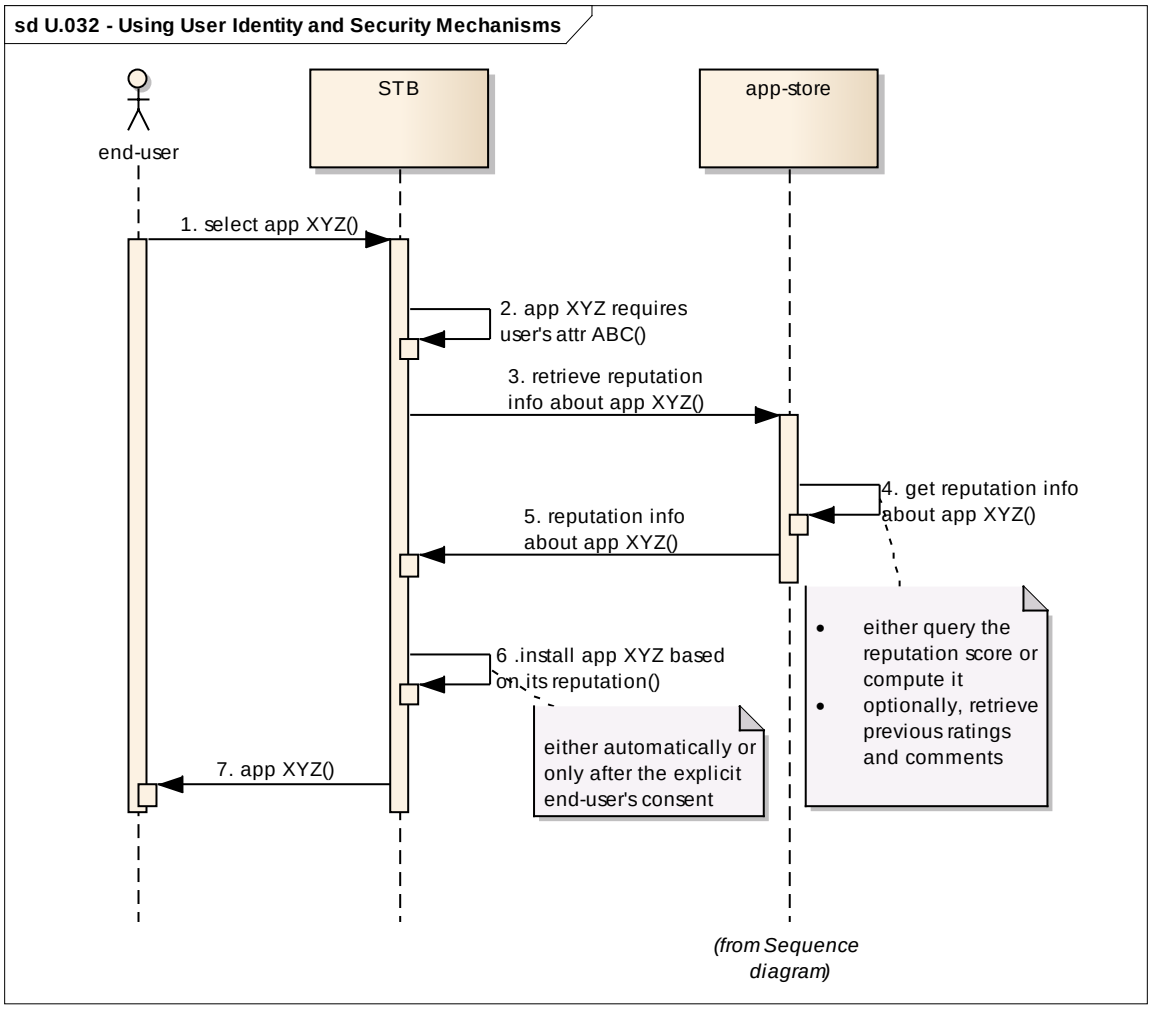

Figure 9. Sequence diagram for U.032 – Using user identity and security mechanisms (see D2.1 [11])

Its main steps are as follows:

1. The user selects the application XYZ to download and install it (step 5 in the sequence diagram shown in Figure 7, section 3.4.1)

2. However, the application XYZ requires certain user's identity attributes such as e-mail, age or credit card in order to be installed

3. The STB queries the app-store about the reputation information regarding application XYZ (`requestReputationInformation()`,Table 16)

4. The app-store either computes a customized reputation score (`computeReputation()`,Table 7) or just retrieves a common one already stored (`retrieveReputation()`,Table 10). Optionally, it can also provide feedbacks given by previous users (`retrieveFeedbackBundle()`,Table 6)

5. The reputation information about application XYZ is given back to the STB (<Reputation> element, section 3.1.2)

6. Then, based on the reputation of the application XYZ, it is allowed or not to access to the requested user's identity attributes (`allowAccessToIdentityAttributes()`, Table 11)

7. In case the application XYZ is allowed to access the requested user's identity attributes, it is actually installed and delivered to the user

4. Identity Management Module

This chapter describes one main part of the delivery of WP3: The Identity Management module. The module contains all users, basic information about them, security credentials, and connections between users, devices, and contexts.

IdM will provide an API to access these parameters easily, and in a future stage add some interpretation to them as well to enable services building logic upon them.

4.1. Data model

This section contains the data model.

Note: The data model field obligation is indicated with 'M/O' - meaning 'mandatory/optional'. The amount of objects is indicated in the occurrences column with '1/n' - meaning 'unique/multiple'.

Note: The JSON type 'boolean' summarizes true/false as permitted values for the object.

Note: The JSON type 'date' specifies a string as permitted value for the object in the format YYYY-MM-DD acc. ISO 8601, where YYYY equals the four-digit year, MM equals the two-digit month (01=January, etc.), and DD equals the two-digit day of month (01 through 31).

Note: The first value in an array is the default value. Other values shall be used or can be selected acc. particular use cases.

4.1.1. User

The user data model is shown in Table 20. The JSON type refers to the specification from www.json.org. It is for example implicit that the password shall not be provided as 'bare string' but rather MD5 hashed (or any format that is required by apps).

Level 1	Level 2	Level 3	JSON Type	JSON Object	Obligation	Ocurrences
				(preliminary)		
Identifier (unique)			object		M	1
	ID		string	id	M	1
	Alias		array	alias	O	n
Personal information			object		M	1
	Gender		string	gender	O	1
	First Name		string	first_name	M	1
	Last Name		string	last_name	O	1
	Date of birth		date	dob	O	1
	Contact means		object		O	1
		E-mail addresses	array	mail	O	n
		Phone number	array	phone	O	n
Credentials			object		M	1
	Password		string	password	M	1
	PIN		number (digits)	pin	O	1
	Multi-modal information		string	mmi	O	n
Links			object		O	1
	Device		link	<< device	O	n
	User (incl. weight: family, friend, implicit/explicit, etc.)		link	users_id	O	n
	Context		link	<< context	O	1
	Profile		link	profile	O	1
API version:	1					
Date:	06.09.2012					

Table 20. User data model

The data model may be extended for future API versions.

The 'link' type is a reference pointing to the API ID for the respective element. In a later version of the API more than just bare objects/arrays will be provided. The IdM enabler is aimed provide more intelligence for providing related information. Links between elements can be weighted, i.e., they are not only directed connections between two elements (whereas 'device' is the origin), but may have a label (weight) that specifies the connection further. This can be used to parameterize connections between users further (not only *that* I know another user, but *how*).

The multi-modal information is expected to be a transparently passed BASE64 encoded string.

Note: Table 20 shows only the high level JSON type. Due to the fact that this specification is not final the content is subject to further studies. This includes primarily, but is not limited to, all nested arrays and objects.

A JSON sample of the data model is shown below:

```
{
  "alias": "john.doe",
  "created_at": "2012-09-06T13:56:02Z",
  "dob": "1951-01-13",
  "first_name": "John",
  "gender": "m",
  "id": 3,
  "last_name": "Doe",
  "mail": "john.doe@example.net",
  "mmi": "SEJCLU5leHQ=",
  "password": "5ebe2294ecd0e0f08eab7690d2a6ee69",
  "phone": "+12345678901",
  "pin": 1234,
  "profile": null,
  "updated_at": "2012-09-06T15:15:07Z",
  "users_id": null
}
```

It could be the body of a response to the following HTTP request:

```
GET /users/3.json HTTP/1.1

User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0
OpenSSL/0.9.8r zlib/1.2.5

Host: localhost:3000

Accept: */*
```

4.1.2. Device

The device data model is shown in Table 21. The JSON type refers to the specification from www.json.org.

Level 1	Level 2	Level 3	JSON Type	JSON Object	Obligation	Ocurrences
				(preliminary)		
Identifier (unique)			object		M	1
	ID		string	id	M	1
	Alias		string	alias	O	n
Properties (unique name attributes/properties/settings/etc.			object		M	1
State* (active/not active)			boolean	state	O	1
Links			object		O	1
	User (incl. weight: family, friend, implicit/explicit, etc.)		link	users_id	O	n
	Context		link	<< *context*	O	1
Note *: This attribute is still subject to discussion. Can be integrated with context.						
API version:	1					
Date:	06.09.2012					

Table 21. Device data model

The data model may be extended for future API versions.

The properties element can contain device specific sub-elements. There is currently no need to define that, as it is used per service and data is transparent to the API. This statement may however be subject to future change.

The 'link' type is a reference pointing to the API ID for the respective element. Links between elements can be weighted, i.e., they are not only directed connections between two elements (whereas 'device' is the origin), but may have a label (weight) that specifies the connection further. This can be used to parameterize the relation of e.g. a user to a device (e.g. watcher, owner, user, etc.). Further logic can process and use this information.

4.1.3. Context

The context data model is shown in . The JSON type refers to the specification from www.json.org.

Level 1	Level 2	Level 3	JSON Type	JSON Object	Obligation	Ocurrences
				(preliminary)		
Identifier (unique)			object		M	1
	ID		string	id	M	1
	Alias		string	alias	O	n
State (active/not active)			boolean	state	M	1
Links			object		O	1
	User (incl. weight: family, friend, implicit/explicit, etc.)		link	users_id	O	n
	Device		link	devices_id	O	n
Note: The context may be split up in active and usual contexts. Data model thus subject to change.						
API version:	1					
Date:	06.09.2012					

Table 22. Context data model

The data model may be extended for future API versions.

The 'link' type is a reference pointing to the API ID for the respective element.

It is for further study how actual and usual contexts are mapped.

The status might be omitted in the future, if the activity mapping is handled by separation of responsibilities between the enabler.

Contexts play a major role in the implementation of the identity management. The IdM uses the context element to map the usual and actual context of a user. While the context assignment in WP3 is rather static and only its activity changes, the context assignment in scope of WP5 (see D5.2 [8]) is dynamic.

The "usual context" represents the context (defined in D3.1 [9]), in which a user is usually present, i.e., has been present before. It is relevant for defining the sub-set for multi-modal identification.

The "active context" represents the current context of a user, where he is presently active in. It is used for interaction.

"Context" is used to statically group users and devices. They can, however, still have direct relations.

4.2. Design

The module consists of front-end, back-end, and database layers.

The front-end is an HTTP web service in the form of a RESTful API. The back-end contains the module logic and communicates with the data base layers.

4.2.1. Database implementation

4.2.1.1. Description

The data model shows the three main domains:

- User
- Device
- Context

Typically, users are permanently related to each other and to their respective devices.

The context maps a temporary relation between users (that may or may not be permanently related, but most likely are) and between devices.

Relations can have a “weight”, i.e., roles that further define the relation, but this relation is not mandatory. Users can administer other users (e.g. TV owner can create user accounts) and also devices (e.g. device owner can configure the device). Users can also use devices – with restricted permissions.

Figure 10 shows the relations between the domains in both identity and profile management.

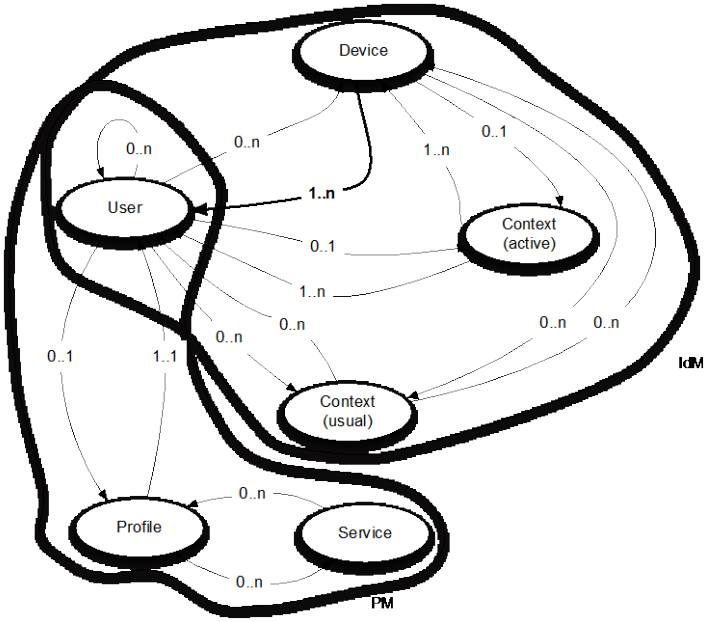

Figure 10. Relations between identity and profile management domains

A user can have relations to many other users.

A user can have many devices. Devices must belong to one or many users and their relation is weighted.

A user can only be active in a single context. An active context can have one or many users attached and its relation may be weighted towards the controlling user.

A user can be part of many usual contexts. A usual context can have users attached.

A device can only be active in a single context. An active context can have one or many devices attached and its relation may be weighted towards the controlling device.

A device can be part of many usual contexts. A usual context can have devices attached.

A user can have only one profile. A profile is always assigned to a particular user. Services may be assigned to profiles.

Relations of profile and service are covered in section 4.3.

4.2.1.2. Database layout

```
users
      integer :id
      string :alias
      datetime :created_at
      datetime :updated_at
      string :first_name
      string :last_name
      string :gender
      date :dob
      string :mail
      string :phone
      string :password
      integer :pin
      text :mmi
      integer :users_id
      integer :profile
devices
      boolean :state
      integer :users_id
      string :alias
contexts
      boolean :state
      integer :users_id
      integer :devices_id
      string :alias
```

Each table has id (integer), created timestamp, and modified timestamp fields. The time stamp has the date format '2012-01-01 13:25:37 UTC'.

The data format is subject to future change, as currently the table-linking '_id' fields allow only singular links (integer). Objects and arrays respectively allow weighted and non-weighted multi-links.

4.3. Interface

4.3.1. Internal Implementation

The identity management module has been implemented acc. the design that is described in section 4.2.

4.3.2. API

The basic functions of the identity management component are described in D6.1.1, sec. 6.3.4 [10]. The mentioned functions in this deliverable are not elaborated and mapped to the API.

Note: In case no parameters are added, this shall be indicated by inserting 'n/a' (not applicable) into the table.

Note: Data types for the parameters are stated in section 4.1 covering the data model.

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete user in the system**		
Function:	Retrieve user		
Method:	GET	**Resource:**	/users/*{userid}*
Parameters:			
userid	The id of the user that shall be retrieved.		
BODY	The body of the message shall contain basic user parameters acc. [ref. this doc, sec. 4.1]		

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete user in the system**		
Function:	Add user		
Method:	POST	**Resource:**	/users
Parameters:			
BODY	The body of the message may contain user parameters acc. [ref. this doc, sec. 4.1]. These parameters are provisioned to the created user.		

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete user in the system**		
Function:	Modify user		
Method:	PUT	**Resource:**	/users/*{userid}*
Parameters:			
userid	The id of the user that shall be modified.		

BODY	The body of the message shall contain user parameters acc. [ref. this doc, sec. 4.1]. These parameters are provisioned to the user that shall be updated. They will override existing parameters.

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete user in the system**		
Function:	Delete user		
Method:	DELETE	**Resource:**	/users/*{userid}*
Parameters:			
userid	The id of the user that shall be deleted.		

Function (Ref. D6.1.1):	**Add, modify, delete user from a context**		
Function:	Create context		
Method:	POST	**Resource:**	/contexts
Parameters:			
n/a	n/a		

Function (Ref. D6.1.1):	**Add, modify, delete user from a context**		
Function:	Add user to context		
Method:	POST	**Resource:**	/contexts/*{contextid}*/users?id=*{userid}*
Parameters:			
contextid	The context that is being modified.		
userid	The ID of the user to be added.		

Function (Ref. D6.1.1):	**Add, modify, delete user from a context**		
Function:	Modify user in context (necessity to be discussed in further study)		
Method:	PUT	**Resource:**	/contexts/*{contextid}*/users/*{userid}*
Parameters:			
contextid	The context that is being modified.		
userid	The ID of the user to be modified.		

Function (Ref. D6.1.1):	**Add, modify, delete user from a context**		
Function:	Delete user from context		
Method:	DELETE	**Resource:**	/contexts/{contextid}/users/{userid}
Parameters:			

contextid	The context that is being modified.
userid	The ID of the user to be deleted.

Function (Ref. D6.1.1):	**Add, modify, delete user from a context**		
Function:	Delete context		
Method:	DELETE	**Resource:**	/contexts/{contextid}
Parameters:			
contextid	The context that is being deleted.		

Function (Ref. D6.1.1):	**Retrieve users active in a context (at the present moment)**		
Function:	see above		
Method:	GET	**Resource:**	/contexts/{contextid}/users
Parameters:			
contextid	The context that is being used.		

Function (Ref. D6.1.1):	**Retrieve users relevant for a context (usually active there)**		
Function:	see above		
Method:	GET	**Resource:**	/contexts/*{contextid}*/users
Parameters:			
contextid	The context that is being used.		

Function (Ref. D6.1.1):	**Retrieve devices active in a context (at the present moment)**		
Function:	see above		
Method:	GET	**Resource:**	/contexts/*{contextid}*/devices
Parameters:			
contextid	The context that is being used.		

Function (Ref. D6.1.1):	**Retrieve devices relevant for a context (usually active there)**		
Function:	see above		
Method:	GET	**Resource:**	/contexts/*{contextid}*/devices
Parameters:			
contextid	The context that is being used.		

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete device in the system**

Function:	Retrieve device		
Method:	GET	**Resource:**	/devices/{deviceid}
Parameters:			
deviceid	The device id of the device that shall be retrieved.		
BODY	The body of the message shall contain basic device parameters acc. [ref. this doc, sec.]		

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete device in the system**		
Function:	Add device		
Method:	POST	**Resource:**	/devices
Parameters:			
BODY	The body of the message shall contain device user parameters acc. [ref. this doc, sec.]		

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete device in the system**		
Function:	Modify device		
Method:	PUT	**Resource:**	/devices/{deviceid}
Parameters:			
deviceid	The device id of the device that shall be retrieved.		
BODY	The body of the message shall contain basic device parameters acc. [ref. this doc, sec.]		

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete device in the system**		
Function:	Delete device		
Method:	DELETE	**Resource:**	/devices/{deviceid}
Parameters:			
deviceid	The device id of the device that shall be retrieved.		

Function (Ref. D6.1.1):	**Add, modify, delete device from a context**		
Function:	Add device to context		
Method:	POST	**Resource:**	/contexts/*{contextid}*/devices?id=*{deviceid}*
Parameters:			
contextid	The context that is being modified.		
deviceid	The ID of the device to be added.		

Function (Ref. D6.1.1):	**Add, modify, delete device from a context**		
Function:	Modify device in context (necessity to be discussed in further study)		
Method:	PUT	**Resource:**	/contexts/{contextid}/devices/{deviceid}
Parameters:			
contextid	The context that is being modified.		
deviceid	The ID of the device to be added.		

Function (Ref. D6.1.1):	**Add, modify, delete device from a context**		
Function:	Delete device from context		
Method:	DELETE	**Resource:**	/contexts/{contextid}/devices/{deviceid}
Parameters:			
contextid	The context that is being modified.		
deviceid	The ID of the device to be added.		

Function (Ref. D6.1.1):	**Retrieve devices of a user**		
Function:	see above		
Method:	GET	**Resource:**	/users/*{userid}*/devices
Parameters:			
userid	The user that is being used.		

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete relation between a user (relation, value)**		
Function:	Retrieve relation between users		
Method:	GET	**Resource:**	/users/*{userid}*/users
Parameters:			
n/a	n/a		

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete relation between a user (relation, value)**		
Function:	Add/modify relation between users		
Method:	PUT	**Resource:**	/users/*{userid1}*/users?id=*{userid1}*&bi=*{bi}*&w=*{w}*
Parameters:			
userid1	ID of user to be modified		
userid2	ID of user relation to be added		
bi	Indicator whether relation shall be added bilateral (default: no)		

w	weight of the relation

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete relation between a user (relation, value)**		
Function:	Delete relation between users		
Method:	DELETE	**Resource:**	/users/{userid1}/users?id={userid1}&bi={bi}
Parameters:			
userid1	ID of user to be modified		
userid2	ID of user relation to be deleted		
bi	Indicator whether relation shall be added bilateral (default: no)		

Function (description):	**Supporting function**		
Function:	API description: Retrieve available resources		
Method:	GET	**Resource:**	/doc
Parameters:			
n/a	n/a		

The resource that is mentioned in the API description is adhered to the root URL, for example:

- Root URL: *https://idm.example.org/api/v1/*
- Full URL for first case: *https://idm.example.org/api/v1/user/a1b2c3*

The response code will provide general information about the success of the request. The response headers carry information relevant to the request and further requests. The response bodies of all requests will contain the complete data sets.

Data can be searched using the search string for the value in the form */users/?q=Jon&key=firstName*. The answer will be provided in the body of the HTTP response message.

Unless specified otherwise, the default content is used in the body. The requested resource might specify the content type in case multiple content types are offered (e.g. */users.json* or */users.xml*).

The following functions (based on D6.1.1 [10]) are not explicitly mapped in the API description above, but rather build with the existing commands:

- Provide multi-modal vectors for users relevant for a context (single modes, all modes)
- Fulfill requirements of security management (e.g. credentials)
- Retrieve, add, modify, delete roles of a user
- Attach, detach user to device (*PUT /users/{userid}?device={deviceid}*)

Further documentation will provide samples how this can be done.

4.4. Sequence Diagrams

Major functions (see section 4.3.2) will be put in a sample context using sequence diagrams in the future.

The sample use case section 4.3.3 from deliverable D2.1 [11] is depicted in Figure 11:

- Peter + Paul on Peters couch
- Paul opens smartphone app
- Paul joins group
- Rec. personal EPG

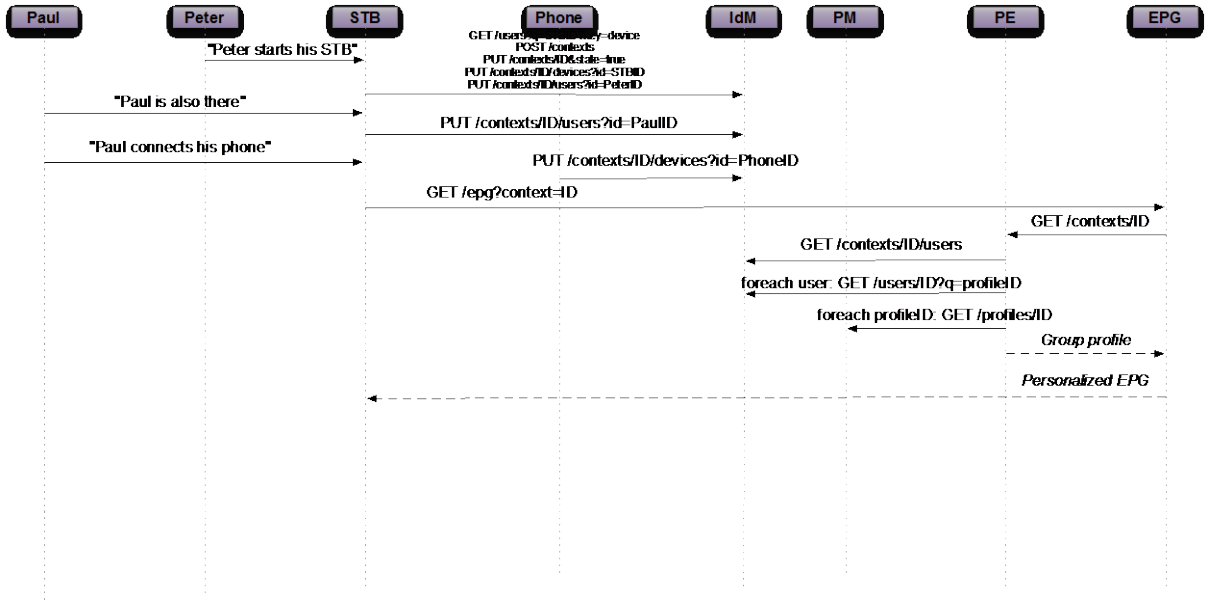

Figure 11. Sequence diagram for the flow in deliverable [D2.1 [11], sec 4.3.3]

GET /users?q=STBID?key=device (STB → IdM)

Retrieve users that are usually active on device. Used for recognition (omitted in this simplified sample).

POST /contexts (STB → IdM)

Create new context.

PUT /contexts/ID&state=true (STB → IdM)

Context is changed to active.

PUT /contexts/ID/devices?id=STBID (STB → IdM)

STB is added to context.

PUT /contexts/ID/users?id=PeterID (STB → IdM)

Peter is added to the context. It is out of scope of this scenario how it is done. It should be done by recognition (after probability threshold is reached) or implicit selection at the STB.

PUT /contexts/ID/users?id=PaulID (STB → IdM)

Paul is added to the context. It is out of scope of this scenario how it is done. It should be done by recognition or implicit selection at the STB.

PUT /contexts/ID/devices?id=PhoneID (Phone → IdM)

Paul's phone is added to the context. It is out of scope of this scenario how it is done. Acc. the scenario description, the STB recognizes the device. It can either add it (by exchanging an ID) or exchange authorization parameters that permit the device to add itself to the context.

GET /epg?context=ID (STB → EPG)

Sample request for EPG for a certain context.

GET /contexts/ID (EPG → PE)

EPG requests personalized profile for context.

GET /contexts/ID/users (PE → IdM)

PE requests all active users in the given context. The devices are not relevant for this use case.

foreach user: GET /users/ID?q=profileID (PE → IdM)

The PE requests the profile IDs for all active users.

foreach profileID: GET /profiles/ID (PE → PM)

The PE requests the single user profile for all active users to form the group profile.

Note: This sample does not assume that multi-modal recognition is used. The multi-modal interface would in this case request all users of the device and their multi-modal identification array and then provision the recognized users. A further step would identify the users beyond threshold probability as identified.

Note: All transactions that have no response have a direct implicit response containing the requested data in the message body.

5. Profile Management Module

This chapter describes the profile management module.

5.1. Data model

This section contains the data model.

Note: The data model field obligation is indicated with 'M/O' - meaning 'mandatory/optional'. The amount of objects is indicated in the occurrences column with '1/n' - meaning 'unique/multiple'.

5.1.1. User Profile

The profile data model is shown in Table 23.

Level 1	Level 2	Level 3	JSON Type	JSON Object	Obligation	Ocurrences
				(preliminary)		
Identifier (unique)			object		M	1
	ID		string	id	M	1
	Alias		array	alias	O	n
Services			object		O	n
	Service ID		string	service	M	1
	User ID in service		string	suid	O	1
	Permissions		object	permissions	O	1
	Attributes		object	attributes	O	1
		Attr. n	object		O	n
Links			object		M	1
	User		link	user	M	1
API version:	1					
Date:	08.09.2012					

Table 23. Profile data model

The data model may be extended for future API versions.

For providing the highest flexibility, service parameters in the profile and service definitions are decoupled. Thus, the service ID is used to link to the service profile. Profile dependent parameters can be added if necessary

5.1.2. Service Profile

The service data model is shown in Table 24.

Level 1	Level 2	Level 3	JSON Type	JSON Object	Obligation	Ocurrences
				(preliminary)		
Identifier (unique)			object		M	1
	ID		string	id	M	1
	Alias		array	alias	O	n
Services			object		M	1
	Name		string	name	M	1
	Address		string	address	M	1
API version:	1					
Date:	08.09.2012					

Table 24. Service data model

The data model may be extended for future API versions.

The service profile defines services that the user profile references to. For now, it specifies the address of the service. Parameters can be added if necessary.

5.2. Design

The module consists of front-end, back-end, and database layers.

The front-end is an HTTP web service in the form of a RESTful API. The back-end contains the module logic and communicates with the data base layers.

5.2.1. Database implementation

5.2.1.1. Description

The data model shows the two main domains:

- Profile
- Service

Figure 10 in section 4.2.1.1 shows the relations between the domains in both identity and profile management.

The profile is referenced in the user data.

5.3. Interface

5.3.1. Internal Implementation

The profile management module has been implemented acc. the design that is described in section 5.2.

5.3.2. API

The basic functions of the profile management component are described in [D6.1.1, sec. 6.3.5]. The mentioned functions in this deliverable are not elaborated and mapped to the API.

Note: In case no parameters are added, this shall be indicated by inserting 'n/a' (not applicable) into the table.

Note: Data types for the parameters are stated in section 5.1 covering the data model.

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete profile in the system**		
Function:	Retrieve profile		
Method:	GET	**Resource:**	/profiles/{profileid}
Parameters:			
profile id	The id of the profile that shall be retrieved (linked in user struct).		

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete profile in the system**		
Function:	Add profile		
Method:	POST	**Resource:**	/profiles
Parameters:			
BODY	The body of the message may contain profile parameters acc. [sec. 5.1.1]. These parameters are provisioned to the created user profile.		

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete profile in the system**		
Function:	Modify profile		
Method:	PUT	**Resource:**	/profiles/{profileid}
Parameters:			
profileid	The id of the profile that shall be modified.		

BODY	The body of the message shall contain profile parameters acc. [ref. this doc, sec.]. These parameters are provisioned to the profile that shall be updated. They will override existing parameters.

Function (Ref. D6.1.1):	**Retrieve, add, modify, delete profile in the system**		
Function:	Delete profile		
Method:	DELETE	**Resource:**	/profiles/{profileid}
Parameters:			
profile	The id of the profile that shall be deleted.		

The resource that is mentioned in the API description is adhered to the root URL, for example:

- Root URL: https://pm.example.org/api/v1/
- Full URL for first case: https://pm.example.org/api/v1/profiles/a1b2c3

The response code will provide general information about the success of the request. The response headers carry information relevant to the request and further requests. The response bodies of all requests will contain the complete data sets.

Data can be searched using the search string for the value in the form /profiles/?q=abc12345&key=id.

Unless specified otherwise, the default content is used in the body. The requested resource might specify the content type in case multiple content types are offered (e.g. /profiles.json or /profiles.xml).

5.4. Sequence Diagrams

The sequence diagram in section 4.4 contains also a sample flow for the profile management.

6. Security Manager

This chapter describes high level design for implementation of Security Manager into HBB Next project. It is designed with following key aspects:

- Provide key&certificate managementrelated functions within HBB-NEXT domain
- Provide privacy and access control fou users
- Be platform independent
- Support open APIs for smooth integration of current or future HBB-NEXT technology building blocks
- Support any of present and future multimedia HBB-NEXT services
- Reliability, resilience and security
- Cost-effective operation and maintenance

Please note that this document describe design of proposed implementation that is specific for functional prototype. Any other design specifics that are beyond of the prototype focus is subject of another document. However specific prototype design described here is fully align with the general Security Manager design.

6.1. Scope

The Security Manager (SM) component is responsible to:

- manage multi-factor authentication, authorization, and policy enforcement for different levels of privacy and for profile data access control,
- handle authentication, i.e. it acts as an identity provider towards the STB. The IdM shall act as back-end for the authorization process,
- manage and verify tokens,
- provide PKI management for HBB Next domain: Certificate Authority (CA) and Certificate Revocation List (CRL),
- store of security related data (pass phrases, certificates, keys, tokens, …),

- grant secure access of the HBB Next local services to the external networks,
- update/retrieve certain data in IDM.

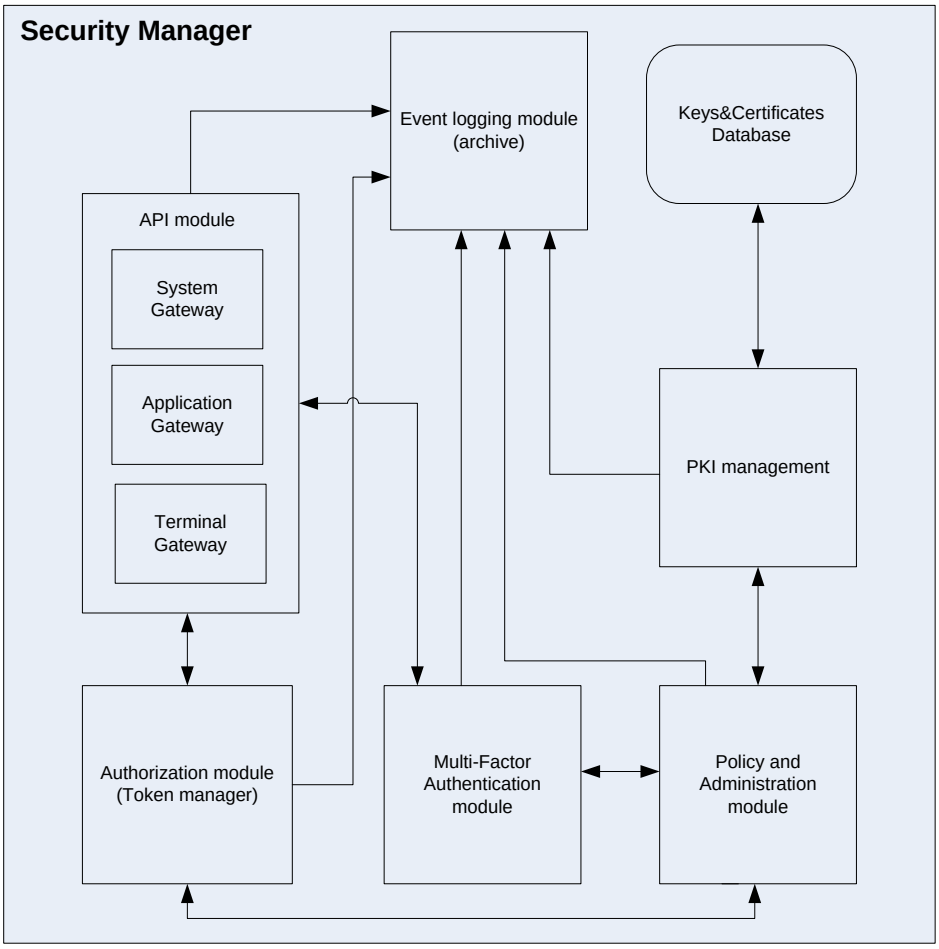

Figure 12. Internal architecture of the Security Manager

The Security Manager consists of several modules:

Authorization module (Token manager) – will evaluate if the terminal has access to the HBB network as role-based access controller. This module will either grant or deny access to the HBB domain.

Multi-Factor Authentication (MFA) module – will be responsible for managing multi-factor authentication processes, depending on required level of security during User/Group is accessing the application/s.

MFA will perform a user identification process to create a list of "best matches" and then perform a series of verification processes to determine a conclusive match. Number of necessary verification processes depends on the required authentication level for accessing the service/application/data. HBB-NEXT project supposes to have several required levels of authentication to verify users before accessing the service with required “level of security”. (Personal EPG – low level is required; instant messaging – middle level is required, e-banking – high level is required)

Policy and Administration module – policy enforcer for MFA module and point of security rules mapping and configuration.

PKI management – will be intended to manage Public Key Infrastructure (PKI) tasks within HBB-NEXT domain, acting as certificate management system within HBB-NEXT domain.

Event logging module (archive) – is event logger for all activities performed by SM.

Keys&Certificates Database – will hold sensitive cryptographic key information and single public key certificates.

API module – secure interface for communication among HBB-NEXT entities consisting of three different sub-modules, the so-called gateways.

- *System Gateway* – an interface towards HBB-NEXT core modules (Identity Management, Profile Manager, Trust & Reputation)
- *Application Gateway* – an interface towards HBB-NEXT application, which will be used only for communicating with Application Servers and not to applications hosted on the terminal. *Note – necessity of this interface will be a part of further detailed design analysis within WP3.*
- *Terminal Gateway* – an interface towards Terminal/ End-user device. This Gateway is needed because of different rules for external devices/terminals might be used than to all servers within Core and Application layers.

6.1.1. Solution Overview

6.1.1.1. Depended entities

Identity Management (IdM) - This module is responsible for Identity Management role in HBB Next environment. It stores and manages all user related identity data and provides it for other entities, excluding security related data (i.e. passphrases, secrets, certificates, keys …). Security Manager updates respective data in the IDM module to keep it actual.

Multimodal Interface (MM) - This module is responsible for identification and recognition of subjects that are presented in certain area either by its voice or by its visage. Additionally, for security purpose, MM module provides to the Security Manager data that are subject of additional check to confirm (or increase) security level of the user. Security Manager might enforce various ways how to prove user's identity, based on the situation or following action needs.

EPG - This module offer Electronic Program Guide (EPG) for the HBB Next service users. Based on the metadata and related services, it generate data that are sent to the display unit. One aspect that influence process of asset generation is the content that actual user is able to view or follow. Here the Security Manager fulfils the role of supplying engine providing to EPG Service information whether this user is authenticated enough to be authorized to view certain content. Authentication level is determined with cooperation of MM module.

Facebook FrontEnd application (FB FE app) – This module is described for prototype purpose only. FB FE app represents here general social network service that is used by HBB-NEXT users. The Security Manager is responsible for the secure access of FB FE app to the FB app central service, by granting access via distribution of tokens. Security Manager also provides service whether certain user is obliged to use such service (authenticated enough) and manage triggering of additional multifactor authentication.

The above mentioned modules are planned to be used for demo purposes, but all other application modules/entities will depend on Security Module policy enforcement, thus shall be mentioned here.

6.1.2. Management

The Security Manager might be connected to the management system of respective provider. This system is beyond of the scope of this design document and is subject of specific provider related integration. The general requirement is to connect the provider's own higher level OSS/BSS via a firewall that also does NAPT.

6.1.3. Hardware architecture

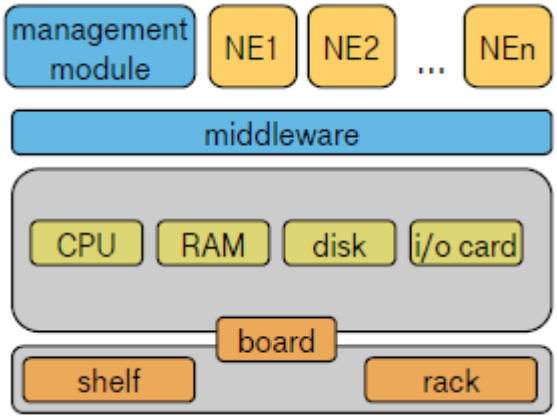

Figure 13. Security Manager - Overview of hardware architecture

Security manager is designed as HW platform independent component that must follow criteria:

- CPU: 2.2GHz dualCore
- RAM: 2GB
- HDD: 250GB
- Network: 100 Mbps ETH

6.1.4. Authorization model

Authorization is important part of overall security in the HBB Next domain, especially for all terminals, such us set-top-boxes or mobile HBB next-enabled devices as well as all modules which requires secure APIs communication.

Authorization module as a part of Security Manager takes the responsibility to grant or deny access to the HBB next domain for all terminals and together with authentication shall take the role of Provider for Centric Identity. Provider of Centric Identity is not to be used as Identity Management for the users of HBB next services, but for the HBB Next terminals and all modules which are in direct communication with Security Module (through the secured APIs) or requires secure APIs in between.

Model will be based on the OpenID and OAuth standards together with the Public Key Infrastructure (PKI) architecture.

Note: Authorization model is part of further design analysis, where all details will be described.

6.1.5. PKI architecture

This chapter describes the Public Key Infrastructure (PKI) design within the HBB Next domain. The components of Security Manager module responsible for certificate generation, revocation, lifecycle and key distribution management are highlighted in Figure 14.

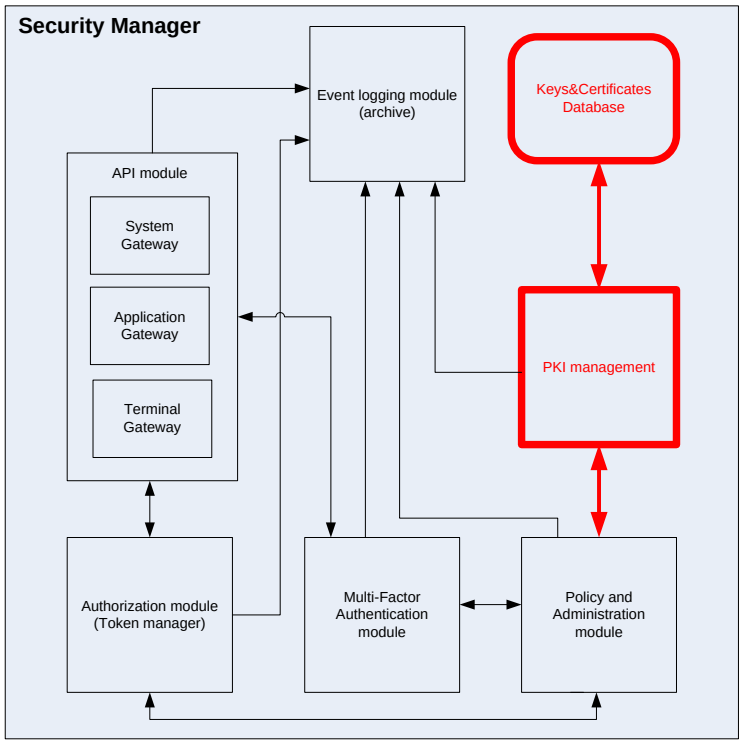

Figure 14. Overview of responsible modules

PKI Management and Keys&Certificates Database plays role of Public Key Infrastructure internally, for one HBB Next domain. This chapter describes internal structure of those components as well as the whole enrolment and revocation strategy, including the details about key distribution within the "home" HBB Next domain. Relations between HBB Next domains are not part of this chapter and will be analysed separately.

PKI management is a place of certificate authorities – Certificate Authority and Registration Authority. Optionally, it can host a responder for certificate status, if Certification Revocation List (CRL) is not used.

Usage of CRL or Online Certificate Status Protocol (OCSP) or combination of themis subject of further design analysis. This deliverable provides first designs and protocols for the following components:

6.1.5.1. PKI Management

PKI Management is one of the modules of the Security Manager, used to create, distribute, store and revoke digital certificates within the HBB Next domain.

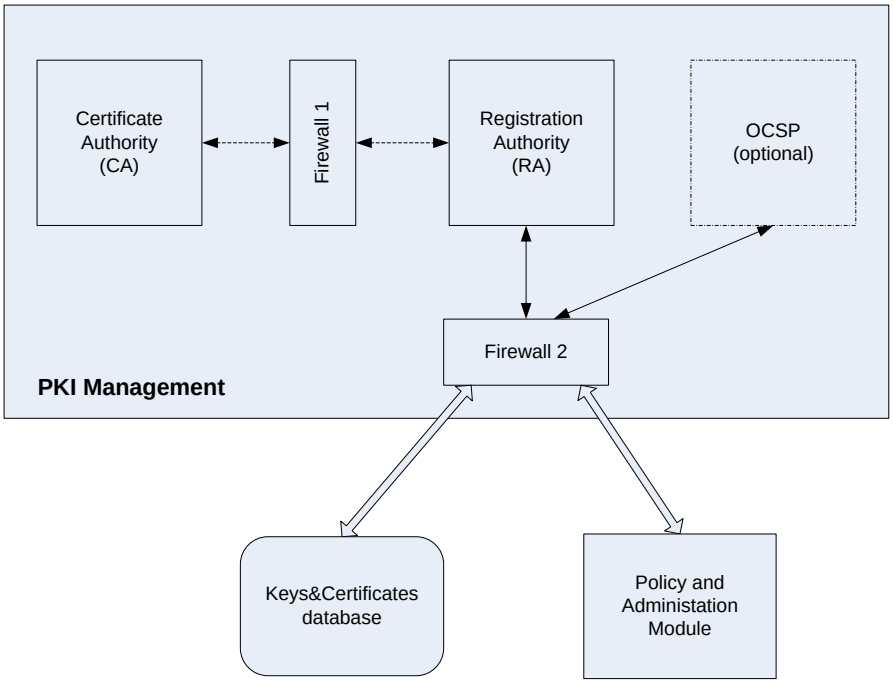

Figure 15. PKI Management – internal architecture

Certificate Authority (CA)

CA is a trusted certificate issuer and the top of trusted hierarchy within the HBB Next domain. CA will be strictly separated and wouldn't be networked with any other component, except with the associated RA. Connection will be separated by an internal firewall, protecting the CA.

Registration Authority (RA)

The RA is used for modest security level, which means that only the RA can forward certification request to CA within the HBB Next domain. Otherwise, isolated CA approach would be used (highest security level), and human intervention is needed.

The RA will only communicate with the certificate requester and post a copy of a certificate to an LDAP directory, located in the Keys&Certificates database.

Online Certificate Status Protocol (OCSP) sever

The OCSP server is used for checking the revocation status of the already issued certificates. This component does not have to be presented if Certificate Revocation List stored in LDAP is used. Both methods, OCSP vs CLR, has pros/cons and must be evaluated in further design.

Usage of OCSP is marked as optional for now.

Firewall 1

A logical component which strictly defines Access Control (ACL) rules, protects the CA and only allows communication between the CA and the associated RA.

Firewall 2

A logical component protects whole PKI Management module used for communication towards other related components of the SM.

PKI management shall be located on physically separated server.

6.1.5.2. Keys&Certificates Database

Keys&Certificates database represents central storage for keys, certificates and CRLs used for PKI Management, based on LDAP service.

The component is protected with firewall and shall be connected to PKI module only.

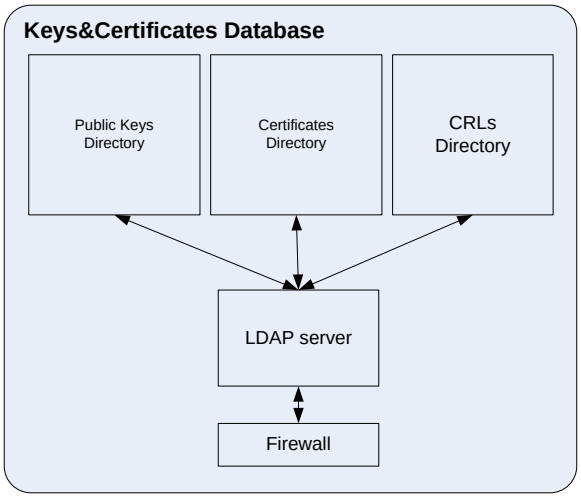

Figure 16. Keys&Certificates database

6.1.6. Use case

Below mentioned use case shows role of SM as well as other components within HBB-NEXT architecture.

6.1.6.1. Preconditions:

- Lisa - owns HBB-NEXT enabled mobile device or tablet,
- Tom - is Lisa's brother. He has different interests than his sister and has no mobile device,
- They have at home HBB-NEXT enabled device – TV&STB (it may be a single device). This device is connected to the Internet,
- Lisa's and Tom's parents – they are setting rules on TV&STB HBB-NEXT device for their kids
- All devices are connected to the Internet (for the purposes of this scenario),
- Mia – Lisa's friend, she is using other HBB-NEXT provider (other HBB-NEXT domain)
- Profiles of Lisa and Tom will be pre-filled (for demo only). Both are using one HBB-NEXT provider

- Lisa is using HBB-NEXT service and her tablet is connected to internet
- Her profile shows that she has brother Tom and also Tom's photo (demo case – filled before)
- Lisa will use applications that demonstrates the applicability of multifactor-authentication, concretely:
    - *EPG* – Lisa's customized program guide – Level 1 - sufficient identification is face / voice recognition
    - *Facebook – Level 2* – say a passphrase or password if nobody else is in the room (this will be always monitored) OR PIN/Passcode with using of remote control if someone else is in the room (saying of passphrase will be disabled)
    - *Messaging – Level 2* - same conditions as for Facebook app
    - *Shopping – before payment* - say a passphrase or PIN. The system shall also look for someone else in the room (this could be detected by MM (multi-modal interface) and IDM service has announced Shopping)
    - *Shopping- payment phase – Level 3* - Lisa wants to pay selected items in the shop - SMS confirmation code or pupil recognition is required (it must be discussed with Gregor)
- same applications will be available through TV&STB

6.1.6.2. Scenario:

1. Lisa is in the living room and has its own programs and videos available through the VoD and EPG menu and she is chatting with Mia (chat service is triggered via MM with the profile of Lisa).
2. Tom is coming and is identified by camera on TV&STB. Screen shows – "Tom is coming" Now, EPG expands by their common preferences (as an example). Running IM conversation between Lisa and Mia is moved to the background and is now

available on the Lisa's tablet only. If Lisa wants to continue with the chatting, she will be asked to enter a PIN via the remote control.

3. Tom on launching Facebook app on the TV&STB - because Lisa was there before, the system knows that no notifications are necessary, and so allows him to sign in (via PIN/passcode) even presence of Lisa.

4. Lisa and Tom are in the living room debate on video that they saw. System offers them the opportunity to see this video with similar content because it was discerned from conversation (Given that this option is enabled Tom and Lisa's profile, and be able to ban permanently)

5. Tom in on the Facebook / (or other app) and wants to buy a tree for his virtual farm. It is forbidden to him because lack of his age. Purchase can be enabled by parents only, thus they are receiving alert message (only parents and persons authorized by them shall be able to allow their children to buy something.

End of demo.

6.1.7. Sequence Diagram

This chapter describes communication of all components involved in the prototype use case, described in the chapter above. We use sequence diagram as self-explanatory form of description.

Main simplification used here:

- We suppose that Security Manager is already pre-provisioned with respective data, i.e. IDs, pass phrase, trust level data, …

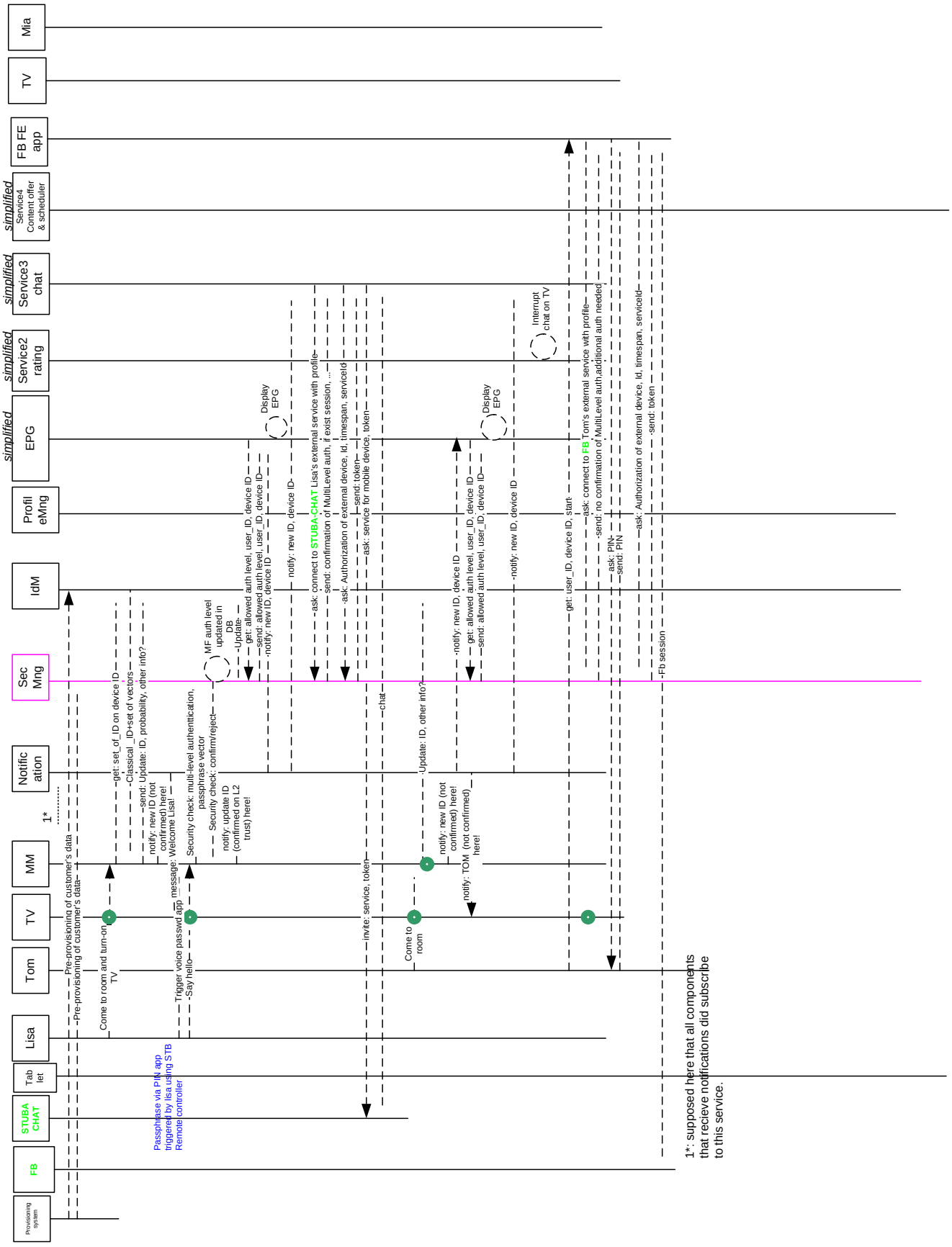

Figure 17. Sequence Diagram

6.1.8. API description

6.1.8.1. Multifactor authentication level of given user

Function:	**Retrieve multifactor authentication level of given user**		
Method:	GET	**Resource:**	users/{*userid*}
Parameters:			
user_ID			
[device_ID]			

Function:	**Check multifactor authentication level of given user**		
Method:	GET	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[device_ID]			

Function:	**Update multifactor authentication level of given user**		
Method:	PUT	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[device_ID]			

Function:	**Delete multifactor authentication level of given user**		
Method:	DELETE	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[device_ID]			

6.1.8.2. User related security data

Function:	**Add user related security data**		
Method:	PUT	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[device_ID]			
Access_code			

Function:	**Modify user related security data**		
Method:	PUT	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[device_ID]			
Access_code			

Function:	**Delete user related security data**		
Method:	DELETE	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[device_ID]			
Access_code			

6.1.8.3. Security check

Function:	**Verify the security check level**		
Method:	GET	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[device_ID]			

6.1.8.4. API security management

Function:	**Add new entity for its registration in SM module**		
Method:	ADD	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[entity_ID]			

Function:	**Modify existing entity in SM module**		
Method:	PUT	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[entity_ID]			

Function:	**Delete existing entity in SM module**		
Method:	DELETE	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[entity_ID]			

Function:	**Revoke of certificate of existing entity in SM module**		
Method:	PUT	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[entity_ID]			

Function:	**Generate new certificate for existing entity in SM module**		
Method:	GET	**Resource:**	/users/{*userid*}
Parameters:			
user_ID			
[entity_ID]			

7. Conclusion

In this document the first outputs of WP3 have been described: Applications like voice identifications, 2D face recognition and Reputation framework have been demonstrated within first beta versions. Others, like Identity manager, Security Manager and Profile Manager have been described in details from system and design point of view. In the next year of project all applications will be enhanced and step by step integrated into one demo application. Faster classification methods will be implemented and tested so the recognition process would be able to run for an infinite time interval. New decision taking algorithms will be suggested to meet multi speaker particularities with a possibility of detecting unknown speaker (not being recorded in the database) by employing some confidence criteria or by creating general speaker model. There is an on-going research yielding to implementation of a gesture recognition system based on a pre-defined database. We plan to build a more sophisticated system which will recognize not only static gestures but also dynamic gestures and their combination. The preliminary documentation to the present applications has been described in this document.

8. References

[1] D2.2 System-, Service-, and User Requirements", http://www.hbb-next.eu/index.php/documents

[2] Oravec Miloš, Mazanec Ján, Pavlovičová Jarmila, Pavel Eiben,FedorLehocki: Face Recognition in Ideal andNoisyConditionsUsing Support Vector Machines, PCA and LDA, Face Recognition (Ed. Milos Oravec), ISBN: 978-953-307-060-5, IN-TECH, 2010

[3] X.Tan, S.Chen, Z.-H. Zhou, and F. Zhang, Face Recognition from a Single Image per Person: A Survey, Pattern Recognition, 39(9), pp.1725-1745. 2006.

[4] Oravec,M., Pavlovičová,J., Mazanec,J., Omelina,Ľ., Féder,M., Ban,J.: Efficiency of Recognition Methodsfor Single Sample per Person Based Face Recognition, chapter in monograph Reviews, Refinementsand New Ideas in Face Recognition (Ed. Peter M. Corcoran), ISBN 978-953-307-368-2, IN-TECH, Croatia, 2011, pp. 181-206

[5] R. Verschae, J. Ruiz-del-Solar, and M. Correa, "Face recognition in unconstrainedenvironments: a comparativestudy," in Proceedingsofthe Workshop on Faces in Real-Life Images: Detection, Alignment, and Recognition (ECCV '08), pp. 1–12, Marseille, France, October 2008

[6] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-Learning-Detection," Pattern Analysis and Machine Intelligence, 2011.

[7] OASIS Open Reputation Management System (ORMS) Draft Version 0.1, availableathttp://wiki.oasis-open.org/orms/WorkingDraft

[8] D5.2 DESIGN AND PROTOCOL: Multimodal Interface and Context Aware Recommendation Engine; http://www.hbb-next.eu/index.php/documents

[9] D3.1 ANALYSIS: State of The Art on Identity, Security and Trust, http://www.hbb-next.eu/index.php/documents

[10] D6.1.1 Initial Version of the HBB-NEXT System Architecture, http://www.hbb-next.eu/index.php/documents

[11] D2.1 Usage Scenarios and Use Cases, http://www.hbb-next.eu/index.php/documents

9. Abbreviations

OASIS ORMS	OASIS Open Reputation Management Systems
AKA	Authentication and Key Agreement
AP	Aggregation Proxy
BSS	Business Support System
CDR	Charging Data Record
CF…	Call Forwarding…
CH	Communication Hold
CLF	Connectivity Session Location Repository Function
CLI	Command Line Interface
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
HLD	High Level Design
HSS	Home Subscriber Server
ILOM	Integrated Lights Out Management
LI	Lawful Intercept
LLD	Low Level Design
QoS	Quality of Service
RTP	Real-time Transport Protocol
UA	User Agent

10. Annex A

The Annex A contains examples of parametric files mentioned in the section 2.1.3.

An example of config_param.txt:

```
vzorkovacia_frekvencia 22050            #sampling frequency
dlzka_ramca 20                          #frame length
posun_ramca 10                          #frame shift
minimalna_frekvencia 250                      #min. Frequency
maximalna_frekvencia 8000               #max. frequency
pocet_filtrov 28                        #number of filters
dct_min_coef 1                          #first DCT coefficient
dct_max_coef 20                         #last DCT coefficient
prah_min_energie 0.02                   #minimal energy threshold
najmensi_absolutny_vykon 130            #minimal power
preemfaza -0.97                         #preemphase
CepstralMeanSubtraction 0               # true or false
AdaptacnyKeficientPriemeruCepstra 0.99863     #adaptation parameter for
mean
zlozka data\                            #directory of the data
```

An example of config_kd_knn.txt:

```
PocetSusedov 4                          #number of neighbours
MaxPocetVektorovNaList 160              #number of vector per leaf
VahovanyKNN 1                           #weighted or classical KNN
LokalnaVzdialenost Euclid               #local distance type
RozhodnutieVahovane 1                   #majority rule or weighted
decision
DlzkaNahravkyTrain 14                   #length of training recording
MinDlzkaNahravkyTrain 5                       #min. length of training
recording
DlzkaBuffraRozpoznavanie 700            #length of a recording buffer
PocetBuffrovRozpoznavanie 4             #number of buffers
MinPomerReci 0.5                        #min. ratio of detected speech
DatovySubor knn_vektory.knn             #file name of sample database
```

```
SuborDatabazyMien databaza_hovoriacich.txt    #file name with speaker
names
```