



WP42 - Universal Authentication Service

D42.2 - Interface and Module Specification and Documentation

Document Identification	
Date	11.12.2013
Status	Final
Version	1.1

Related SP / WP	SP4 / WP42	Document Reference	D42.2
Related Deliverable(s)	42.1	Dissemination Level	PU
Lead Participant	TUD	Lead Author	Moritz Horsch
Contributors	Moritz Horsch (TUD), Monika Drabik (CA), Alfredo Rial (IBM), Detlef Hühnlein (ECS) Johannes Schmölz (ECS)	Reviewers	Omar Almousa (DTU)

This document is issued within the frame and for the purpose of the FutureID project. This project has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318424

This document and its content are the property of the FutureID Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the FutureID Consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the FutureID Partners.

Each FutureID Partner may use this document in conformity with the FutureID Consortium Grant Agreement provisions.



Abstract

The FutureID project builds a comprehensive, flexible, privacy-aware and ubiquitously usable identity management infrastructure for Europe. It integrates existing eID technologies, trust infrastructures, emerging federated identity management services, and modern credential technologies. It creates a user-centric system for the trustworthy and accountable management of identity claims.

The Broker Service provides an interface for service providers to connect to the FutureID infrastructure. Therefore, service providers are able to easily integrate arbitrary eID services using various user credentials. In the Claims Transformer Mode the Broker includes the Universal Authentication Services (UAS) which in turn performs the authentication of the users. The UAS will support a wide range of authentication protocols implemented by the various authentication tokens deployed across Europe.

Authentication protocols are described in the Authentication Protocol Specification (APS) language, which makes it possible to support arbitrary protocols in a very efficient manner. The APS descriptions of the authentication protocols in turn refer to appropriate Basic Services. Basic Services provide a set of common functionality like cryptographic and smart card services, which are needed by the majority of the authentication protocols.

The UAS acts as an Authentication Service and is capable of authenticating users. The UAS encompass two interfaces, the Authentication-Interface for the communication with the users (i.e., FutureID clients) and the Universal Service Interface to communicate with the Broker. The core component of the UAS is the Job Execution Environment, which runs arbitrary authentication protocols specified in the APS language.

This document describes the individual components of the UAS and specifies the interfaces. Further, it provides a brief overview the Basic Services and the Job Execution Environment.

Document name:	Interface and Module Specification and Documentation				Page:	1 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

Document Information

History

Version	Date	Author	Changes
0.1	10.05.2013	Moritz Horsch	Created document structure
0.11	20.05.2013	Moritz Horsch	Added introduction section
0.12	30.05.2013	Moritz Horsch	Added FutureID infrastructure section
0.13	12.06.2013	Moritz Horsch	Added Universal Authentication Service section
0.14	24.06.2013	Moritz Horsch	Added requirements section
0.15	08.07.2013	Moritz Horsch	Added Universal Service Interface section
0.2	01.08.2013	Monika Drabik	Added Basic Services section
0.3	08.08.2013	Alfredo Rial	Added Authentication Interface section
0.4	15.08.2013	Detlef Hühnlein	Added Job Execution Environment section
0.5	16.08.2013	Moritz Horsch	Added conclusion section
0.6	20.09.2013	Johannes Schmölz	Draft of Job Execution Environment section
0.7	23.09.2013	Detlef Hühnlein	Internal Review
0.8	20.10.2013	Moritz Horsch	Added changes from Omar Almousa
1.0	28.10.2013	Moritz Horsch	Finalized deliverable
1.1	11.12.2013	Moritz Horsch	Updated terms to new Reference Architecture

Document name:	Interface and Module Specification and Documentation				Page:	2 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

Table of Contents

Abstract	1
Document Information	2
Table of Contents	3
Table of Figures	5
1. Introduction	6
1.1 Scope	7
1.2 Outline	7
1.3 Terminology	7
1.3.1 Key Words	7
1.3.2 Abbreviations and Notations	7
1.3.3 Terms	8
2. FutureID Infrastructure	11
3. Requirements	13
4. Universal Authentication Service	14
4.1 Authentication Interface	16
4.2 Universal Service Interface	16
4.3 Authentication Protocol Specification	16
4.4 Control	16
4.5 Configuration	16
4.6 Job	16
4.7 State Management	16
4.8 Job Execution Environment	17
4.9 Basic Service	17
4.10 Additional Services	17
5. Job Execution Environment	18
6. Basic Services	21
7. Universal Service Interface	23
7.1 Functionality	23
7.2 Operational Environment	23
7.3 Functions	23
7.3.1 Authenticate	23
8. Authentication Interface	27
8.1 Functionality	27
8.2 Operational Environment	27
8.3 Functions	27
8.3.1 StartPAOS	28
8.3.2 DIDAuthenticate	31
8.3.3 Transmit	34
9. Conclusion	37
10. Bibliography	38

Document name:	Interface and Module Specification and Documentation				Page:	3 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

A. Appendix	40
A.1 Policy	40
A.2 RequestedAttributeType	40
A.3 Attribute	40

Document name:	Interface and Module Specification and Documentation				Page:	4 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

Table of Figures

Figure 1: FutureID infrastructure	11
Figure 2: Flowchart Universal Authentication Service	14
Figure 3: Universal Authentication Service Architecture	15
Figure 4: Structure of an APS-specific JAR file	19
Figure 5: The Job Execution Environment and corresponding interface components	19
Figure 6: Schematic flow of messages between the Broker Service, the UAS and the Client	20
Figure 7: Overview of Basic Services	22
Figure 8: Universal Service Interface	23
Figure 9: Authentication Interface	27

Document name:	Interface and Module Specification and Documentation				Page:	5 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

1. Introduction

The FutureID project builds a comprehensive, flexible, privacy-aware and ubiquitously usable identity management infrastructure for Europe. It integrates existing eID technologies, trust infrastructures, emerging federated identity management services, and modern credential technologies. It creates a user-centric system for the trustworthy and accountable management of identity claims.

The Broker Service provides a common interface for service providers to connect to the FutureID infrastructure and hereby easily integrate arbitrary eID services using various user credentials. In case of the Claims Transformer Mode the Broker Service includes the Universal Authentication Services (UAS) to perform user authentication.

As the existing eID cards, eHealth cards, and eSign cards already support a large variety of different authentication protocols and it is expected that future authentication tokens will support other credentials and authentication protocols, it would be close to impossible to implement all required protocols using a conventional approach, because this would require a specialized program module for each authentication protocol.

In order to solve this problem, protocols are described in the Authentication Protocol Specification (APS) language which makes it possible to support arbitrary authentication protocols in a very efficient manner. The APS descriptions of the authentication protocols in turn refer to appropriate Basic Services, such as cryptographic primitives or smart card commands according to ISO/IEC 7816 [1]. As the different authentication protocols are all composed of a rather limited set of Basic Services, the problem of supporting arbitrary authentication protocols is reduced to providing this limited set of basic functionality and providing appropriate APS descriptions for the different authentication protocols.

The UAS acts as an Authentication Service and is capable of authenticating users. In the long run, the UAS should be also capable to act as a signature and credential service. The UAS encompasses two interfaces, one for the communication with the users and one for the communication with the Broker Service (or similar services). The core component of the UAS is the Job Execution Environment, which runs arbitrary authentication protocols specified in the APS language.

To enhance the UAS with additional functionalities like signature creation, credential issuance, or verification services, which may not supported by the APS language, the UAS can be extended with further software components.

Document name:	Interface and Module Specification and Documentation				Page:	6 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

1.1 Scope

The scope of this document is to describe the design and the architecture of the Universal Authentication Service. In particular, it specifies the Authentication Interface and the Universal Service Interface, which provides the communication interface to the Broker Service (or similar services) and the FutureID Client, respectively. Furthermore, this document provides a brief overview of the Basic Services and the Job Execution Environment of the Universal Authentication Service. The Basic Services provide an essential set of services for authentication protocols like a cryptographic service and a smart card service. The Job Execution Environment provides a runtime environment to execute APS-based authentication protocols. The Basic Services and the Job Execution Environment are specified and implemented in T42.4 and T42.5, respectively.

1.2 Outline

This document is structured as follows: Section 2 gives a short overview of the architecture of the FutureID infrastructure. Section 0 summaries the requirements of the Universal Authentication Service as defined in D42.1 [2]. The Universal Authentication Service is briefly described in Section 4. Details about the individual components are presented in the following sections. Section 5 describes the Job Execution Environment and Section 6 describes the Basic Services. The interfaces of the Universal Authentication Service, namely the Universal Service Interface and the Authentication Interface are specified in Section 7 and 8, respectively. The document is concluded in Section 9.

1.3 Terminology

This section describes the meaning of the most important words used in this document to precise communication and to provide a common understanding.

1.3.1 Key Words

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [3].

1.3.2 Abbreviations and Notations

This section itemizes the abbreviations and notations used in this document.

ABC	Attribute-based Credential
API	Application Programming Interface
APS	Authentication Protocol Specification
BS	Basic Service
BS	Broker Service
CV	Credential Verifier
DID	Differential Identity
eHealth	Electronic Health

Document name:	Interface and Module Specification and Documentation				Page:	7 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

eID	Electronic Identity
eSign	Electronic Signature
FS	Federation Service
HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol
JEE	Job Execution Environment
OCSP	Online Certificate Status Protocol
PAOS	Reverse HTTP Binding for SOAP
PEPS	Pan European Proxy Services
SAM	Security Access Module
SAML	Security Assertion Markup Language
SP	Service Provider
TLS	Transport Layer Security
TS	Trust Service
UAS	Universal Authentication Service
USI	Universal Service Interface
WSDL	Web Services Description Language
XML	Extensible Markup Language

1.3.3 Terms

This section provides descriptions for selected terms that are used in this document. The descriptions are taken from the FutureID terminology [4].

Access Control	The process of restricting access to resources to privileged entities. Access control policies determines who can view access what resources, under which conditions, what they can do with it, and the type of device which may be used.
Assurance	A measure of certainty that a statement or a claim is true.
Assurance Level	A relative measure (e.g., low, medium, high) of the strength of assurance that can be placed in an identity claim. A lower level of assurance means less certainty in an identity claim, while a higher level of assurance indicates a higher degree of certainty. Assurance levels are established through authentication events and are dependent on a number of factors including: the rigorousness of the original registration and identity proofing process; the strength of the presented credential; the authentication event itself (i.e., successful log-on, in person verification); and, the underlying infrastructure/environment within which the authentication event occurs.
Attribute	An attribute is a physical or abstract named property belonging to an entity. An attribute typically has a value.
Authentication	Authentication is the process of corroborating a claimed set of attributes or facts with a specified, or understood, level of confidence.
Availability	Availability can be described as the property of being accessible and useable upon demand by an authorized entity. In the context of service level agreements, availability generally refers to the degree to which a system may suffer degradation or interruption in its service to the

Document name:	Interface and Module Specification and Documentation				Page:	8 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

customer as a consequence of failures of one or more of its parts.

Credential	<p>i. An identifiable object that can be used to authenticate the claimant is what it claims to be and authorize the claimant's access rights.</p> <p>ii. Data that is transferred to establish the claimed identity of an entity.</p> <p>iii. The private part of a paired Identity assertion (user-id is usually the public part). The thing(s) that an Entity relies upon in an Assertion at any particular time, usually to authenticate a claimed Identity. Credentials can change over time and may be revoked. Examples include: a signature, a password, a drivers licence number (not the card itself), an ATM card number (not the card itself), data stored on a smart-card (not the card itself), a digital certificate, a biometric template.</p>
Electronic Signature	Data in electronic form which is attached to or logically associated to other electronic data and serves as a method of authentication. From a legal perspective, an electronic signature is not necessarily considered equivalent to a handwritten signature. When it meets a number of conditions, it can be put on par with a handwritten one.
Entity Authentication	Entity authentication is a process that provides assurance of the claimed identity of an entity, as it corroborates the (claimed) partial identity of an entity and a set of its observed attributes.
Federation	Federation is the process of establishing a relationship between two or more entities or an association of entities comprising any number of service providers, identity providers or other entities. As a noun, a federation refers to a group of entities have an agreement to create and process federated identities.
Identification	<p>i. Identification in terms of registration: identification as the process of using claimed, observed or assigned attributes of an entity to establish a partial identity of that entity.</p> <p>ii. Identification as entity authentication: identification as the verification of the link between an entity and the asserted (partial) identity.</p> <p>iii. Identification in terms of identifiability: identification as the process of "individualizing" a particular entity within a set of subjects.</p>
Identifier	An identifier can be defined as a non-empty set of attributes of an entity which uniquely identifies the entity within one or more specific contexts or sectors.
Identity	The identity of an entity is the dynamic collection of all of the entity's attributes. An entity has only one identity. As such, the identity of an entity this is more a fluid and evolving ("philosophical") concept, rather than a practical one. An entity has only one identity, but it is neither possible nor desirable for an information system to gather all the attributes of a specific entity. Information systems focus on a specific subset of relevant attributes, commonly referred to as "partial identities".
Integrity	Quality which implies that the items of interest (facts, data, attributes etc.) have not been subject to manipulation by unauthorized entities.
Interoperability	The ability of independent systems to exchange meaningful information

Document name:	Interface and Module Specification and Documentation				Page:	9 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

and initiate actions from each other, in order to operate together to mutual benefit. In particular, it envisages the ability for loosely-coupled independent systems to be able to collaborate and communicate; the possibility of use in services outside the direct control of the issuing assigner.

Level of Assurance	See Assurance Level.
Performance	Performance is the speed measured and perceived by individual users of the system (latency), or the speed perceived by the total infrastructure, from the point of view of the server (throughput, number of accepted transactions per second).
Policy	<p>A policy is one or more definite goals, courses or methods of action to guide and determine present and future decisions.</p> <p>Policies are implemented or executed within a particular context (such as policies defined within an organization or business unit) or across contexts (e.g., in the case of an identity federation).</p> <p>Common examples of such policies are security policies, privacy policies, access control policies, registration policies etc.</p>
Reliability	The term reliability is used to denote any system that consistently produces the same results, preferably meeting or exceeding its specifications.
Revocation	The act (by someone having the authority) of annulling something previously done.
Service	A digital entity comprising software, hardware and / or communications channels that interacts with subjects.
Service Provider	Entity that provides services to other entities.
Token	A token is any hardware or software component that contains credentials related to attributes. They may take any form, ranging from a digital data set to smart cards or mobile phones. They can be used for both data/entity authentication and authorization purposes.
Verification	The process or an instance of establishing the truth or validity of something.

Document name:	Interface and Module Specification and Documentation				Page:	10 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

2. FutureID Infrastructure

This section provides a brief overview of the architecture of the FutureID infrastructure to point out the task and the role of the Universal Authentication Service.

FutureID builds a comprehensive and ubiquitous identity infrastructure for Europe including existing eID technologies and building on well-established standards. The FutureID infrastructure is illustrated in Figure 1 and comprises the User Platform, Service Provider, and FutureID Infrastructure Services. In detail, the FutureID infrastructure includes a Broker Service (BS), Federation Service (FS), Trust Service (TS), and a Credential Verifier (CV). Please note that multiple instances of the Broker Service, Federation Service, and Credential Verifier may exist.

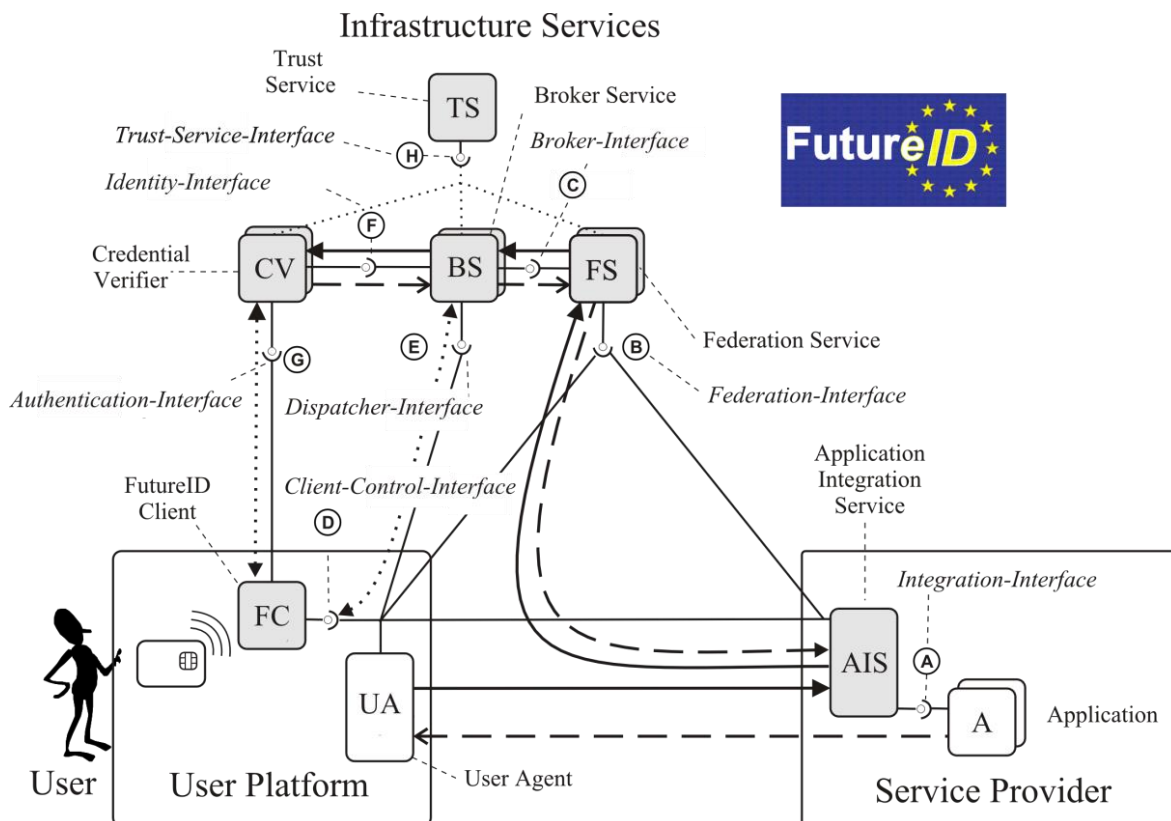


Figure 1: FutureID infrastructure

Document name:	Interface and Module Specification and Documentation				Page:	11 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

The Broker Service provides a common interface for service providers to connect to the FutureID infrastructure and hereby easily integrate arbitrary eID services using various credentials. The Broker Service provides two operation modes: First, in the Dispatcher Mode the Broker Service determines an appropriate existing credential verifier for the service provider, which is capable and suitable to authenticate the user. Second, in the Claims Transformer Mode the Broker Service includes the Universal Authentication Service as an internal credential verifier to perform user authentication and to issue session and attribute-based credentials. The Dispatcher Mode might cause message overhead and needs multiple redirects to enable the authentication of the user. To provide an optimised message flow and to enable the Claims Transformer Mode the Broker Service and the Universal Authentication Service takes care about the user authentication. Moreover, the Universal Authentication Service provides a novel approach to integrate arbitrary authentication protocols. The Universal Authentication Service operates a Job Execution Environment to run authentication protocols that are described in the Authentication Protocol Specification (APS) language, which allows describing authentication protocols in an abstract and convenient manner.

The Federation Service provides a federation-dialect-specific interface for service providers to get integrated into the FutureID infrastructure. It receives the authentication requests from the service provider's Application Integration Service and accesses the Broker Service through the Broker Interface. Please note that a Broker usually runs multiple Federation Services, in detail, one for each supported federation dialect like SAML (Security Assertion Markup Language).

The Credential Verifier verifies user credentials and issue session or attribute-based credentials. Such a Credential Verifier can be internal or external to the Broker Service. For instance, the Universal Authentication Service acts as an internal CV. The Trust Service is attached to the Broker Service and provides a comprehensive repository for trusted certificates and services as well as other trust related information (cf. WP43). Further the Trust Service is capable of evaluating certificates and assertions.

The Applications (A) operated by service providers encapsulate services that are integrated into the FutureID infrastructure using the Application Integration Service (AIS) and host services that are consumed by the users.

The FutureID Client (FC) is an eID application that allows end-users to use arbitrary credentials and authentication methods for services within the FutureID infrastructure. It is developed in SP3 and implements among other things the user interaction as well as the transport and security protocols. In detail, it includes the Interface Device Service (WP31), eID Services (WP32), eSign Services (WP33), User Interface (WP34), Trustworthy Client Platform (WP35), Browser Integration (WP36), and Client Testbed (WP37).

Document name:	Interface and Module Specification and Documentation				Page:	12 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

3. Requirements

This section summarizes the requirements for the Universal Authentication Services (UAS) as described in D42.1 [2].

The UAS SHOULD provide generic interfaces, which support the transport of messages via a predefined set of message bindings. In particular, the UAS MUST provide an Authentication Interface and SHOULD support at least the `StartPAOS`, `DIDAuthenticate`, and `Transmit` messages as specified in BSI-TR-03112 [5] and ISO/IEC 24727 [6]. The Authentication Interface can be used by the FutureID Client to perform a user authentication. In addition, the UAS MUST provide a Universal Service Interface and should support at least the `Authenticate` function as specified in CEN/TS 15480 [7]. This interface provides the gateway for the Broker Service (cf. WP41). Further, the UAS SHALL be extendable with further interfaces.

The UAS MUST also support SOAP [8] and PAOS [9] for message transport and Hardware Security Modules (HSM) for the secure storage and application of cryptographic keys.

The UAS MUST provide a reasonable set of Basic Services and MUST be able to execute APS-specified authentication protocols. Furthermore, it SHOULD provide a reasonable error handling, including sufficiently detailed error messages in both machine-readable and human-readable format.

The UAS MUST support the protocol-specific and privacy-specific requirements of the eID solutions and in particular it SHOULD NOT store user attributes in a permanent manner. Furthermore, the UAS MUST protect processed personally identifiable information.

The UAS SHOULD provide an interface for forensics, audit and transparency requests, providing information on both its operations and processes and on all subsequent UAS events related to a certain (set of) transaction(s). It MUST be ensured that this interface is only available for authorized entities and that access to this interface is logged.

The UAS SHOULD create and maintain appropriate logs of all events of importance in the context of UAS transactions. The integrity of such event logs MUST be ensured and only authorized entities SHALL have access to these logs. Further, the UAS SHOULD provide means for redaction in case an individual claims its rights to deletion or correction of personal data.

Document name:	Interface and Module Specification and Documentation				Page:	13 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

4. Universal Authentication Service

The Broker Service provides a common interface for service providers to connect to the FutureID infrastructure and hereby easily integrate arbitrary eID services using various user credentials. The Broker Service provides two operation modes: First, in the Dispatcher Mode the Broker Service determines an appropriate existing authentication service for the service provider, which is capable and suitable to authenticate the user. Second, in the Claims Transformer Mode the Broker Service includes the Universal Authentication Services (UAS) to perform user authentication and to issue session and attribute-based credentials (ABC).

In order to provide an optimised message flow and enable the Claims Transformer Mode, the UAS will support a wide range of authentication protocols implemented by the various authentication tokens deployed across Europe. In comparison to the Dispatcher Mode, through which the Broker Service just forwards requests to existing Identity Providers, the Claims Transformer Mode enables session credentials and attribute-based credentials issued by the UAS.

Authentication protocols are described in the Authentication Protocol Specification (APS) language, which makes it possible to support arbitrary protocols in a very efficient manner. The APS descriptions of the authentication protocols in turn refer to appropriate Basic Services, such as cryptographic primitives or smart card commands according to ISO/IEC 7816 [1]. As the different authentication protocols are all composed of a rather limited set of Basic Services, the problem of supporting arbitrary authentication protocols is reduced to providing this limited set of basic functionality and providing appropriate APS descriptions for the different authentication protocols.

The UAS acts as an Authentication Service and is capable of authenticating users. In the long run, it should be also capable to act as a signature and credential service. The UAS encompasses two interfaces, one for the communication with the users and one for the communication with the Broker Service or similar services. The core component of the UAS is the Job Execution Environment, which runs arbitrary authentication protocols specified in the APS language.

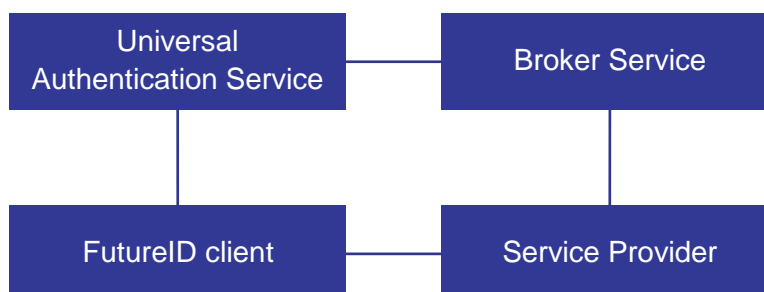


Figure 2: Flowchart Universal Authentication Service

Document name:	Interface and Module Specification and Documentation				Page:	14 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

The interaction between the Universal Authentication Service, the Broker Service, the Service Provider, and the FutureID Client is illustrated in Figure 2. The client contacts the Service Provider and gets forwarded to the Broker Service to perform user authentication. The Broker Service next selects the appropriated Authentication Service, in this case the Universal Authentication Service. Now the Client is redirected to the UAS and the authentication is performed. Finally, the Client will be sent back to the Service Provider and access to the service is granted.

The architecture of the UAS is illustrated in Figure 3 and the individual components are described in the following subsections.

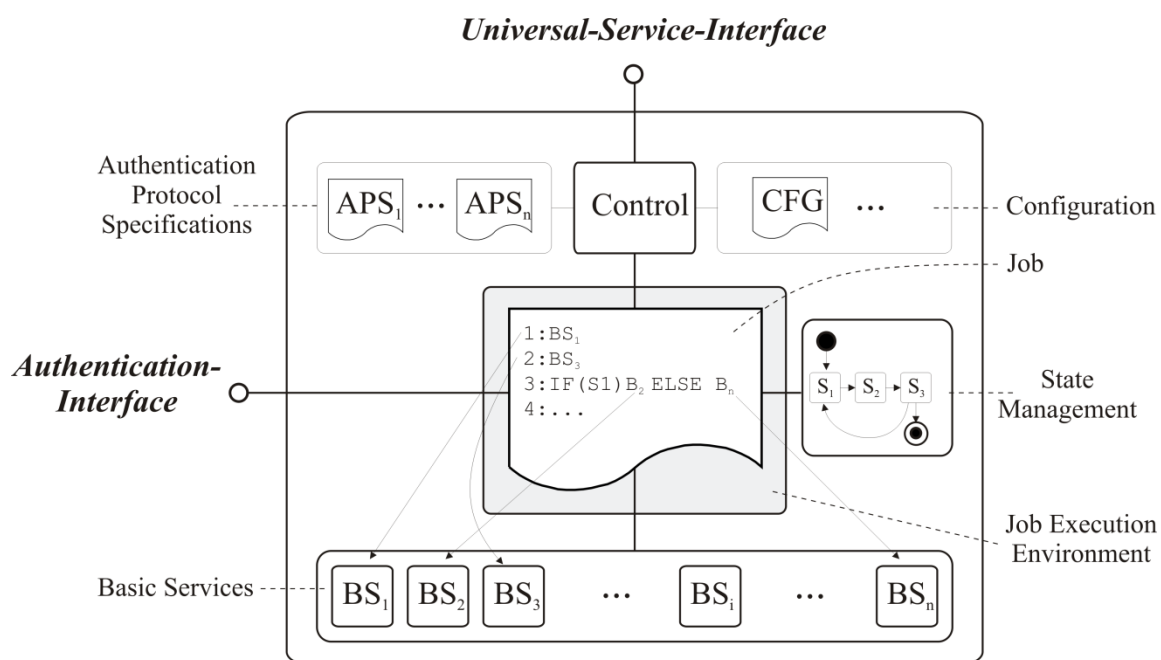


Figure 3: Universal Authentication Service Architecture

As depicted in Figure 3 the UAS supports a set of multiple APS-specified authentication protocols. As a complement to the high-level protocol specifications using the APS language, additional configuration files provide further technical input for the execution of the authentication protocols. As illustrated in Figure 3 the functions of the APS file, i.e., an APS job, map to respective Basic Services.

Document name:	Interface and Module Specification and Documentation				Page:	15 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

4.1 Authentication Interface

The Authentication Interface provides the connection between the UAS and the FutureID Client. The interface specifies functions through which user authentication can be performed. In detail, the functions comprise the establishing of a channel, authenticating the user, and transmitting data between both parties. The interface is defined as a Web Service using WSDL [10] and requires a TLS [11] established channel between the participants. The Authentication Interface is addressed in Section 8.

4.2 Universal Service Interface

The Universal Service Interface defines the interface between the UAS and the Broker Service or similar service providers, which uses the UAS to authenticate specific entities. The interface is also defined as a Web Service and requires a protected and authenticated channel between both the UAS and the calling service. The Universal Service Interface is described in Section 7.

4.3 Authentication Protocol Specification

The Authentication Protocol Specification (APS) language is a formal language which allows describing authentication protocols in an abstract and convenient manner. The APS language is specified in D42.3 [12]. Please refer to this deliverable for further information about the APS language.

4.4 Control

The control component is the core of the UAS and is responsible for managing the interfaces, the protocol execution, the configuration files, and so forth.

4.5 Configuration

A configuration provides additional information to an APS description, which is necessary for the protocol execution.

4.6 Job

A job is a set of APS commands, which describes the execution of an authentication protocol. The APS commands map to individual Basic Services. Details about the APS commands can be found in D42.3 [12].

4.7 State Management

The State Management is responsible for controlling the different steps and states of a job, which is representing an authentication protocol and executed by the Job Execution Environment.

Document name:	Interface and Module Specification and Documentation				Page:	16 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

4.8 Job Execution Environment

The Job Execution Environment is capable of loading and executing APS files, which describes an authentication protocol. The Job Execution Environment is in described in Section 5. For further information please see D42.6 [13].

4.9 Basic Service

Basic Services provide common and elementary functionality for authentication protocols. Such services are, for instance, cryptographic functions like encryption and signature as well as smart card commands according to ISO/IEC 7816 [1]. The Basic Services should be the basic set of functions to support arbitrary authentication protocols. Therefore, the services must be very general, but also combinable to build more complex authentication protocols. The Basic Services are addressed in Section 6. For further information please refer to D42.4 [14].

4.10 Additional Services

In the long run the UAS may become a universal service for authentication, signature creation, credential issuance, verification services, and so forth. The goal should be to realize the complete branch of the desired functionality through the APS language. Nevertheless, this might not be possible in any case, so the UAS should also support integrating services which are not defined in APS.

Document name:	Interface and Module Specification and Documentation				Page:	17 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

5. Job Execution Environment

The Job Execution Environment (JEE) is able to load and execute protocols that are defined in the APS language. Since the JEE is not able to execute the abstract APS directly, it must first be compiled to some kind of executable script code. For this purpose the JEE supports JavaScript to which an APS file is compiled to be executed.

An important feature of the Job Execution Environment is the possibility to access and execute the Basic Services (BS) which provide different functions for common tasks, e.g. to compute a hash value, obtain the status of a certificate via Online Certificate Status Protocol (OCSP) or to create a certain Application Protocol Data Unit (APDU), which is to be sent to an Interface Device (IFD) component, which in turn communicates with a smart card. Furthermore the JEE may be equipped with additional JavaScript-based Extended Services (ES), which combine several calls and hence may be used to provide higher level functionality.

The difference between the Extended Services and the Basic Services is that the Basic Services are available in the Universal Authentication Service per default. Extended Services are usually more light-weight and are shipped together with a specific APS file.

The environment that is needed by a specific protocol can be specified within the manifest file, which is distributed together with the APS file, which specifies the authentication protocol. The Job Execution Environment uses this configuration file to set up a context in which the authentication protocol is executed. Every instance of a protocol has its own context so that the different instances do not interfere with each other.

Protocol descriptions are distributed in APS-specific JAR files and can be loaded by the Job Execution Environment during runtime. Besides the script files that define the protocol and the configuration that is used to set up the context for the protocol, the JAR file can optionally contain additional Extended Services, which can be provided in form of JavaScript files. The structure of an APS-specific JAR file is depicted in Figure 4.

Document name:	Interface and Module Specification and Documentation				Page:	18 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

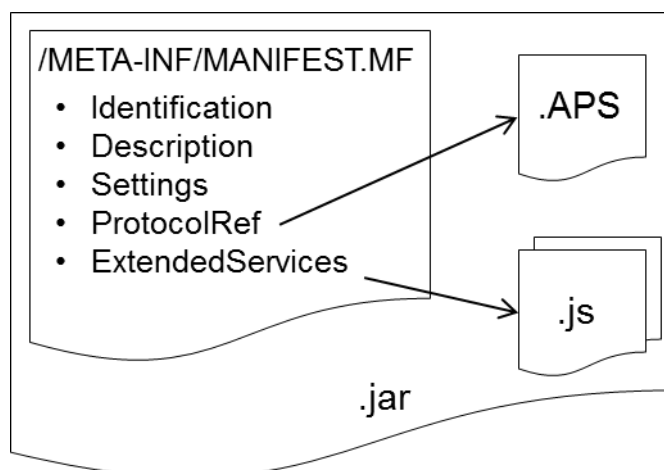


Figure 4: Structure of an APS-specific JAR file

The Job Execution Environment as depicted in Figure 5 is part of the Universal Authentication Service (UAS) and can be accessed through appropriate interface components. These interface components allow to communicate with the Broker Service via the Universal Service Interface and the FutureID Client via the Authentication Interface.

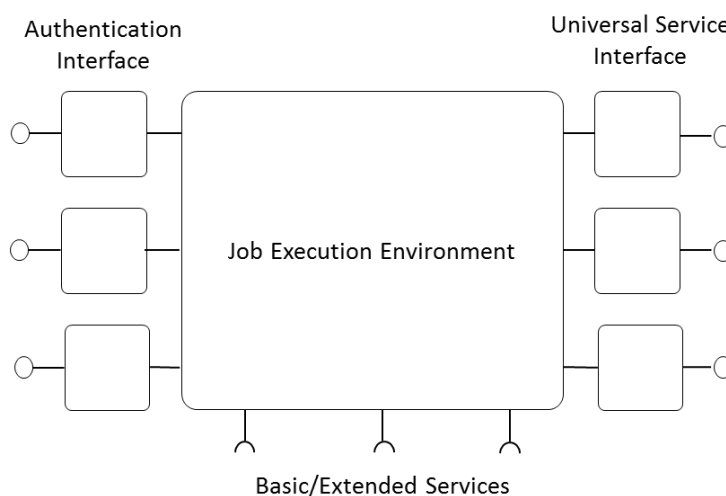


Figure 5: The Job Execution Environment and corresponding interface components

A schematic flow of the messages between the Broker Service, the UAS and the FutureID Client is depicted in Figure 6.

Document name:	Interface and Module Specification and Documentation				Page:	19 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

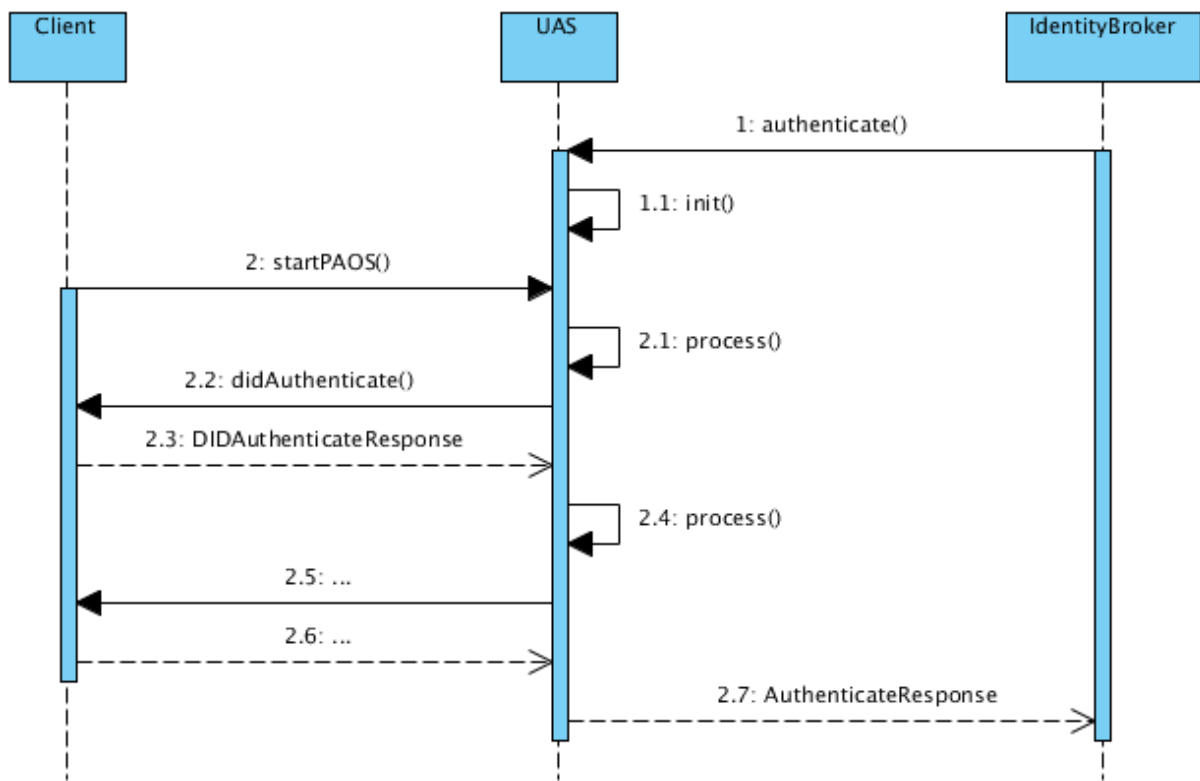


Figure 6: Schematic flow of messages between the Broker Service, the UAS and the Client

Document name:	Interface and Module Specification and Documentation				Page:	20 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

6. Basic Services

Basic Services are combined to implement complex authentication procedures as defined in APS files and executed in an appropriate Job Execution Environment. The Basic Service may comprise cryptographic services and smart card specific commands for example, which may build Application Protocol Data Units (APDU) according to ISO/IEC 7816-4,-8, [1] which can be sent to a local or remote IFD service.

The main function of Basic Service is to provide some sort of object that can be sent to the FutureID Client, an auxiliary service or returned to the Broker Service. These objects may for example contain an APDU or some other XML structure. Although the general APDU format is standardized, different smart cards in practice may use different implementations. In this case the Universal Authentication Service may select appropriate Basic Services, which fit to a specific smart card.

The overview of the Basic Service is depicted on the Figure 7.

The APS file, that defines the authentication protocol, is translated to JavaScript and then executed by the Execution Environment. Following the actions defined in APS the Execution Environment calls concrete functions of a Basic Service.

The smart card related Basic Services SHOULD utilise CardInfo-structures according to ISO/IEC 24727-3 [6] in order to support standardized smart cards in a generic fashion. In addition to this it SHOULD be possible to transmit card specific APDUs directly in order to handle proprietary smart cards and arbitrary authentication protocols.

This applies also to XML Services. Every single XML Service must implement basic functions. The way of implementation may be arbitrary, but coherent with the smart card software.

The objects, which are created with appropriate Basic Services, are then transmitted through a secure channel to the Client site.

The detailed description of the Basic Services interfaces is the main objective of D42.4 [14].

Document name:	Interface and Module Specification and Documentation				Page:	21 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

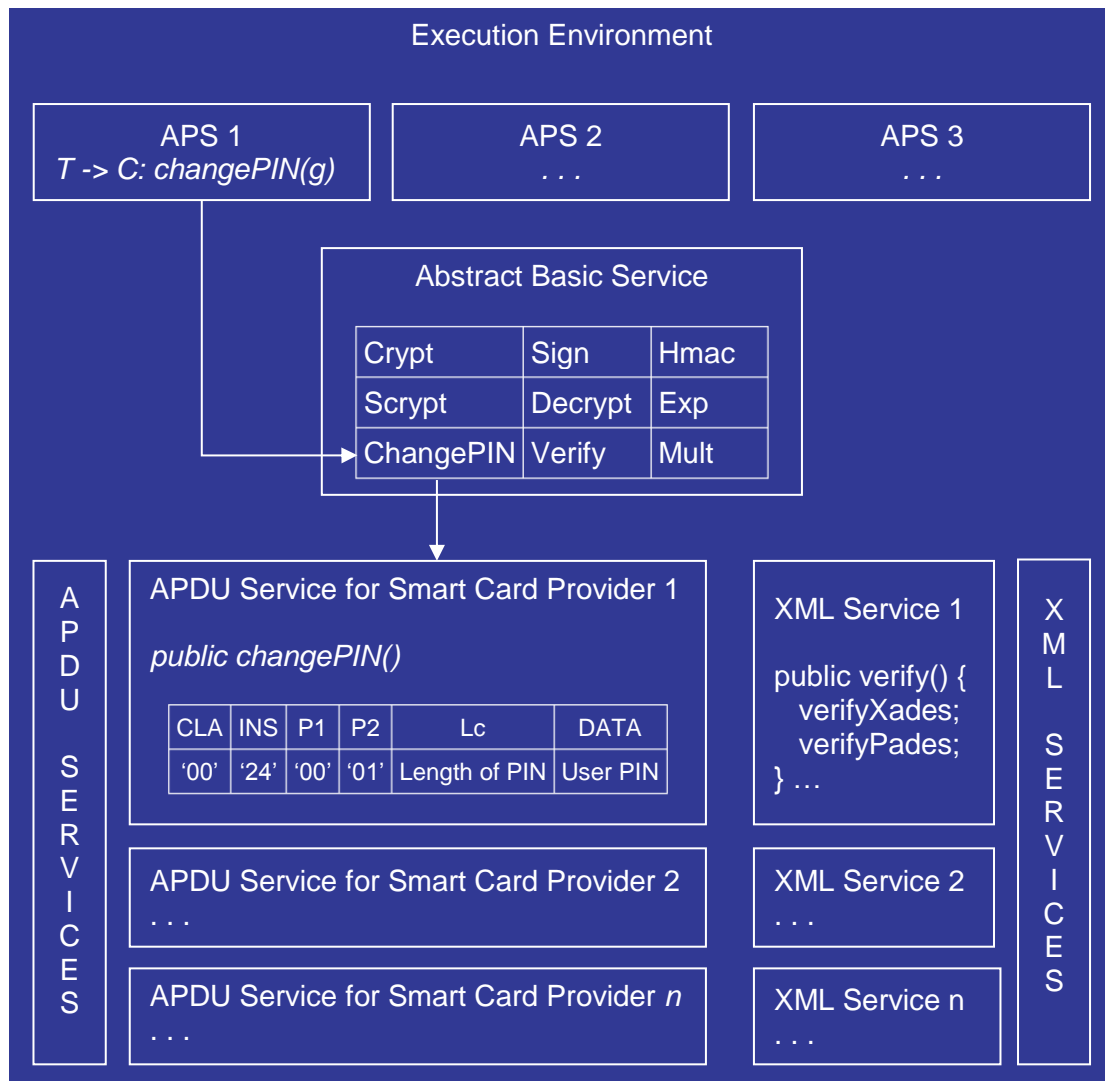


Figure 7: Overview of Basic Services

7. Universal Service Interface

This section describes the functionality, defines the requirements for the operational environment, and specifies the functions of the Universal Service Interface (USI).

7.1 Functionality

The USI provides a common interface to use the UAS as an Authentication Service. As shown in Figure 8 the interface will be used by the Broker Service and finally the Service Providers. Service Providers request a user authentication by the Broker Service and the Broker in turn will use the UAS to perform the authentication.

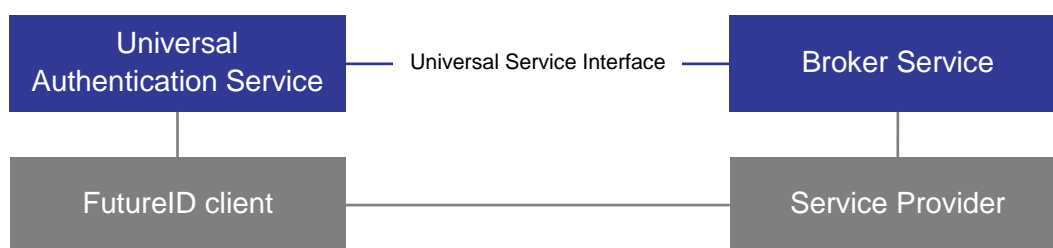


Figure 8: Universal Service Interface

The Broker Service will use the USI to set up the authentication and redirect the user to the UAS. The user in turn will use the Authenticate Interface of the UAS to perform the authentication (cf. Section 8).

7.2 Operational Environment

The USI will be provided as a web service and specified in the Web Services Description Language (WSDL).

The communication between the UAS and the Broker Service or a similar service **MUST** be encrypted. Further, both parties **MUST** be mutual authenticated and the message integrity **MUST** be ensured. The confidentiality of the communication and the mutual authentication can be achieved by using TLS with client authentication [11]. The message integrity can be realised by using WS-Security [15].

7.3 Functions

This section specifies the functions of the Authentication Interface. It comprises the `Authenticate` function.


7.3.1 Authenticate

The `Authenticate` function (cf. CEN 15480 [16], Section 11 and D41.2 [17], Section 7.1) provides a generic function for service providers to request an authentication of a user by the

Document name:	Interface and Module Specification and Documentation				Page:	23 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

UAS. The parameters comprise details about the authentication policy and in particular the requested identity attributes, which should be retrieved during the authentication by the user. Further, it may include some optional information like the profile and technical necessary parameters like a session identifier.

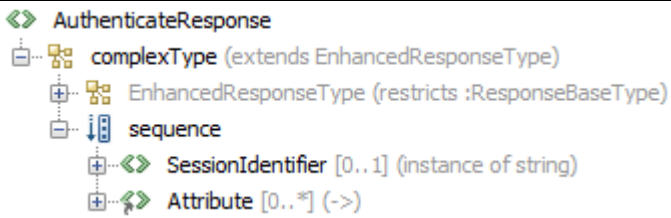
REQUEST

Name	Authenticate
Description	The Authenticate function is used to request an authentication of an entity.
Parameters	 <pre> <> Authenticate complexType (extends EnhancedRequestType) EnhancedRequestType (restricts :RequestBaseType) sequence SessionIdentifier [0..1] (instance of string) Policy [0..1] (->) RequestedAttributes [0..1] (instance of RequestedAttributesType) </pre>
Name	@RequestID
Description	MUST be present in the asynchronous profile (cf. D41.2 [17]) to correlate the request to a previous response, otherwise it MUST be ignored.
Name	@Profile
Description	MAY be used to indicate a specific mode of operation. The synchronous profile SHOULD be used in default (cf. D41.2 [17], Section 7.3.1).
Name	dss:OptionalInputs
Description	MAY contain optional input elements.
Name	SessionIdentifier
Description	Contains the identifier of a session in which the authentication is performed. If the parameter is omitted, the corresponding response message MUST contain a SessionIdentifier element.
Name	wsp:Policy
Description	Specifies the applicable authentication policy. If the parameter is omitted, the authentication service MAY apply a default policy (cf. Section A.1).
Name	RequestedAttributes
Description	Specifies identity attributes which are to be retrieved after the authentication procedure has been performed (cf. Section A.2). If the parameter is omitted, the authentication service SHOULD NOT return any identity attributes. The RequestedAttributes element MAY be part of the Authenticate request. Instead of explicitly specifying the set of requested identity attributes, this element MAY contain an attribute RequestAllDefaults of type anyURI to indicate that a specific set of default attributes is requested. If the RequestedAttributes element neither contains saml:Attribute elements nor the RequestAllDefaults attribute, the authentication service MAY return a set of default attributes.

Document name:	Interface and Module Specification and Documentation	Page:	24 of 40
Reference:	D42.2	Dissemination:	PU
Version:	1.1	Status:	Final

Parameters	Name	saml:Attribute
	Description	This element MAY appear an arbitrary number of times and is used to request a certain identity attribute. Details with respect to the syntax and semantic of this generic element are defined in SAML [18], Section 2.7.3.1.

RESPONSE

Name	AuthenticateResponse												
Description	Return of the Authenticate function.												
Parameters													
Name	@RequestID												
Description	This attribute MUST be present in the asynchronous profile (cf. D41.2 [17]) to correlate the present request to a previous response. In case of the synchronous profile this attribute is ignored.												
Name	dss:Result												
Description	Contains the status information and the errors of an executed action.												
Parameters	<table border="1"> <tr> <td>Name</td> <td>ResultMajor</td> </tr> <tr> <td>Description</td> <td> <ul style="list-style-type: none"> ▪ /resultmajor#ok ▪ /resultmajor#pending ▪ /resultmajor#error ▪ /resultmajor#warning </td> </tr> <tr> <td>Name</td> <td>ResultMinor</td> </tr> <tr> <td>Description</td> <td> <ul style="list-style-type: none"> ▪ /resultminor/al/common/noPermission ▪ /resultminor/al/common/internalError ▪ /resultminor/al/common/parameterError ▪ /resultminor/al/common/notSupported ▪ /resultminor/al/common/requestIDUnknown ▪ /resultminor/dp/unknownChannelHandle ▪ /resultminor/dp/communicationError ▪ /resultminor/dp/trustedChannelEstablishmentFailed ▪ /resultminor/dp/unknownProtocol ▪ /resultminor/dp/unknownCipherSuite </td> </tr> <tr> <td>Name</td> <td>ResultMessage</td> </tr> <tr> <td>Description</td> <td>Contains more detailed information about the error.</td> </tr> </table>	Name	ResultMajor	Description	<ul style="list-style-type: none"> ▪ /resultmajor#ok ▪ /resultmajor#pending ▪ /resultmajor#error ▪ /resultmajor#warning 	Name	ResultMinor	Description	<ul style="list-style-type: none"> ▪ /resultminor/al/common/noPermission ▪ /resultminor/al/common/internalError ▪ /resultminor/al/common/parameterError ▪ /resultminor/al/common/notSupported ▪ /resultminor/al/common/requestIDUnknown ▪ /resultminor/dp/unknownChannelHandle ▪ /resultminor/dp/communicationError ▪ /resultminor/dp/trustedChannelEstablishmentFailed ▪ /resultminor/dp/unknownProtocol ▪ /resultminor/dp/unknownCipherSuite 	Name	ResultMessage	Description	Contains more detailed information about the error.
Name	ResultMajor												
Description	<ul style="list-style-type: none"> ▪ /resultmajor#ok ▪ /resultmajor#pending ▪ /resultmajor#error ▪ /resultmajor#warning 												
Name	ResultMinor												
Description	<ul style="list-style-type: none"> ▪ /resultminor/al/common/noPermission ▪ /resultminor/al/common/internalError ▪ /resultminor/al/common/parameterError ▪ /resultminor/al/common/notSupported ▪ /resultminor/al/common/requestIDUnknown ▪ /resultminor/dp/unknownChannelHandle ▪ /resultminor/dp/communicationError ▪ /resultminor/dp/trustedChannelEstablishmentFailed ▪ /resultminor/dp/unknownProtocol ▪ /resultminor/dp/unknownCipherSuite 												
Name	ResultMessage												
Description	Contains more detailed information about the error.												

Document name:	Interface and Module Specification and Documentation	Page:	25 of 40
Reference:	D42.2	Dissemination:	PU
Version:	1.1	Status:	Final

Name	<code>dss:OptionalOutputs</code>
Description	MAY contain optional output elements.
Name	<code>SessionIdentifier</code>
Description	Contains the identifier of a session in which the authentication is performed. This element MUST be present if it was missing in the <code>Authenticate</code> request.
Name	<code>saml:Attribute</code>
Description	This element MAY appear an arbitrary number of times and is used to convey the requested identity attributes (cf. Section A.3).

Document name:	Interface and Module Specification and Documentation				Page:	26 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

8. Authentication Interface

This section describes the functionality, defines requirements for the operational environment, and specifies the functions of the Authentication Interface.

8.1 Functionality

The Authentication Interface provides an interface between the Universal Authentication Service and the FutureID Client, as shown in Figure 9. The FutureID Client contacts the Service Provider and gets forwarded to the Broker Service to perform user authentication. The Broker Service selects the appropriate authentication service, which in this case is the UAS. The FutureID Client will be redirected to the UAS and the authentication will be performed. Finally, the client will be sent back to the Service Provider and access to the service will be granted.

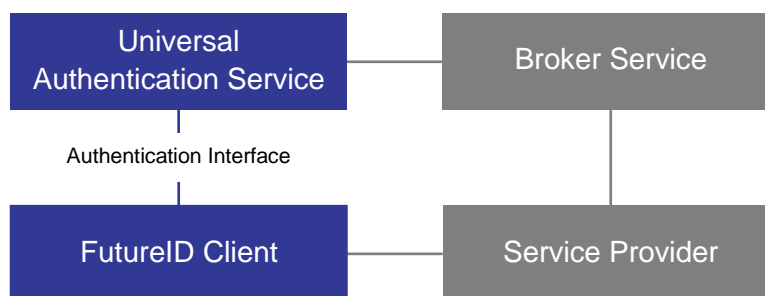


Figure 9: Authentication Interface

The Authentication interface specifies functions through which user authentication can be performed. In detail, the functions comprise the establishment of a channel, authenticating the user, and transmitting data between both parties. In particular, it SHOULD support at least the `StartPAOS`, `DIDAuthenticate`, and `Transmit` messages as specified in TR-03112 [5] and ISO/IEC 24727 [6].

8.2 Operational Environment

The interface is defined as a Web Service using WSDL [10] and requires a TLS [11] established channel between the participants.

8.3 Functions

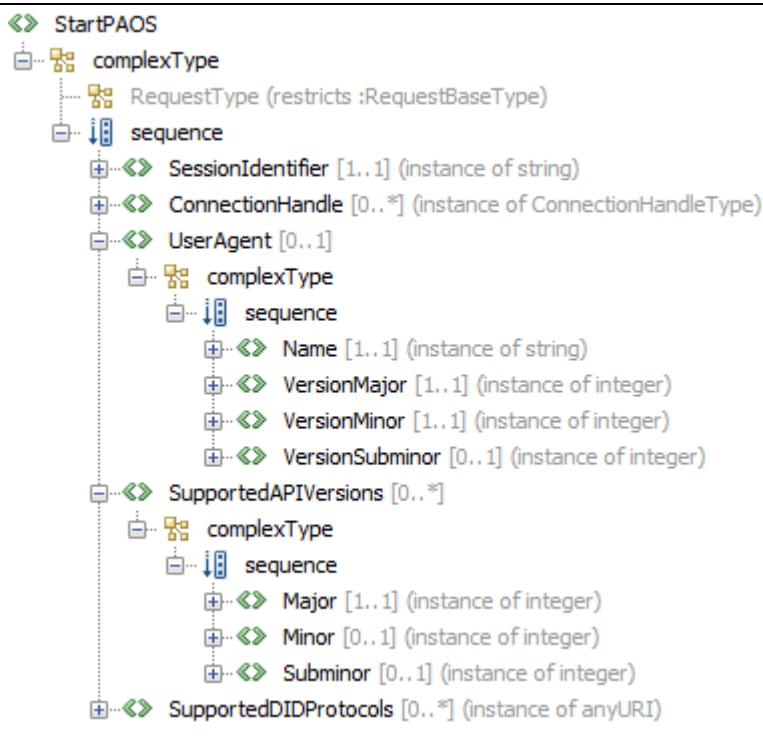
This section specifies the functions of the Authentication Interface. It comprises the `StartPAOS`, `DIDAuthenticate`, and `Transmit` functions as specified in BSI-TR-03112 [5] and ISO/IEC 24727 [6]. These functions should be sufficient for the majority of authentication protocol. However, the Authentication Interface can be easily extended by additional functions.

Document name:	Interface and Module Specification and Documentation				Page:	27 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

8.3.1 StartPAOS

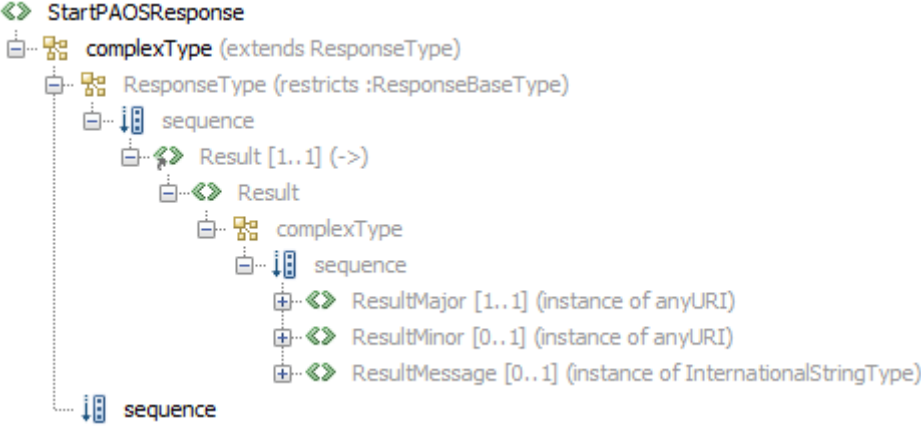
The function `StartPAOS` is used for the establishment of a PAOS connection according to [9] (cf. also TR-03112-7 [19], Section 2.6). PAOS stands for *Reverse HTTP binding for SOAP*. SOAP is a lightweight protocol for the exchange of information. SOAP enables exchange of SOAP messages using a variety of underlying protocols. The formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange is called a binding. PAOS specifies a header block and a binding of that header block to HTTP. The main difference with respect to the normal HTTP binding for SOAP is that in PAOS a SOAP request is bound to a HTTP response and vice versa.

REQUEST

Name	<code>StartPAOS</code>
Description	The function <code>StartPAOS</code> is used for the establishment of a PAOS connection.
Parameters	 <pre> <<>> StartPAOS complexType RequestType (restricts :RequestBaseType) sequence SessionIdentifier [1..1] (instance of string) ConnectionHandle [0..*] (instance of ConnectionHandleType) UserAgent [0..1] complexType sequence Name [1..1] (instance of string) VersionMajor [1..1] (instance of integer) VersionMinor [1..1] (instance of integer) VersionSubminor [0..1] (instance of integer) SupportedAPIVersions [0..*] complexType sequence Major [1..1] (instance of integer) Minor [0..1] (instance of integer) Subminor [0..1] (instance of integer) SupportedDIDProtocols [0..*] (instance of anyURI) </pre>
Name	<code>SessionIdentifier</code>
Description	Allows identifying the session between the service and the client.
Name	<code>ConnectionHandle</code>
Description	SHOULD occur for each connected card application and each empty card terminal slot available at the client.

Name	UserAgent	
Description	If present, SHALL contain information identifying the client. MUST NOT contain any information about the user, the computer or any software installed on the computer of the user apart from the client.	
Parameters	Name	Name
	Description	SHALL contain the name of the Client. SHALL NOT contain any version information.
	Name	VersionMajor
	Description	SHALL contain the major version of the client.
	Name	VersionMinor
	Description	SHALL contain the minor version of the client.
	Name	VersionSubminor
	Description	If present, SHALL contain the subminor version of the client.
Name	SupportedAPIVersions	
Description	Contains version numbers of supported versions of the API.	
Parameters	Name	Major
	Description	Version number of the API.
	Name	Minor
	Description	If not present, all minor versions corresponding to the given major version are supported.
	Name	Subminor
	Description	If not present, all subminor version corresponding to the given major/minor versions are supported.
Name	SupportetDIDProtocols	
Description	SHALL contain a list of supported DID protocols.	

RESPONSE

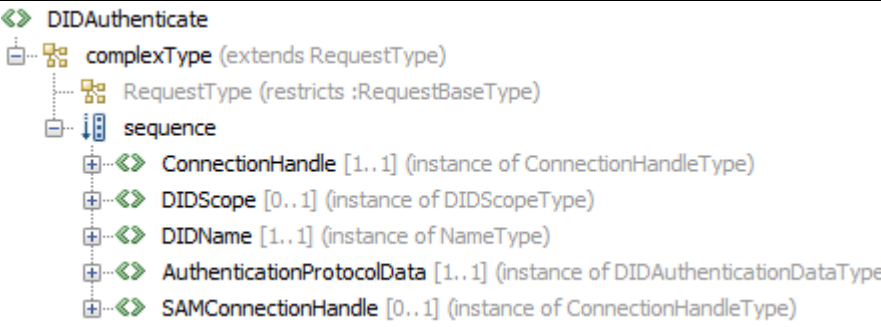
Name	StartPAOSReponse																				
Description	Return of the StartPAOS function.																				
Parameters	 <p>The diagram shows the structure of the StartPAOSResponse. It is a complexType that extends ResponseType. ResponseType restricts ResponseBaseType. The structure is as follows:</p> <ul style="list-style-type: none"> StartPAOSResponse (complexType) <ul style="list-style-type: none"> ResponseType (restricts :ResponseType) <ul style="list-style-type: none"> sequence <ul style="list-style-type: none"> Result [1..1] (->) <ul style="list-style-type: none"> Result (complexType) <ul style="list-style-type: none"> sequence <ul style="list-style-type: none"> ResultMajor [1..1] (instance of anyURI) ResultMinor [0..1] (instance of anyURI) ResultMessage [0..1] (instance of InternationalStringType) 																				
Name	dss:Result																				
Description	Contains the status information and the errors of an executed action.																				
Parameters	<table border="1"> <tr> <td>Name</td> <td colspan="2">ResultMajor</td> </tr> <tr> <td>Description</td> <td colspan="2"> <ul style="list-style-type: none"> /resultmajor#ok /resultmajor#error </td> </tr> <tr> <td>Name</td> <td colspan="2">ResultMinor</td> </tr> <tr> <td>Description</td> <td colspan="2"> <ul style="list-style-type: none"> /resultminor/al/common#noPermission /resultminor/al/common#internalError /resultminor/al/common#parameterError /resultminor/dp/nodeNotReachable /resultminor/dp/timeout </td> </tr> <tr> <td>Name</td> <td colspan="2">ResultMessage</td> </tr> <tr> <td>Description</td> <td colspan="2">Contains more detailed information about the error.</td> </tr> </table>			Name	ResultMajor		Description	<ul style="list-style-type: none"> /resultmajor#ok /resultmajor#error 		Name	ResultMinor		Description	<ul style="list-style-type: none"> /resultminor/al/common#noPermission /resultminor/al/common#internalError /resultminor/al/common#parameterError /resultminor/dp/nodeNotReachable /resultminor/dp/timeout 		Name	ResultMessage		Description	Contains more detailed information about the error.	
Name	ResultMajor																				
Description	<ul style="list-style-type: none"> /resultmajor#ok /resultmajor#error 																				
Name	ResultMinor																				
Description	<ul style="list-style-type: none"> /resultminor/al/common#noPermission /resultminor/al/common#internalError /resultminor/al/common#parameterError /resultminor/dp/nodeNotReachable /resultminor/dp/timeout 																				
Name	ResultMessage																				
Description	Contains more detailed information about the error.																				

8.3.2 DIDAuthenticate

The `DIDAuthenticate` function is defined in the standards BSI-TR-03112-4 [20] and ISO/IEC 24727-3 [20]. ISO/IEC 24727-3 defines an interface that contains functions to establish (cryptographically protected) connections to smart cards, to manage card applications, to read or write data, to perform cryptographic operations and to manage the respective key material (in the form of so-called "differential identities"). All functions which use or manage Differential Identities (DID) are parameterised by means of protocol-specific object identifiers so that the different protocols MAY be used with a standardised interface (refer to [19]).

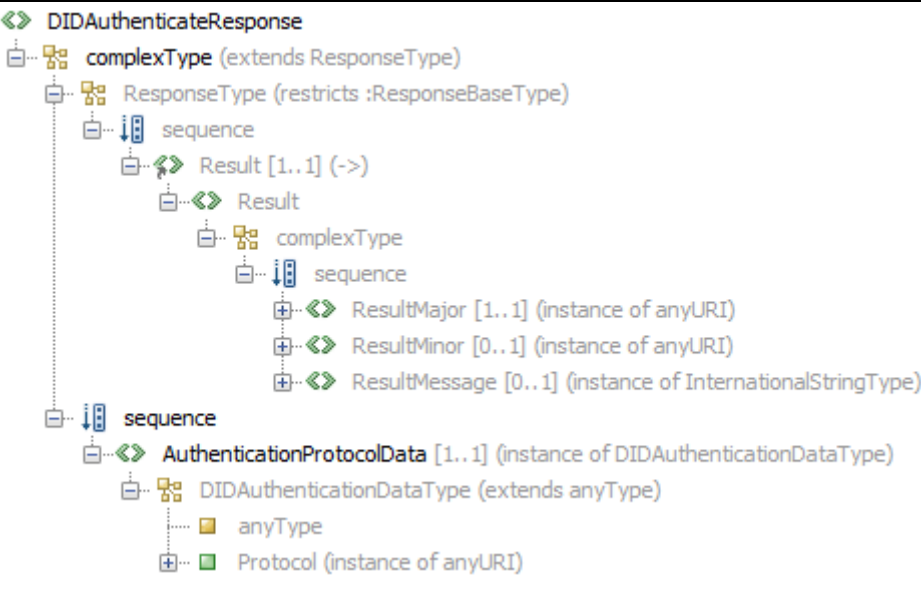
The `DIDAuthenticate` function belongs to the set of functions that use or manage Differential Identities. It can be used to execute an authentication protocol using a DID addressed by `DIDName`, which is created previously by using the `DIDCreate` function or similar means during the personalization phase. The `DIDAuthenticate` function contains an invocation parameter `AuthenticationProtocolData` of type `DIDAuthenticationDataType`, which depends on the concrete authentication protocol. Analogously, this function returns a parameter `AuthenticationProtocolData` of type `DIDAuthenticationDataType`, which also depends on the authentication protocol.

REQUEST

Name	<code>DIDAuthenticate</code>		
Description	The <code>DIDAuthenticate</code> function can be used to execute an authentication protocol using a DID addressed by <code>DIDName</code> .		
Parameters			
	Name	<code>ConnectionHandle</code>	
	Description	Contains a handle through which the connection to a card application is addressed.	
	Name	<code>DIDScope</code>	
	Description	Denotes the scope of the DID (local and global).	
	Name	<code>DIDName</code>	
	Description	Contains the name of the DID which is used for authentication. The	

	authentication protocol is determined by the AuthenticationProtocolData element below.
Name	AuthenticationProtocolData
Description	Contains the data necessary for the respective authentication protocol. The structure of the DIDAuthenticationDataType is specified on the basis of the protocol. The DIDAuthenticationDataType serves as a generic template for the definition of protocol-specific authentication protocol data elements.
Name	SAMConnectionHandle
Description	MAY address a connection to a card application, which serves as Security Access Module (SAM). The detailed role of the SAM within the authentication protocol MUST be defined within the specification of the authentication protocol.

RESPONSE

Name	DIDAuthenticateReponse					
Description	Return of the DIDAuthenticate function.					
Parameters						
Name	dss:Result					
Description	Contains the status information and the errors of an executed action.					
Parameters	<table border="1"> <tr> <td>Name</td> <td>ResultMajor</td> </tr> <tr> <td>Description</td> <td> <ul style="list-style-type: none"> ▪ /resultmajor#ok ▪ /resultmajor#error ▪ /resultmajor#nextRequest </td> </tr> </table>		Name	ResultMajor	Description	<ul style="list-style-type: none"> ▪ /resultmajor#ok ▪ /resultmajor#error ▪ /resultmajor#nextRequest
Name	ResultMajor					
Description	<ul style="list-style-type: none"> ▪ /resultmajor#ok ▪ /resultmajor#error ▪ /resultmajor#nextRequest 					


	Name	ResultMinor
	Description	<ul style="list-style-type: none"> ▪ /resultminor/al/common#incorrectParameter ▪ /resultminor/sal#namedEntityNotFound ▪ /resultminor/sal#protocolNotRecognized ▪ /resultminor/sal#inappropriateProtocolForAction ▪ /resultminor/sal#notInitialized ▪ /resultminor/sal#securityConditionNotSatisfied ▪ /resultminor/sal#insufficientResources ▪ /resultminor/dp#communicationFailure
	Name	ResultMessage
	Description	Contains more detailed information about the error.
Name	AuthenticationProtocolData	
Description	Contains the necessary data for the respective authentication protocol.	

Document name:	Interface and Module Specification and Documentation				Page:	33 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

8.3.3 Transmit

The `Transmit` function is specified by the IFD-Interface for the terminal layer of the smart card [21]. The IFD-Interface generalises the specific card terminal types and various interfaces and communicates with the smart card. The IFD-Interface provides card terminal functions, card functions and user interaction functions. The card functions group allows to connect and to disconnect to a card, to begin and end a transaction with a card and to send APDUs to a card. The `Transmit` function is the one used to send APDUs to a card.

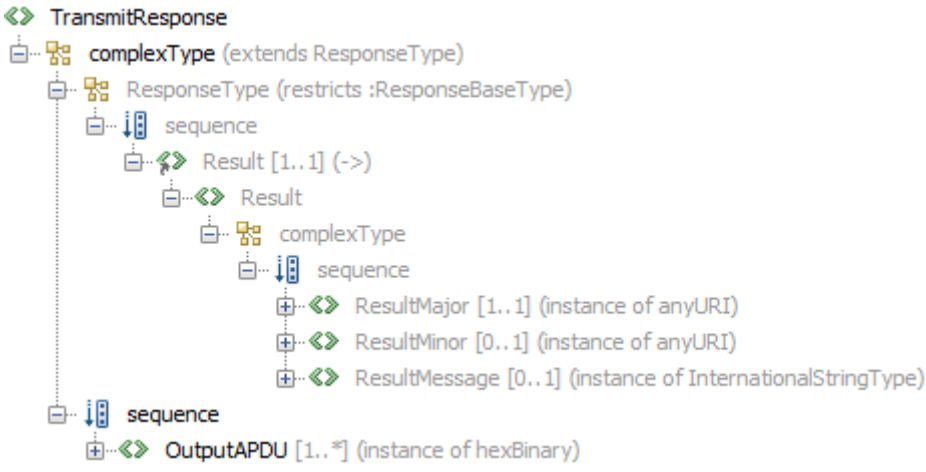
REQUEST

Name	Transmit						
Description	The <code>Transmit</code> function sends one or more APDU(s) to a connected smart card. In order to support the batch processing a set of <code>AcceptableStatusCode</code> elements (0x9000 etc.) MAY be attached to each <code>InputAPDU</code> . If the smart card returns some not expected status code it is – even in case of secure messaging – clear that there is a serious error and it does not make sense to feed the remaining <code>InputAPDU</code> elements in the batch to the connected smart card.						
Parameters	 <pre> classDiagram class Transmit class complexType["complexType (extends RequestType)"] class RequestType["RequestType (restricts :RequestBaseType)"] class sequence1["sequence"] class SlotHandle["SlotHandle [1..1] (instance of SlotHandleType)"] class SlotHandleType["SlotHandleType (restricts hexBinary)"] class InputAPDUInfo["InputAPDUInfo [1..*] (instance of InputAPDUInfoType)"] class InputAPDUInfoType["InputAPDUInfoType"] class sequence2["sequence"] class InputAPDU["InputAPDU [1..1] (instance of hexBinary)"] class AcceptableStatusCode["AcceptableStatusCode [0..*] (instance of hexBinary)"] Transmit --> complexType complexType --> RequestType RequestType --> sequence1 sequence1 --> SlotHandle SlotHandle --> SlotHandleType sequence1 --> InputAPDUInfo InputAPDUInfo --> InputAPDUInfoType InputAPDUInfoType --> sequence2 sequence2 --> InputAPDU sequence2 --> AcceptableStatusCode </pre>						
Name	SlotHandle						
Description	With the <code>SlotHandle</code> the connection established with <code>Connect</code> to the smart card is addressed.						
Name	InputAPDUInfo						
Description	MAY be present multiple times and contains the command APDU, which is sent to the eCard and optionally accept-able status codes. It is of type <code>InputAPDUInfoType</code> , which is explained below.						
Parameters	<table border="1"> <tr> <td>Name</td> <td>InputAPDU</td> </tr> <tr> <td>Description</td> <td>Contains the APDU which is to be sent to the smart card.</td> </tr> </table>			Name	InputAPDU	Description	Contains the APDU which is to be sent to the smart card.
Name	InputAPDU						
Description	Contains the APDU which is to be sent to the smart card.						

	Name	AcceptableStatusCode
	Description	MAY be present multiple times per InputAPDU-element in order to specify the set of expected status codes. If the status code which is returned from the smart card is not among the expected values the batch processing SHOULD be stopped as this indicates that there is a serious error condition. If the <code>AcceptableStatusCode</code> -element is omitted, any returned status code is assumed to be acceptable.

Document name:	Interface and Module Specification and Documentation				Page:	35 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

RESPONSE

Name	TransmitReponse														
Description	Return of the Transmit function.														
Parameters	 <pre> <> TransmitResponse complexType (extends ResponseType) ResponseType (restricts :ResponseType) sequence Result [1..1] (->) Result complexType sequence ResultMajor [1..1] (instance of anyURI) ResultMinor [0..1] (instance of anyURI) ResultMessage [0..1] (instance of InternationalStringType) sequence OutputAPDU [1..*] (instance of hexBinary) </pre>														
Name	dss:Result														
Description	Contains the status information and the errors of an executed action.														
Parameters	<table border="1"> <tr> <td>Name</td> <td>ResultMajor</td> </tr> <tr> <td>Description</td> <td> <ul style="list-style-type: none"> /resultmajor#ok /resultmajor#error </td> </tr> <tr> <td>Name</td> <td>ResultMinor</td> </tr> <tr> <td>Description</td> <td> <ul style="list-style-type: none"> /resultminor/ifdl/common#timeoutError /resultminor/ifdl/common#invalidSlotHandle /resultminor/al/common#unknownError </td> </tr> <tr> <td>Name</td> <td>ResultMessage</td> </tr> <tr> <td>Description</td> <td>MAY contain more detailed information about the error.</td> </tr> </table>			Name	ResultMajor	Description	<ul style="list-style-type: none"> /resultmajor#ok /resultmajor#error 	Name	ResultMinor	Description	<ul style="list-style-type: none"> /resultminor/ifdl/common#timeoutError /resultminor/ifdl/common#invalidSlotHandle /resultminor/al/common#unknownError 	Name	ResultMessage	Description	MAY contain more detailed information about the error.
Name	ResultMajor														
Description	<ul style="list-style-type: none"> /resultmajor#ok /resultmajor#error 														
Name	ResultMinor														
Description	<ul style="list-style-type: none"> /resultminor/ifdl/common#timeoutError /resultminor/ifdl/common#invalidSlotHandle /resultminor/al/common#unknownError 														
Name	ResultMessage														
Description	MAY contain more detailed information about the error.														
Name	OutputAPDU														
Description	MAY be present multiple times and contains the APDU returned by the card. A successful call of the Transmit function MUST contain exactly as many InputAPDU as OutputAPDU elements.														

Document name:	Interface and Module Specification and Documentation				Page:	36 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

9. Conclusion

The Broker Service provides a generic interface for service providers to authenticate their users using arbitrary eID credentials. In case of the Claims Transformer Mode the Broker Service includes the Universal Authentication Services (UAS) to perform the user authentication. The UAS acts as an authentication service and supports arbitrary authentication mechanism through the Authentication Protocol Specification (APS) language. The Basic Services provides a common set of functions to support a wide range of authentication protocols.

The Job Execution Environment is the core component of the UAS and is capable of running APS-specified authentication protocols. The UAS encompasses two interfaces: The Universal Service Interface provides a generic interface for the Broker Service to allow services providers to use the UAS as an authentication service. The Authentication Interface provides an interface for the FutureID client to perform the user authentication.

The UAS focuses on providing a convenient and sophisticated Identity Provider with a novel approach of supporting arbitrary authentication protocols. In the long run the UAS should be extended to support signature, credential, and verification services.

Document name:	Interface and Module Specification and Documentation				Page:	37 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

10. Bibliography

- [1] ISO/IEC, *Identification cards - Integrated circuit cards*, International Standard, ISO/IEC 7816, Part 1 - 13,15, 2004 - 2011.
- [2] FutureID, *WP42 - Universal Authentication Service, D42.1 - Requirements report*, 2013.
- [3] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, RFC 2119, <https://tools.ietf.org/html/rfc2119>, 1997.
- [4] FutureID, *WP21 - Vision, Approach and Inventory, D21.1 - FutureID Terminology*, 2013.
- [5] Bundesamt für Sicherheit in der Informationstechnik (BSI), *eCard-API-Framework, Technical Guideline TR-03112, Part 1 - 7, Version 1.1.2*, https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html.
- [6] ISO/IEC, *Identification cards - Integrated circuit card programming interfaces*, ISO/IEC 24727, Part 1 - 5.
- [7] European Committee for Standardization (CEN), *Identification card systems - European Citizen Card, CEN/TS 15480, Part 1 - 4*, 2008.
- [8] B. Don, E. David, K. Gopal, L. Andrew and M. Noah, *Simple Object Access Protocol (SOAP) 1.1*, 2000.
- [9] Liberty Alliance Project, *Liberty Reverse HTTP Bindin for SOAP Specification, Version 2.0*, <http://www.projectliberty.org/liberty/content/download/909/6303/file/liberty-paos-v2.0.pdf>.
- [10] World Wide Web Consortium (W3C), *Web Services Description Language (WSDL) 1.1*, <http://www.w3.org/TR/wsdl.html>, 2001.
- [11] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, <https://tools.ietf.org/html/rfc5246>, 2008.
- [12] FutureID, *WP42 - Universal Authentication Service, D42.3 - Specification of APS-language*, 2013.
- [13] FutureID, *WP42 - Universal Authentication Service, D42.6 - Specification of Execution Environment*, 2013.

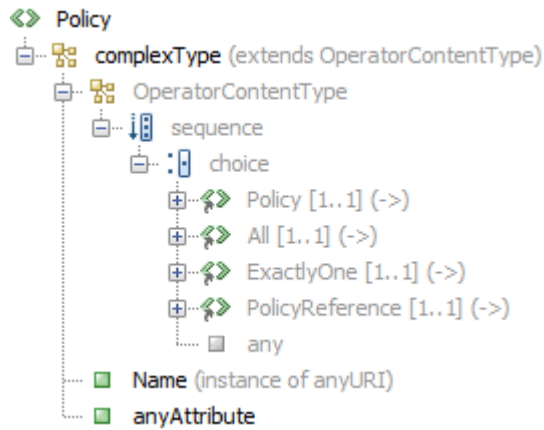
Document name:	Interface and Module Specification and Documentation				Page:	38 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

- [14] FutureID, *WP42 - Universal Authentication Service, D42.4 - Specification of Basic Services*, 2013.
- [15] OASIS, *Web Services Security (WSS)*, 2006: Version 1.1, <http://docs.oasis-open.org/wss/v1.1/>.
- [16] European Committee for Standardization (CEN), *Identification card systems - European Citizen Card - Part 3: European Citizen Card Interoperability using an application interface*, CEN 15480-3, 2010.
- [17] FutureID, *WP41 - Identity Broker, D41.2 - Interface and module specification and documentation*, 2013.
- [18] OASIS, *Security Assertion Markup Language (SAML) V2.0*, <http://docs.oasis-open.org/security/saml/v2.0/>, 2009.
- [19] Bundesamt für Sicherheit in der Informationstechnik (BSI), *eCard-API-Framework - Protocols*, Technical Guideline TR-03112-7, Version 1.1.2, 2012.
- [20] Bundesamt für Sicherheit in der Informationstechnik (BSI), *eCard-API-Framework - ISO 24727-3-Interface*, Technical Guideline TR-03112-4, Version 1.1.2, 2012.
- [21] Bundesamt für Sicherheit in der Informationstechnik (BSI), *eCard-API-Framework - IFD-Interface*, Technical Guideline TR-03112-6, Version 1.1.2, 2012.

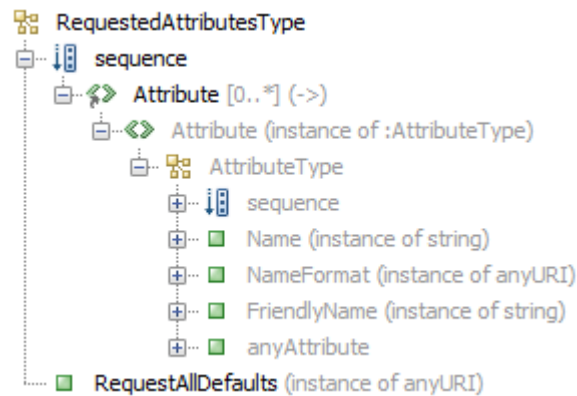
Document name:	Interface and Module Specification and Documentation				Page:	39 of 40	
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status:	Final

A. Appendix

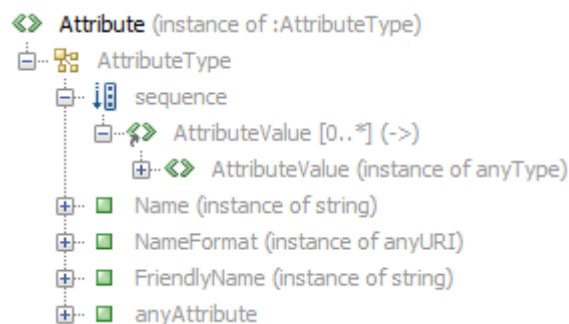
A.1 Policy



A.2 RequestedAttributeType



A.3 Attribute



Document name:	Interface and Module Specification and Documentation				Page:	40 of 40
Reference:	D42.2	Dissemination:	PU	Version:	1.1	Status: Final