

UNIBE.2 - AAA Printing A SWITCH AAA project

Pascal Mainini
University of Bern, IT Services, Security Group
`pascal.mainini@id.unibe.ch`

September 2009

With SWITCHaai a swiss-wide infrastructure for authentication and authorisation across institutions has already been established. So far, this infrastructure lacks accounting functionality.

In the whitepaper *Enhancing SWITCHaai with Micropayment Functionality for Swiss Universities*[SCR06] a possible way of implementing such accounting-functionality has been described based on the example of a multi-institutional printing-service.

In a follow-up project, a prototype has been built to verify the concepts made by the above whitepaper and to detect possible issues which could come up in a practical implementation.

This document describes the prototype built as well as outcomes and issues found during development. It also has a look at possible directions for further projects.

Contents

I	Overview of the Project	5
1	Timeline and Tasks	6
1.1	Adapting the Printing Solution to SWITCHaai	6
1.2	Developing the Prototype	6
1.3	Integrating the Prototype with the Printing Solution	7
2	People and Organisations Involved	8
II	Development of the Prototype	9
3	Technologies used	10
3.1	Python	10
3.2	Twisted	10
3.3	XML	11
3.4	XML-RPC	11
3.5	REST	11
3.6	Logging	11
4	Components	12
4.1	Overview	12
4.2	AAA Service	13
4.2.1	Classes of the AAA Service	14
4.3	AAA Provider	23
4.3.1	Classes of the AAA Provider	24
4.4	Common Modules	28
4.4.1	Module SSL	29
4.4.2	Module Configuration	30
5	Interface Specifications	31
5.1	Application to AAA-Service Interface	31
5.1.1	Method <code>push_attribute</code>	31
5.2	AAA-Service to AAA-Provider Interface	33
5.2.1	Method <code>push_session</code>	33
5.2.2	Method <code>push_assertion</code>	33
6	Open Points	35

III Setup of the Prototype at the University of Bern	36
7 Integration with the Uniprint-Solution	37
8 Internal Tests with an UniBE-Account	39
9 External Tests with a BFH-Account	40
IV Issues Uncovered	41
10 Issues with Third Parties	42
11 Issues with Shibboleth	43
12 Other Issues	44
12.1 Location of the Provider	44
12.2 Laws Regarding Banking	44
12.3 Unclear Attributes	45
V Possible Directions for Future Projects	46
13 Extend Prototype to include PHBern	48
14 Standardisation of Attributes and Brokerage	49
15 Mobile-/Micropayment?	50
16 Swiss-Wide SOA Architecture?	51
VI Conclusion	52
17 Conclusion	53
VII Appendix	54
Contents of Deliverable	55
Installation Instructions	56
Overview of Configuration Files	57
Bibliography	58
List of Figures	59
List of Tables	60

Part I

Overview of the Project

1 Timeline and Tasks

The prototype was developed during a two-year project phase at the *University of Bern*¹. After some initial work in the first quarter of the year 2008, it officially started on first of April 2008 and ended on 30th of September 2009. Some afterwork was done after this time.

In the project, three main tasks have been achieved:

- Adapt third-party printing solution to *SWITCHaai*²
- Develop the prototype implementation of the accounting components
- Integrate the prototype with the printing solution and test it

1.1 Adapting the Printing Solution to SWITCHaai

At first, the printing solution, which had been independently chosen for the *Uniprint*-project at the *University of Bern*, needed to be adapted to the *SWITCHaai* environment.

In a first step, the service itself needed to be *shibbolized*. Secondly, the application needed to be adapted to accept the *SwissEduPersonUniqueID*³ as a user-id.

Unfortunately, this caused a lot more issues and effort than initially expected, see also chapter "Issues with Third Parties"

1.2 Developing the Prototype

In parallel, the development of the prototype took place. The whole development is described in the part "Development of the Prototype".

¹<http://www.unibe.ch>

²<http://www.switch.ch/aai>

³<http://www.switch.ch/aai/support/documents/attributes.html>

1.3 Integrating the Prototype with the Printing Solution

In a last step, the prototype and the printing solution were integrated together, see chapter "Integration with the Uniprint-Solution".

2 People and Organisations Involved

At the *University of Bern* multiple persons were involved in the project.

*Pascal Mainini*¹ as project leader and lead developer of the prototype.

On the administrative side, *Christian Heim*² and *Christian Bieri*³ where the responsible administrative contacts for both, the *UNIBE.1* as well as the *UNIBE.2* project (see *AAA/SWITCH projects*⁴).

From the side of the group "Verwaltungsinformatik" within the IT-department⁵ of the *University of Bern*, *Urs von Lerber*⁶ as well as *Silvia Spring*⁷ were involved as primary contacts to the vendor of the printing solution.

The printing solution itself came from *Ricoh/Genius Bytes*⁸ (only refered to as *Genius Bytes* within this document).

Last but not least, great support was provided at any time from *SWITCH*⁹, especially from their *AAA-team*¹⁰.

¹pascal.mainini@id.unibe.ch

²christian.heim@id.unibe.ch

³christian.bieri@id.unibe.ch

⁴<http://www.switch.ch/aaa/projects/>

⁵<http://www.id.unibe.ch>

⁶urs.vonlerber@id.unibe.ch

⁷silvia.spring@id.unibe.ch

⁸<http://www.geniusbytes.com>

⁹<http://www.switch.ch>

¹⁰<http://www.switch.ch/aaa>

Part II

Development of the Prototype

3 Technologies used

This chapter gives a short overview of the technologies used for building the prototype and why they were chosen.

3.1 Python

As programming language, *Python*¹ has been chosen due to different reasons:

- Simplicity of development and debugging
- "Batteries included" - lots of APIs part of the standard distribution
- Crossplatform compatibility
- Clean source formatting rules - ensures further reusability
- Familiarity of the developers with the language

As Python is a scripting language, development is usually a bit faster than in other, non interpreted languages. Especially the interactiveness given with its shell greatly helps developing and debugging an application.

No platform-dependent code was used for developing the prototype - it should run on any platform, although this was not tested.

Note: Python version 2.5 was used for developing and running the prototype.

3.2 Twisted

Various networking functionalities are required by the prototype. This is achieved using the *Twisted-framework*², a modern framework for all kinds of networking needs. Often referred to as "the framework of your internet" it provides a modern, asynchronous approach in developing clients and servers for network protocols.

¹<http://www.python.org>

²<http://twistedmatrix.com>

3.3 XML

XML is used at different places in the prototype. Configuration for all parts of the prototype is done in *XML* - also, one of the available service-interfaces as well as the interface between the *service* and the *provider* uses *XML-RPC*.

3.4 XML-RPC

*XML-RPC*³ is used for the interface between the *AAA service* and the *AAA provider*. An AAA-enabled webservice can choose to use the *XML-RPC* interface to communicate with the *AAA service*. XML-RPC has been chosen because of its wide acceptance in the industry, resulting in a large number of frameworks and APIs supporting it.

3.5 REST

The other interface available for AAA-enabled webservices is a *RESTful HTTP-interface*⁴ It provides the same functionality as the *XML-RPC*-one but in a more easily accessible way. It is ideally suited for testing purposes as well as in productive systems. The RESTful interface has been implemented to provide third-party software developers with an easy-to-use interface for developing and testing their applications.

3.6 Logging

The prototype uses *Python*'s logging package and thus offers various forms of logging as well as filtering the log-output by priorities.

³<http://en.wikipedia.org/wiki/XML-RPC>

⁴http://en.wikipedia.org/wiki/Representational_State_Transfer

4 Components

4.1 Overview

The prototype consists of two components and some helper classes used by both components.

The two components consist of the *AAA Provider* and the *AAA Service* which integrate into the existing *SWITCHaai-infrastructure*¹, as can be seen in figure 4.1:

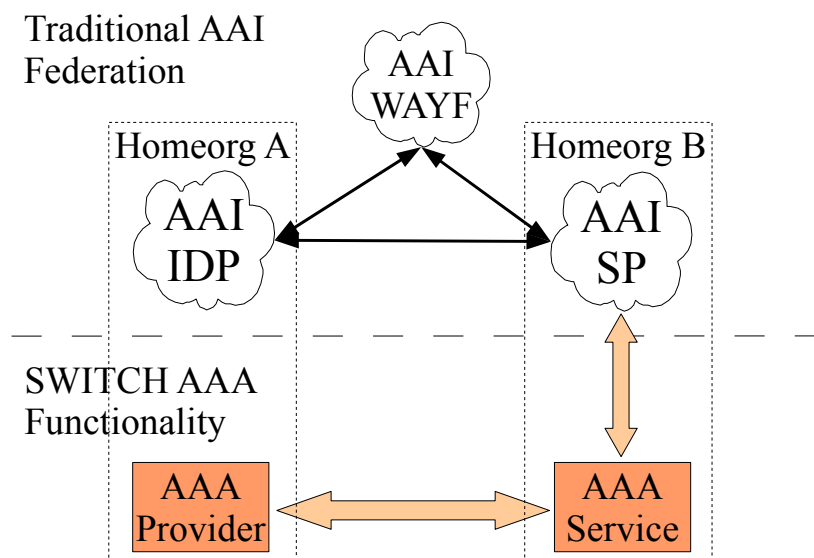


Figure 4.1: Component-Overview AAA Prototype

¹<http://www.switch.ch/aai>

4.2 AAA Service

The *AAA Service* provides the interface to an AAA-enabled application. Typically it is run on the same machine as the *Shibboleth serviceprovider*² and thus the AAA-enabled application itself.

There are two interfaces provided for the application to exchange accounting-data, one based on *XML-RPC* and the other based on *REST*. They are described in Application to AAA-Service Interface-section.

The *AAA Service* collects accounting-data from the application, which is received in one or multiple requests, and then sends them all together to the *AAA Provider* using the AAA-Service to AAA-Provider Interface.

Besides those functions, it also monitors *Shibboleth's* logfile for *SAML-assertions*³. These are also transmitted to the *AAA Provider* as an additional level of proof for the accounted data. This mechanism is described in [SCR06].

Figure 4.2 gives a detailed view of the parts of the *AAA Service*-components and how they interact with an application and *Shibboleth*.

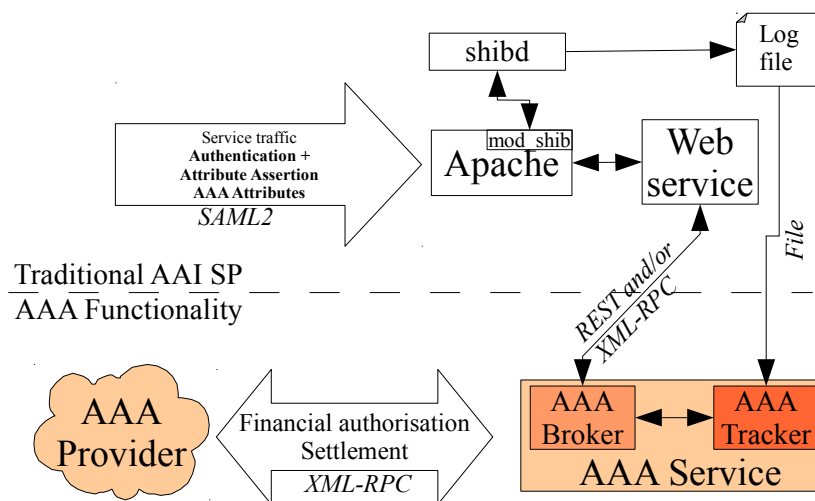


Figure 4.2: Overview of Components and Interfaces in an AAA-Enabled Webservice

²<http://shibboleth.internet2.edu>

³<http://en.wikipedia.org/wiki/SAML>

4.2.1 Classes of the AAA Service

Module AccountingService

The module AccountingService contains no classes.

It just contains the *main-method* which initialises and starts the WebServer as well as the AccountingTracker.

Module WebServer

The `WebServer`-module provides the interfaces to the AAA-enabled service, both the *XML-RPC* and the *RESTful* one.

The core functionality is provided by a `twisted.web.server`⁴ to which multiple resources are attached. This is done in the `WebServer` class.

The `RootResource` class displays an informative text to inform a potential human client what kind of application has been reached and which interfaces exist for automatic transmission of accounting attributes.

Two other resources are attached as children of the `RootResource`: the `SimpleRPCResource` as well as the `SoapResource`. These are responsible for providing the interfaces to the service. The actual implementation (business methods) of the interface is done in the module `AccountingBroker`.

The `SimpleRPCResource` provides the RESTful interface. If it doesn't get a proper method, it will also return a descriptive message to the client. See chapter "service interface specification" for more information.

The other resource, the `SoapResource` provides a standard *XML-RPC* interface to the same methods as the RESTful one. See as well the service interface specification for more information.

The classes are illustrated in the following diagramm.

⁴<http://twistedmatrix.com/documents/current/api/twisted.web.server.html>

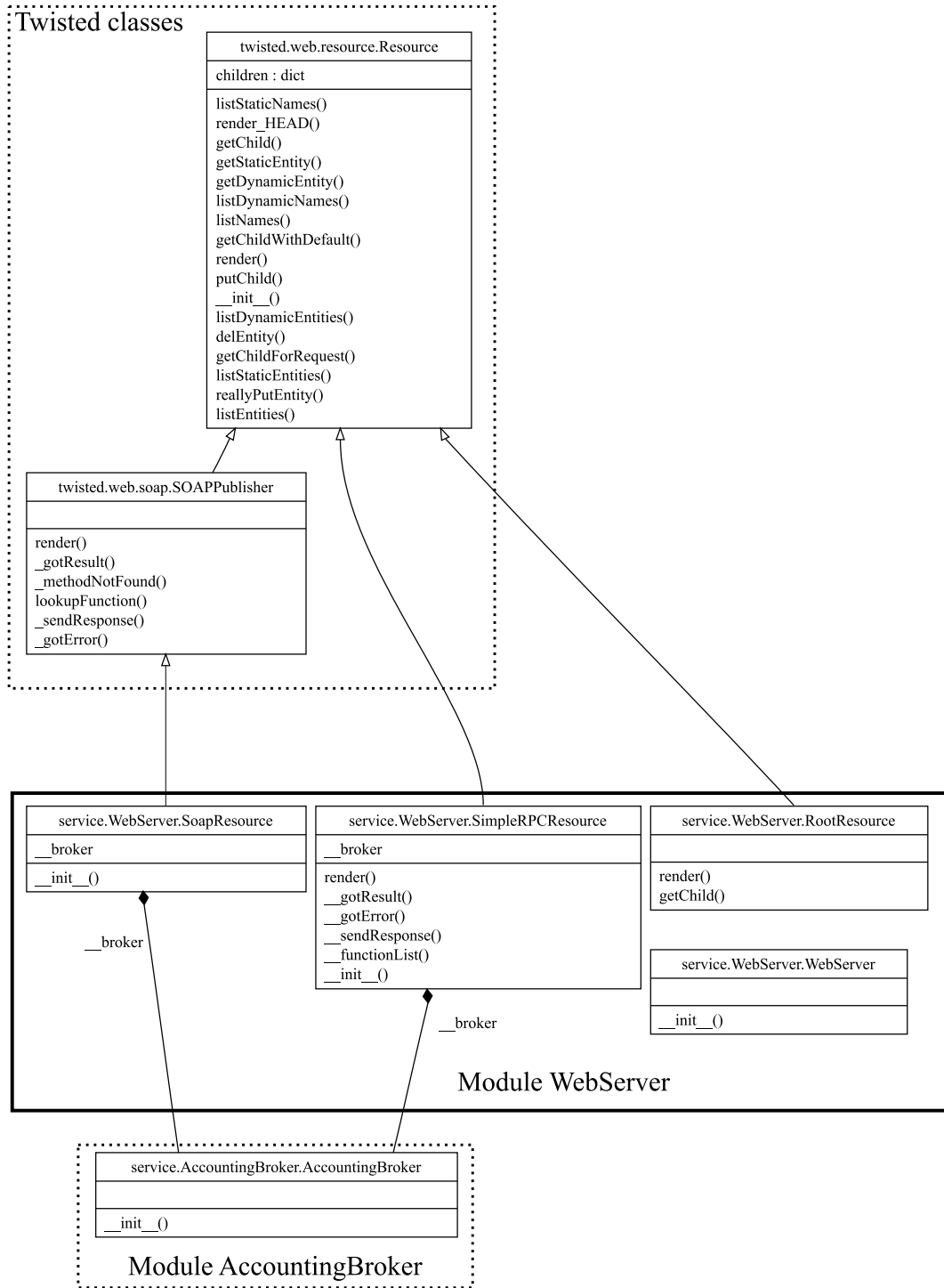


Figure 4.3: Class diagram for the module WebServer

Module AccountingBroker

The `AccountingBroker`-module consists of two main classes as well as some exceptions.

The `AccountingBroker` class provides the actual implementation of the business methods. As it is used by both interfaces, it has been implemented as a *singleton*.

Currently only the method `push_attribute` is implemented for the prototype. In a real world usage scenario, additional methods (for instance for inquiring the available attributes) would be added.

The second important class in the module is the `SessionStore`. As defined in the service interface specification, multiple attributes can be passed to the *AAA service* by multiple method calls in a session. The `SessionStore` is used as a temporary storage to keep together all attributes belonging to a single session, before they are sent in a single method call to the provider.

Additionally, three exceptions are defined in the module, which are used internally.

The `DuplicateAttributeException` is thrown, when an attribute has been received multiple times.

The `InvalidCountException` is thrown if the number of an individual attribute post is bigger than the total count for a session or lower than zero.

Finally, the `SessionOutOfOrderException` gets thrown, if the attributes don't arrive in the proper order or the session has already been closed.

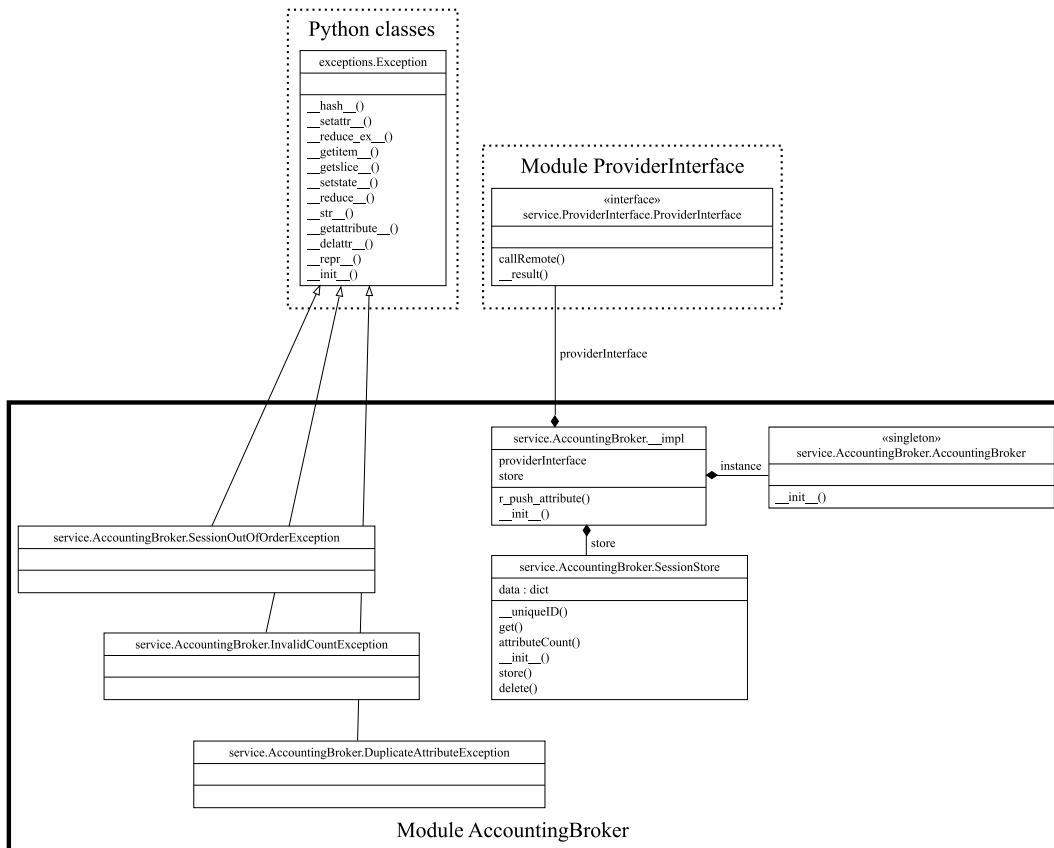


Figure 4.4: Class diagram for the module AccountingBroker

Module AccountingTracker

The module `AccountingTracker` contains no classes.

It just contains the `track()`-method, which gets scheduled by the `AccountingService`-module to track assertions in *Shibboleth*'s logfile.

Module ProviderInterface

The `ProviderInterface`-module, as its name implies, interfaces with the *AAA Provider*. It contains the `ProviderInterface` class and the `ProviderNotFoundException`.

The `ProviderInterface` calls remote methods on the *AAA provider*. It automatically determines which provider to send the requests to by using the provider metadata.

In case no provider can be found, the `ProviderNotFoundException` is thrown.

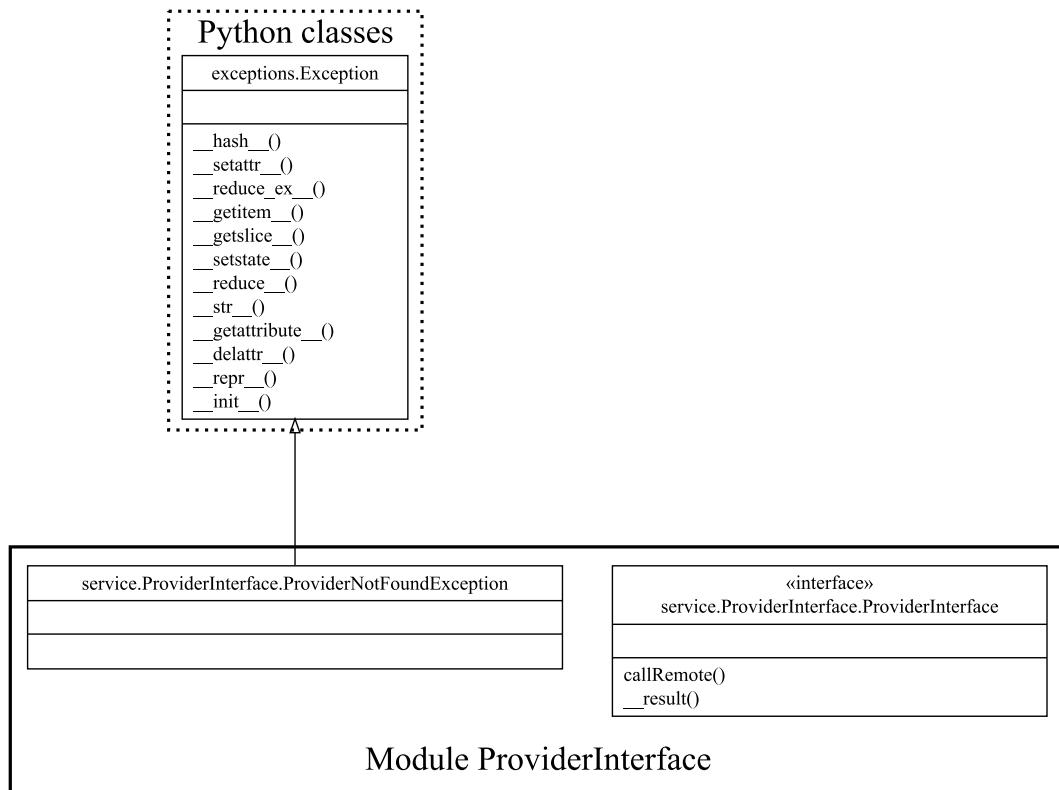


Figure 4.5: Class diagram for the module `ProviderInterface`

Module ServiceConfiguration

The `ServiceConfiguration` class found in this module provides methods for accessing the configuration of an *AAA service* in a convenient way. It uses a lot of the abstraction provided by the `Configuration` class.

It has also been implemented as a *singleton* in order to avoid loading the configuration multiple times.

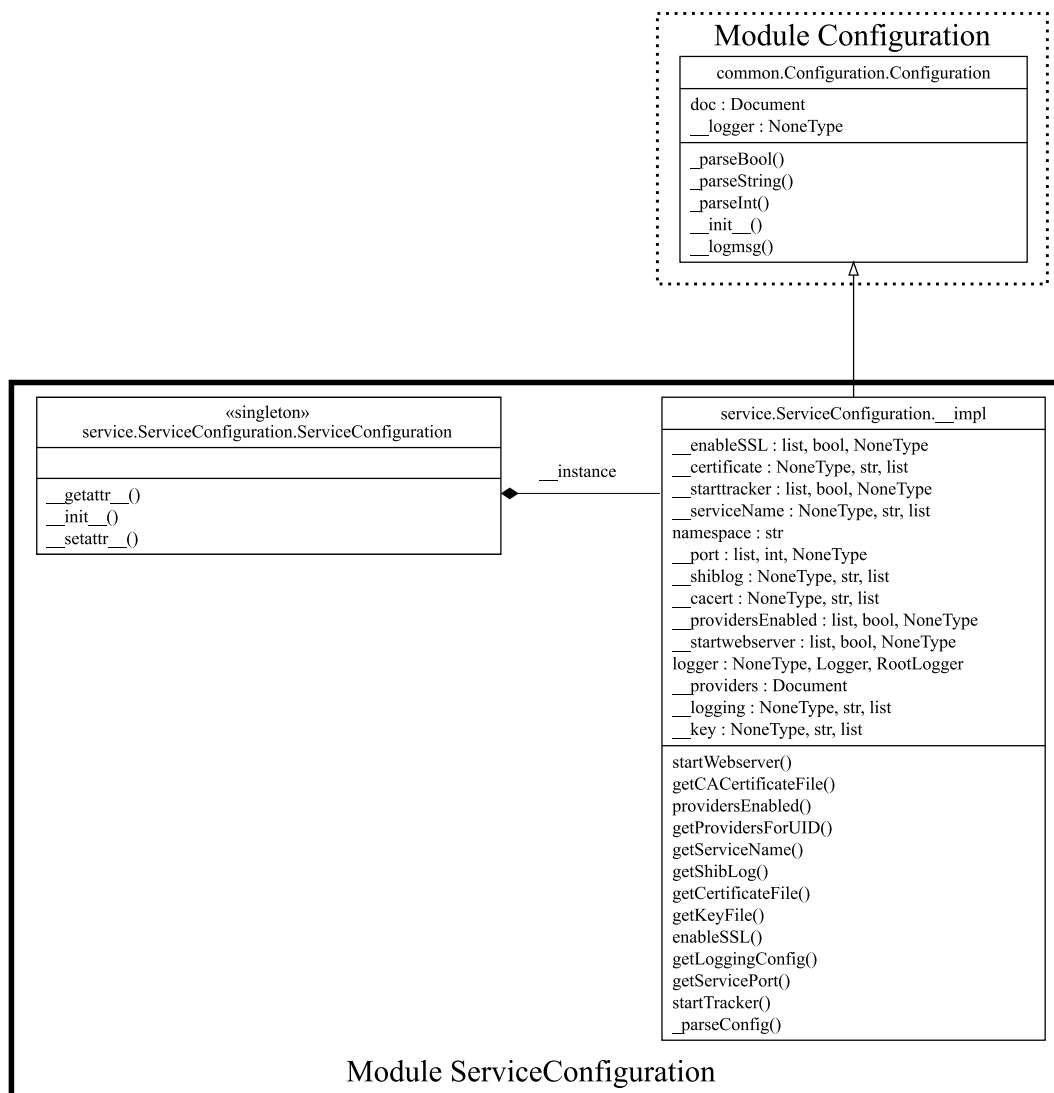


Figure 4.6: Class diagram for the module ServiceConfiguration

Module TestApplicationInterface

The module `TestApplicationInterface` contains no classes, but code for performing some tests on the service interface.

4.3 AAA Provider

The *AAA provider* is the component installed at the institution from which the foreign user comes / to which accounting data is sent from the *AAA service*.

For the prototype implementation, it only provides a very limited subset of features - currently dynamically interchangeable serializers for writing the accounting information out to files or other accounting services, as well as a serializer for writing out the *SAML-assertions* gathered by the tracker.

The only currently implemented serializer is a CSV-serializer, writing values to comma-separated fields in a file. Which fields are written and in which order is fully configurable, see description of the CSVSerializer configuration.

4.3.1 Classes of the AAA Provider

Modules AccountingProvider and ServiceInterface

The `AccountingProvider` and the `ServiceInterface` modules are tightly related to each other.

In the former, the class `SoapResource` decorates all the business methods of the class `ServiceInterface` (in the module of the same name) and makes them available via XML-RPC.

The `AccountingProvider`-module also contains the *main-method* of the *AAA provider*, which sets up a `twisted.web.server`⁵ to provide the methods given by `SoapResource` to the *AAA service*.

The class `ServiceInterface` from the so-called module provides two business methods, one for receiving complete attribute-sessions from the *AAA service*, the other to receive the assertions extracted by the tracker.

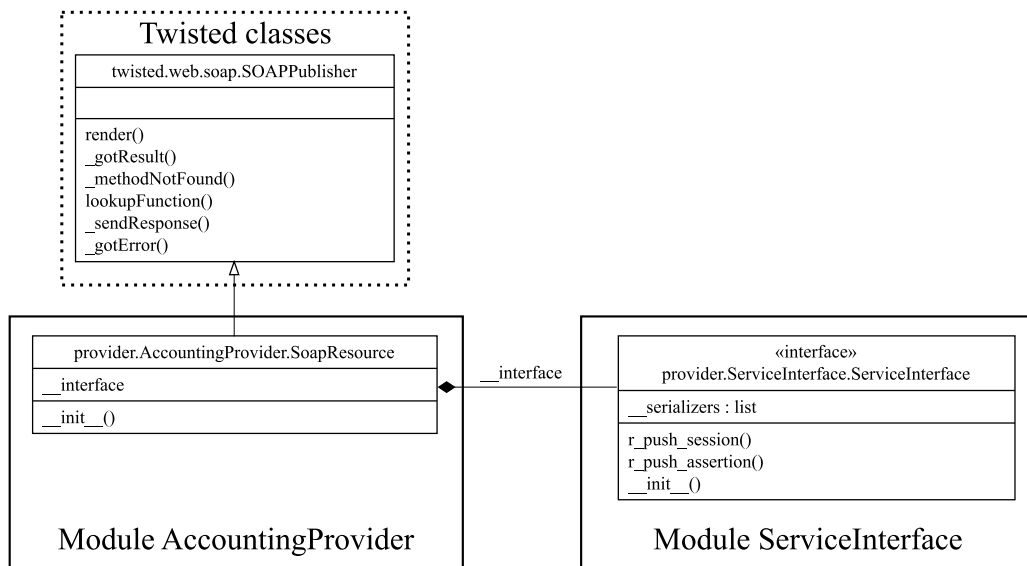


Figure 4.7: Class diagram for the modules `AccountingProvider` and `ServiceInterface`

⁵<http://twistedmatrix.com/documents/current/api/twisted.web.server.html>

Module CSVSerializer

The `CSVSerializer`-module is used for serializing the accounting information to a file with comma separated values.

Actually, the serializer is fully configurable: Not only commas can be used as separator and the ordering of the fields is freely adjustable. More about this feature can be found at the description of the `CSVSerializer` configuration.

Two classes are used to implement this functionality: the `CSVSerializer` is the actual serializer, `CSVSerializerConfiguration` loads the configuration for the serializer.

Multiple serializers can be used by an *AAA provider*, for the prototype however only the `CSVSerializer` has been implemented.

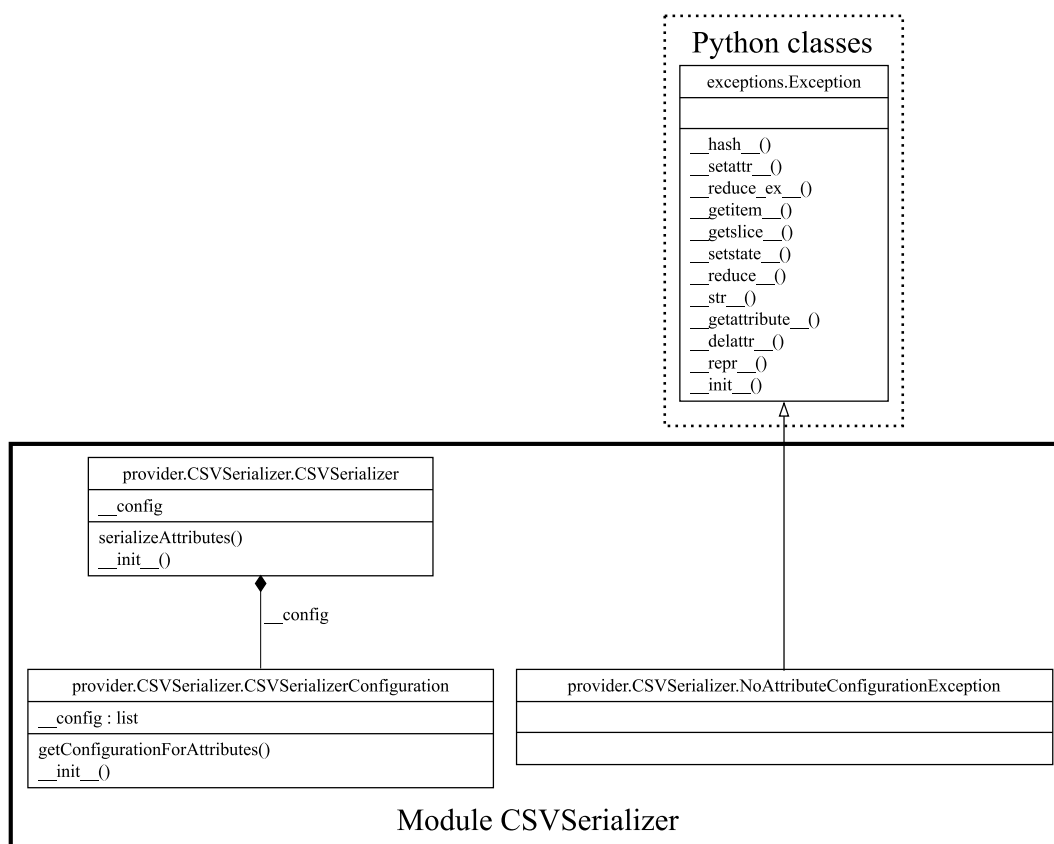


Figure 4.8: Class diagram for the module `CSVSerializer`

Module ProviderConfiguration

The `ProviderConfiguration` class found in this module provides methods for accessing the configuration of an *AAA provider* in a convenient way. It uses a lot of the abstraction provided by the `Configuration` class.

It has also been implemented as a *singleton* in order to avoid loading the configuration multiple times.

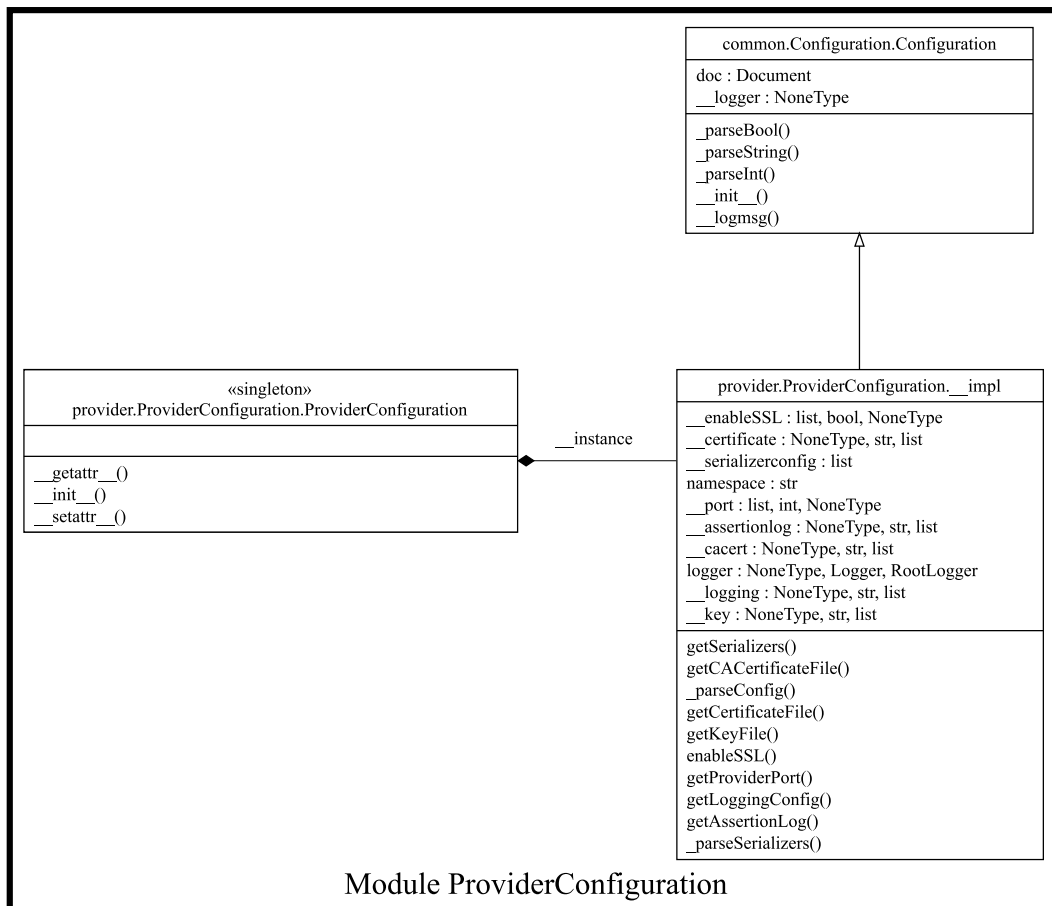


Figure 4.9: Class diagram for the module `ProviderConfiguration`

Module TestServiceInterface

The module `TestServiceInterface` contains no classes, just test-code to send certain attributes over the `ServiceInterface` for testing purposes.

4.4 Common Modules

The modules found in the common directory of the sourcecode are used internally by the other two components of the prototype and are of no relevance to the user.

4.4.1 Module SSL

The SSL-module contains a helper-class for performing client certification validation on the interface between the service and the provider.

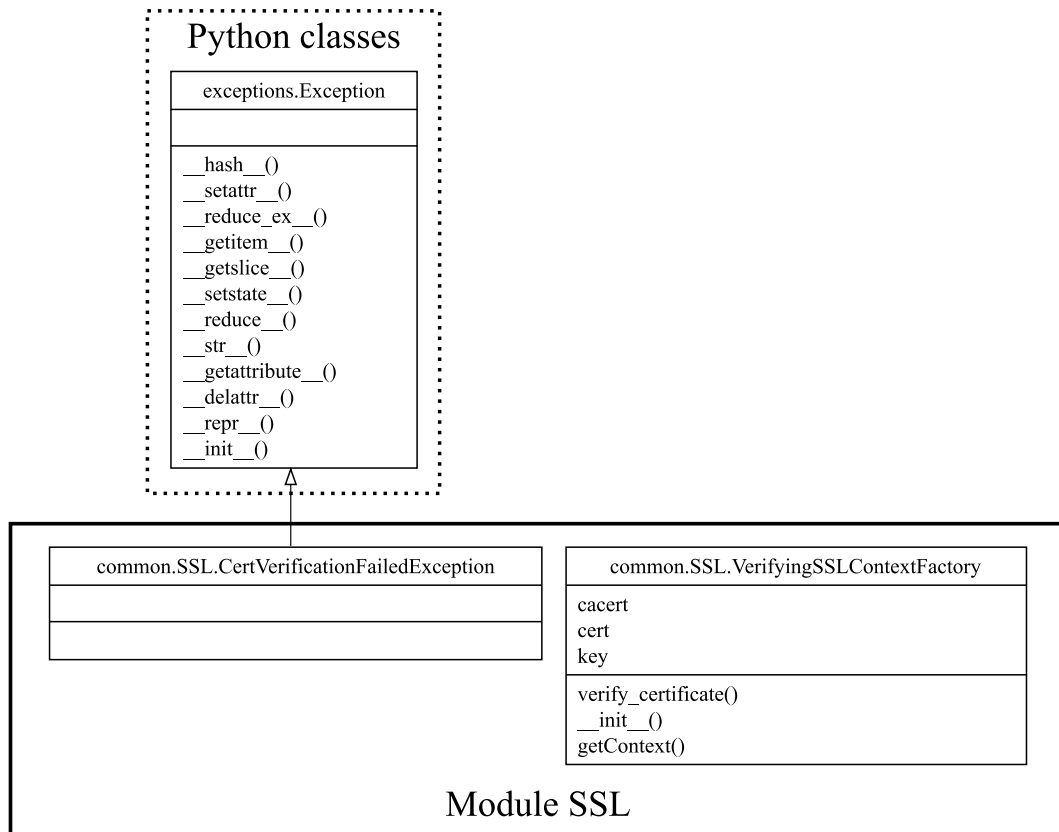


Figure 4.10: Class diagram for the module SSL

4.4.2 Module Configuration

The `Configuration`-module contains common code of both service and provider for parsing the configuration files.

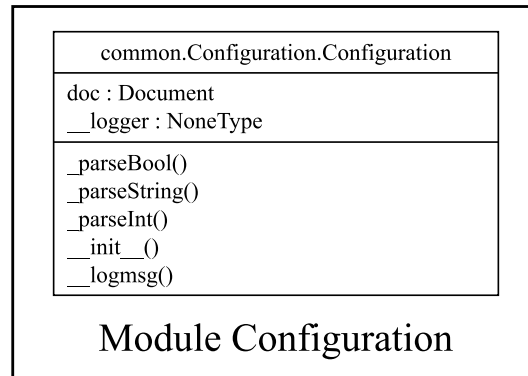


Figure 4.11: Class diagram for the module Configuration

5 Interface Specifications

This chapter contains a brief specification of the two interfaces, the one between the AAA-enabled service and the *AAA service*, as well as the one between the *AAA service* and the *AAA provider*.

Both interfaces are as simple as possible, thus descriptions are quite small and straightforward.

5.1 Application to AAA-Service Interface

The *application-to-aaa-service*-interface is the interface used by an AAA-enabled application to send accounting data to the *AAA service*-component. It is implemented in the module `AccountingBroker` of the *AAA service*. The interface currently only provides a single method: `push_attribute`.

5.1.1 Method `push_attribute`

The method `push_attribute` is used by the AAA-enabled application to push an attribute to the *AAA service*. Its arguments are listed in table 5.1.

<code>app_id</code>	Unique name for the application
<code>uid</code>	<i>swissEduPersonUniqueID</i> associated with the attribute
<code>session_id</code>	Unique id for the current session (see below)
<code>session_count</code>	Number of the current attribute in the current session
<code>session_total</code>	Total number of attributes in the current session
<code>attr</code>	Name of the attribute
<code>val</code>	Value of the attribute

Table 5.1: Arguments of method `push_attribute` (service interface)

It is possible that the transmission of a single accounting step consists of multiple attributes. This is the case in the printing prototype, where both the number of *clicks* and the *product* is generated and has to be transmitted. To bundle multiple attributes, the concept of a session has been introduced. Multiple attributes can be sent in individual requests as a so called session to the *AAA service*. All attributes belonging together need to have the same `session_id` and are numbered within the session.

The `session_id` **must** be unique, thus it's recommended to make it timestamp-based.

Additional information can be found in "Schnittstellenspezifikation Schnittstelle IDS-AAA" ([Mai09])

5.2 AAA-Service to AAA-Provider Interface

The interface between the *AAA service* and the *AAA provider* is not meant to be accessed directly by any other application. It only serves as a backchannel for transmitting the aggregated accounting data to the provider.

The interface is implemented in the module `ServiceInterface` of the *AAA provider* and contains two methods: `push_session` and `push_assertion`.

5.2.1 Method `push_session`

This method is used to send a full accounting session containing multiple attributes to the provider. It has the following arguments:

<code>aaaservice</code>	Name of the <i>AAA service</i> from which this session comes.
<code>data</code>	The data of the session (see below)

Table 5.2: Arguments of method `push_session` (provider interface)

The `data`-argument consists of an array containing all the data of the session, as shown in the following table (order matters!):

<code>uid</code>	<i>swissEduPersonUniqueID</i> associated with the session
<code>app_id</code>	ID of the application
<code>session_id_external</code>	Unique id of the session as given by the application
<code>session_id_internal</code>	Hashed unique id generated by the <i>AAA service</i>
<code>attributes</code>	The AAA-attributes as enclosed list, name first followed by value.

Table 5.3: Contents of the data argument of method `push_session` (provider interface)

5.2.2 Method `push_assertion`

This method is used to send a *SAML*-assertion which has been extracted to the provider, it has the following arguments:

<code>aaaservice</code>	Name of the <i>AAA service</i> from which this <i>SAML</i> -assertion comes.
<code>data</code>	The assertion (see below)

Table 5.4: Arguments of method `push_assertion` (provider interface)

The `data`-argument consists of an array containing the following values (order matters!):

uid	<i>swissEduPersonUniqueID</i> associated with the assertion
assertion	the <i>SAML</i> -assertion itself, base64-encoded
signature	Base64-encoded signature of the assertion, signed with the key of the <i>AAA service</i>

Table 5.5: Contents of the data argument of method `push_session` (provider interface)

6 Open Points

During the development of the prototype, a few points were left open. These were of no direct relevance for the prototype and for a successful end of the project, but are described here nonetheless.

The `ProviderInterface` doesn't handle failures in communication with the provider very well. There is no good error detection and a resend mechanism is missing. Also, only the first provider from the configuration is used.

The prototype as-is is not really ready for distribution. A *twisted application*¹ should be built out of it for easier deployment.

Additionally, an automated installation process should be prepared and the whole application should be delivered in a proper format - maybe as a Python *.egg*

These are the open points so far which have been identified and documented during development of the prototype.

¹<http://twistedmatrix.com/projects/core/documentation/howto/application.html>

Part III

Setup of the Prototype at the University of Bern

7 Integration with the Uniprint-Solution

After successful shibbolisation of the *Uniprint-solution* (a product called *IDS* by *Genius Bytes*), additional parts had to be developed by the vendor for its solution to enable it to:

- Accept a *swissEduPersonUniqueID* as login credential
- Generate a pin-code for non-UniBE customers (UniBE-customers can fetch printjobs using their *Unicard*)
- Generate the accounting-data and deliver it to the *AAA-service*
- Extend the printer-firmware to accept a pincode

The interface between the *IDS-product* and the *AAA-service* has been specified in [Mai09].

As it seemed unlikely that an integration of the *AAA-service* directly on the Uniprint-server would be possible in time, it has been decided to provide a dedicated set-up on a separate server and to have the *IDS*-component report its accounting-information to that server.

This of course is only acceptable for the prototype as it may have severe security implications.

Also, that way, no access to the *Shibboleth*-logs would be possible and thus the part extracting the SAML-assertions from the log, signing them and sending them to the provider had to be tested separately - which was successful.

The following illustration details the integration of the systems during the prototype.

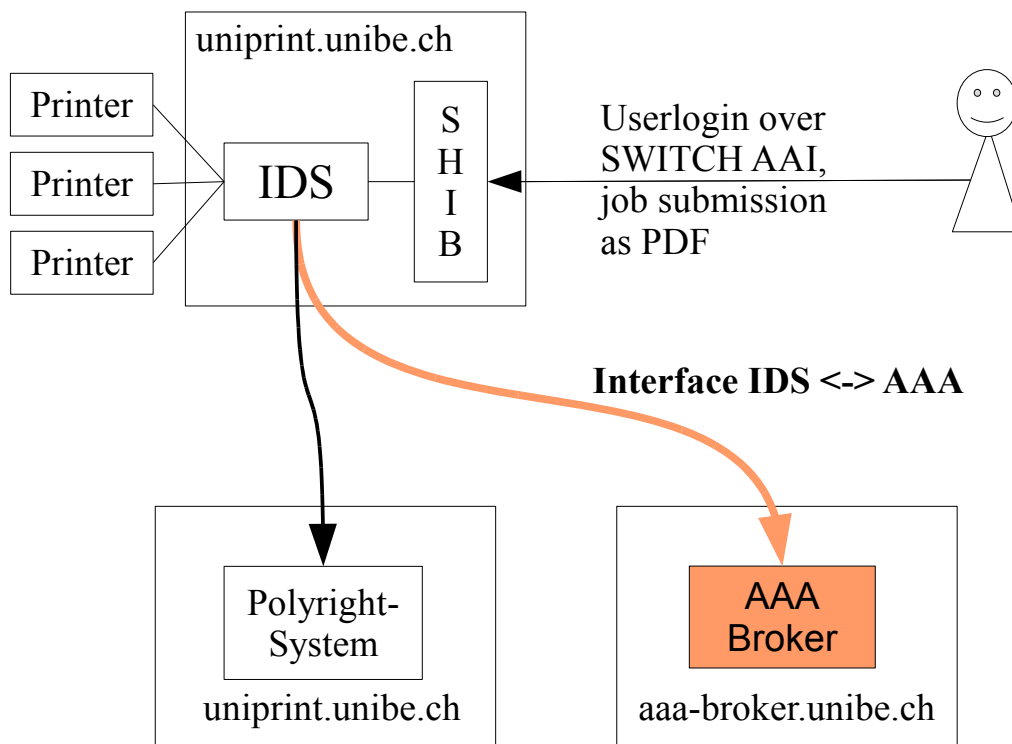


Figure 7.1: Integration with the Uniprint/Geniusbytes-IDS-System

8 Internal Tests with an UniBE-Account

During the development of the prototype, a lot of testing has been performed. The tests were conducted by hand and/or with simple testclients to ensure that the interfaces were implemented correctly.

As soon as all the components were in place and the printing-solution was ready as well (which took more time than expected - see chapter "issues with third parties"), internal testing at the *University of Bern* began.

The scenario tested looked as follows:

- Log in to the printing solution¹ using *SWITCHaai* and a UniBE-account.
- Submit one or multiple printjobs using the webinterface
- Go to one of the *uniprint*-printers
- Release job using the *Unicard*
- Check accounting information received on the *AAA provider host*

In a slightly different form, the same was tested for copy jobs:

- Go to one of the *uniprint*-printers
- Make one or more copies, identified by the *Unicard*.
- Check accounting information received on the *AAA provider host*

The internal tests completed successfully, for both kind of jobs (print and copy).

The number of clicks (A4-page equals) as well as the product type (print color/black and white, copy color/black and white) has been received by the *AAA prototype* from the *IDS printing system* along with the *swissEduPersonUniqueID* of the user.

¹<https://uniprint.unibe.ch>

9 External Tests with a BFH-Account

An AAA-application in the *SWITCHaai-domain* would not make a lot of sense, if it would only have been tested internally with clients from the *University of Bern*.

Due to the good contacts between the *University of Bern* and the *Bern University of Applied Sciences*, thirdparty tests took place between those two institutions.

The scenario tested looks almost identical to the internal tests, with the only difference being the identification of the user at the printer using a PIN-code instead of a *Unicard* - the student cards of the different institutions were not compatible to each other at the time of writing.

In detail, the slightly changed process looked as follows:

- Log in to the printing solution¹ using *SWITCHaai* and a BFH-account.
- Submit one or multiple printjobs using the webinterface
- *Write down the PIN-code generated by the webinterface for that session*
- Go to one of the *uniprint*-printers
- Release job using the the obtained PIN-code.
- Check accounting information received on the *AAA provider host*

Logging in on the printer with the obtained PIN-code is possible in order to perform copy jobs. This has been tested as well.

The tests with a BFH-user were successful - the correct accounting data was obtained in the printing- as well as the copy-usecase.

Note: in order to reduce the amount of work on the BFH-side, the same AAA-provider was used for receiving the accounting-data. This has no implications at all to the distributed concept.

¹<https://uniprint.unibe.ch>

Part IV

Issues Uncovered

10 Issues with Third Parties

During the project a lot of delay has been caused by the fact that there were a lot of dependencies to other parties.

On one side, the project was directly dependent of the *Unicard*-project at the *University of Bern*, it builds upon that project's infrastructure and thus relies on the vendors used in that project.

As the *Unicard*-project is a project with high importance which is carefully observed at the university in general, the development of the prototype had to be set back often. This issue had already been foreseen in the "Projektbeschreibung" ([MKH08]).

Working together with the vendors and developers of the printing solution was not an easy process and caused a lot of additional work - also due to their double efforts in the *Unicard*-project on one side and in this prototype on the other.

As the developer of the *IDS* printing solution, *Genius Bytes*, acted as a subcontractor to the actual printer vendor, *Ricoh*¹, different communication issues arose.

Opposite to what was expected by the project team, shibbolisation of the printing solution took way more time than everyone could think of. On one side, this was caused by access issues and unclear duties in terms of server administration, on the other hand by the very weak knowledge of the technologies used for that process by the vendor.

Additionally, an idea which was meant to speed up this process at first turned out to actually slow it down:

To help the vendor in getting familiar with the technologies, a first shibbolised service was set up initially at his premises. Again, the transition from this test-system to the actual *uniprint*-server caused more work and time to pass than had to be expected.

To summarize things up, a lot of time was consumed in dealing with third-party issues which was then missing on the development side. Without this major effort it has to be assumed that the development could have been pushed further.

¹<http://www.ricoh.ch>

11 Issues with Shibboleth

The prototype interferes only slightly with *Shibboleth*, thus not many issues were to expect.

In an early phase of the project the *Shibboleth* developers were contacted to discuss possible ways and issues of the integration with *Shibboleth* and possible benefits of an integration to the *Shibboleth*-project.

Unfortunately the *Shibboleth*-project didn't seem to be very interested in a discussion. During the development of the prototype, it was therefore paid attention to have only as loose coupling with *Shibboleth* as possible.

Another possible issue lies in the way, how the *SAML*-assertions are currently retrieved from *Shibboleth* for signing and sending to the *AAA-Provider*:

At the time of writing the only known way of extracting those assertions was to set *Shibboleth*'s logging priority to `DEBUG` and to monitor it's logfile for any assertions logged.

Of course, this is not the ideal approach. First, running the `DEBUG`-priority on a productive server is not really the proper way of operations. Second, if *Shibboleth*'s logformat changes, the prototype has to be adjusted.

For a productive accounting-application together with *Shibboleth*, the project team strongly recommends adding an interface to *Shibboleth* for at least getting the *SAML*-assertions in a proper way. Furthermore a tighter integration would benefit both projects.

12 Other Issues

Besides the direct issues, some others were found which affect AAA and/or micropayment more directly.

The most important of them will be presented in the following sections. This is not an exhaustive listing of possible issues.

The next part of this report, "Possible Directions for Future Projects" addresses some of them partly and proposes a way to go for future projects.

12.1 Location of the Provider

The design of the prototype components has been following the schema of the AAI domain, providing so called *AAA services* and *AAA providers*.

Accounting transactions get relayed from the services to the providers at the respective home organisations of the user. This whole model is up for discussion:

- Should the transactions really be delivered directly to the home organisations of the logged in users?
- Should the transactions go to the organisation hosting the service instead?
- Should a central authority take care of the transactions?

These are not technical but more political and/or organisational aspects which could not have been cleared in the prototype phase. The community needs to take up discussions on the different aspects in this matter.

12.2 Laws Regarding Banking

The exchange of money in terms of brokerage, banking and credits is more and more strongly regulated.

Various laws regarding transactions and compliance could interfere with projects in that domain. Clearly, the whole concept has to be reviewed by qualified personnel before seriously exchanging money over an AAA-infrastructure between institutions.

As the prototype project had a clearly technical nature, further analysis was not possible here.

12.3 Unclear Attributes

Even in the small domain of printing, the project team discovered early that not everyone speaks of the same terms and understands the same regarding them.

As an example, vendors of printing solutions speak of *clicks* (meaning A4-page equivalents, an A3-page for instance would be two *clicks*), while most people speak of pages.

Products is another issue: In the prototype, the different "products" like black/white copy, color print etc. are identified by numbers which come directly from the devices and are only standardised within the *UniBE*-domain, even only among devices from the same vendor.

To be able to deploy such an infrastructure nationwide and between multiple institutions, an agreement on the attributes and their respective values has to be made - like it has already been done in the *SWITCH* *ai-federation*.

Again, this is a political issue - not a technical one.

Part V

Possible Directions for Future Projects

This part covers possible topics for follow-up projects after the end of *UNIBE.2*.

These topics came up during the development, the integration and the testing of the prototype and are, what the authors consider, useful and strategic directions which could follow.

13 Extend Prototype to include PHBern

Soon in the future, a new campus will be opened by the University of Bern and the PHBern, the so-called *Von-Roll-Areal*. On this campus, there will be an even tighter integration between the two institutions - thus sharing of resources will become more and more important.

An extension of the prototype to include PHBern would consist of

- Develop a common strategy of vendors and services used together
- Further set productive the prototype to make it usable in a real-world environment
- Create contracts regarding the exchange of money

Such a project would require a lot of project leading skills as well as at least one developer. Furthermore, legal departments of both institutions would have to be heavily involved.

14 Standardisation of Attributes and Brokerage

During the implementation of the prototype it got more and more clear, that accounting in the *SWITCHaai-infrastructure* isn't a straightforward process.

Before continuing in the current direction, attributes should be standardised nationwide between the institutions - like in the *SWITCHaai*-environment, where this has already been done.

The questions of the monetary transfers, contingents etc. remains unsolved and should be discussed further in the community.

This would be less a technical project but much more an organisational one. It would require thorough negotiational strengths and a lot of communication between the institutions.

15 Mobile-/Micropayment?

As a third option for a future project, a study regarding mobile-/micropayment in the swiss higher education landscape comes to mind.

Where could micropayment be used? For what services? Where are the monetary flows between the institutions and their clients?

This project would mainly consist of a study shedding light into those questions - ending up with a lot of usecases. Also, the legal side of mobile-/micropayment must be investigated in great depth before actually starting out.

Additionally, one or more of the usecases found could be explored more in-depth. Micropayment with other devices besides the *Unicard*, for instance mobile phones, seems to be an option. Usecases like paying printing costs with the mobile phone bill seem to be interesting.

Cooperation with other parties like mobile operators could as well be part of the project.

16 Swiss-Wide SOA Architecture?

At last - but probably with the biggest implications - thoughts of a nationwide service oriented architecture (*SOA*¹) came to mind.

In the future, higher education institutions in Switzerland will more and more need common interfaces for data-integration and other services. The University of Bern for instance already has the need for the exchange of student-data in the *BeNeFri-collaboration*².

The goal of this project would be to determine the possibilities, means and consequences of a swisswide service oriented architecture.

¹http://en.wikipedia.org/wiki/Service-oriented_architecture

²Bern-Neuchatel-Fribourg

Part VI
Conclusion

17 Conclusion

This report closes the *SWITCH AAA* project *UNIBE.2* which - despite multiple issues - is regarded as successful from our side.

First of all, the prototype implemented the vital technical parts of the whitepaper ”*Enhancing SWITCH_{aa}i with Micropayment Functionality for Swiss Universities*[SCR06]”.

The prototype also lead to a shibbolised, web-based printing system at the *University of Bern* and printing with a third party account was successfully demonstrated. Thus the prototype clearly shows actions and future directions to take.

A lot of lessons learned on all sides where another benefit of the project - both for the *University of Bern* as well as for other involved parties. The most important of them were also documented in this report.

The project team now hopes that the outcome of the prototype can and will help further projects, serving as a basis for the needed discussions, analyses and political steps.

The project team wants to thank all involved parties, especially *SWITCH* for their ongoing support as well as for funding the research in this project, thus making it possible at all.

Part VII
Appendix

Contents of Deliverable

The deliverable is split up in multiple directories containing the respective content.

Toplevel, three directories can be found:

- `conf` - example configuration for service and provider
- `doc` - project documentation
- `src` - sourcecode of the prototype, split up by package (*common*, *service* and *provider*)

The `doc`-directory contains everything related to this report in the `report`-subdirectory, including classdiagrams and illustrations.

In the `api-doc`-directory, the API-documentation (documentation of the packages, classes and methods) can be found. This is generated documentation.

The `reporting`-directory contains the quarterly reports, the final report as well as the SBF-annual-report.

Besides the documentation in the subdirectories, various other project related documentation can be found in the `doc`-directory.

Installation Instructions

Installation of the prototype is straightforward.

Beside a running *Python 2.5*-installation, the twisted-framework is needed, which can be downloaded from <http://twistedmatrix.com>.

After installation of the prerequisites. two environment variables must be set:

- `PYTHONPATH` - must contain the toplevel src-directory with the packages of the prototype
- `AAACONFIG` - must point to the configurationfile of the service/provider.

Following those preparations, the service and the provider can be started by launching their respective main methods.

Overview of Configuration Files

Overview

Configuration of the prototype is a straightforward process which won't be detailed in this report. Instead, well documented configuration-templates are provided with the deliverable.

AAA Service Configuration

The *AAA service* is configured using the following three configuration files:

- `service.xml` - main configuration file of the service
- `aaa-providers.xml` - provider metadata
- `logging-conf` - configuration of the Python logger

AAA Provider Configuration

The *AAA provider* is configured using the following three configuration files:

- `aaa-provider.xml` - main configuration file of the provider
- `csv-serializer.xml` - configuration for the csv-serializer
- `logging-conf` - configuration of the Python logger

Bibliography

- [Mai09] P. Mainini. Schnittstellenspezifikation: Schnittstelle IDS - AAA. Enclosed in project delivery as interface-ids-aaa.odt, 2009.
- [MKH08] P. Mainini, U. Kienholz, and C. Heim. Projektbeschreibung. Enclosed in project delivery as Projekt-Beschreibung-UniBE-AAA.doc, 2008.
- [SCR06] P. Schnellman, P. Chénais, and A. Redard. Enhancing SWITCHaai with Micropayment Functionality for Swiss Universities. http://www.switch.ch/aai/docs/AAI_Micropayment_Whitepaper.pdf, 2006.

List of Figures

4.1	Component-Overview AAA Prototype	12
4.2	Overview of Components and Interfaces in an AAA-Enabled Webservice	13
4.3	Class diagram for the module WebServer	16
4.4	Class diagram for the module AccountingBroker	18
4.5	Class diagram for the module ProviderInterface	20
4.6	Class diagram for the module ServiceConfiguration	21
4.7	Class diagram for the modules AccountingProvider and ServiceInterface	24
4.8	Class diagram for the module CSVSerializer	25
4.9	Class diagram for the module ProviderConfiguration	26
4.10	Class diagram for the module SSL	29
4.11	Class diagram for the module Configuration	30
7.1	Integration with the Uniprint/Geniusbytes-IDS-System	38

List of Tables

5.1	Arguments of method <code>push_attribute</code> (service interface)	31
5.2	Arguments of method <code>push_session</code> (provider interface)	33
5.3	Contents of the data argument of method <code>push_session</code> (provider interface) . . .	33
5.4	Arguments of method <code>push_assertion</code> (provider interface)	33
5.5	Contents of the data argument of method <code>push_session</code> (provider interface) . . .	34