Accounting and Monitoring of AAI Services

Phase 3

Deliverable D3
AMAAIS: Workflow, Accounting Model, and API

The AMAAIS Partners

University of Zürich (UZH), Switzerland
SWITCH (SWITCH), Switzerland
ETH Zürich (ETHZ), Switzerland

For more information on this document or the AMAAIS project, please contact:

Martin Waldburger
University of Zürich
Department of Informatics (IFI)
Communication Systems Group (CSG)
Binzmühlestr. 14
CH-8050 Zürich
Switzerland

Phone: +41 44 635 4304
Fax: +41 44 635 6809
E-mail: waldburger@ifi.uzh.ch

Document Control

Title: AMAAIS: Workflow, Accounting Model, and API Model, and API

Type: Public

Editor(s): Guilherme Sperb Machado, Christos Tsiaras

E-mail: machado@ifi.uzh.ch, tsiaras@ifi.uzh.ch

Author(s): Guilherme Sperb Machado (GM), Christos Tsiaras (CT),
Martin Waldburger (MW), Matteo Corti (MC), Patrik Schnellmann (PS),)
Andrei Vancea (AV), Burkhard Stiller (BS)

Doc ID: D3

Delivery Date: 09.12.2011

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
0.1	2010-10-11	GM	Deliverable created with content already compiled from AMAAIS members.
0.2	2010-10-17	CT	Included content to the Common Accounting Model (text and figures).
0.3	2010-10-20	GM, MC, AV	Introduction, Workflows, Test Cases, and Common Accounting Model sections completed (First Version draft).
0.4	2010-10-27	CT	Architecture Review text.
0.5	2010-10-27	GM, PS	Changes in the introduction. Text for the API definition section. Text for the Summary.
0.6	2010-10-19	CT	Architecture Review corrections.
0.7	2010-11-21	GM	Executive Summary and Summary/Conclusions. Update for the Test Cases and API definition.
0.8	2010-11-29	GM	Executive Summary corrections, included as anex the Accounting Metadata, Profile, Messages specification. Modifications in the API section.
0.9	2010-11-30	CT	Final editing.
1.0	2010-12-06	GM, MW, BS	Final updates.

Legal Notices

Contents

1 Executive Summary

The goal of the AMAAIS (Accounting and Monitoring of AAI Services) project – a collaboration between the Communications System Group (CSG) at UZH, SWITCH, and ETHZ – is to extend the current Authentication and Authorization Infrastructure (AAI) with accounting and monitoring support, enabling inter-domain accounting and the management of the AAI. The AMAAIS project is structured into project phases. Phase 1 [1] was mainly concerned at collecting use case scenarios, requirements, and the construction of a high-level architecture. Phase 2 focused on the fine design of the architecture, the implementation of the accounting and monitoring architecture – resulting in a prototype –, and its deployment. This document, which describes the first deliverable of Phase 3 (called Deliverable 3), is focused on the use cases workflow definition, test cases definition, and the API definition – which also is part of a broader concept to re-factor the AMAAIS implementation code from previous phases.

The re-factoring process of the AMAAIS implementation arose from two main requirements that are the key for the success of AMAAIS. The first requirement is to support a high level of project sustainability. After the conclusion of the AMAAIS project as a whole, the implementation and results should be available to and maintained for the community or federations which are involved in the development, or, actually, using the system. The use of standards, extensible components, documentation, and stable functionalities enable the AMAAIS to be a sustainable project. The second requirement is to couple the AMAAIS system to the existent Shibboleth software. This closer integration is part of the broader goal of project sustainability, since Shibboleth is supported by many federations around the globe.

Deliverable 3 (D3) targets any kind of users (from large educational institutions or private organizations, to small-medium corporations) that already has a Shibboleth-based AAI infrastructure and wants to enable accounting and monitoring. Moreover, this document is also interesting for users that want to deploy the AMAAIS project, or to have a better understanding of the API of each package to come up with specific Collectors or “data fetchers”.

The main achievements of AMAAIS Deliverable 3 are (1) the implementation design re-factoring process aligned to Shibboleth standards, and (2) the use cases workflow with its test cases. As a result, the activities performed in (1) brought a better organization of the implementation components related to its function methods (API), and the activities in (2) brought a better and clear view of which are the normal or abnormal situations considering the AMAAIS Use Cases in a production scenario.

Specifically, the implementation design re-factoring process will result either in the creation of new implementation components (*i.e.*, Java classes which should be implemented from scratch) or the adaptation of existent ones. For example, in AMAAIS Deliverable 2 components like “remoting” or “AttributeFiltering” were embedded into other classes, in diverse parts of the code. In AMAAIS D3, “remoting” and “attributeFiltering” packages are explicitly and separated components that can be used for different purposes, in different classes. In addition, the new “dataFetcher” package brings an extensible manner to fetch data from different Meters: from a simplistic log file to a database that has a specific database driver. The existence of a “metadataHandler” package also brings the functionality of a central

metadata in the Shibboleth standards. It has the responsibility to grab important information (*e.g.*, accounting server certificates) from a central authority and make it available to internal AMAAIS components (*e.g.*, the "remoting"). In summary, the project will reach a higher maturity level by means of its code base being closely aligned with Shibboleth standards and goals.

The use cases workflow with its test cases results in a better view of what the AMAAIS project is aiming to achieve when deployed in a production environment. The wokflow definition shows that, *e.g.*, under normal conditions the system should process *X* quantity of SMSs per minute, and therefore the Accounting Server should persist that in a way to not cause any service disruption. The test cases, on the other hand, based on the workflow definition, aim to predict in which situations the system might crash or suffer a disruption. Therefore, the test cases definition – which are high-level described – tried to support the implementation and fine-grained test cases to come.

The AMAAIS D3 forms the solid basis for the Phase 3 implementation phase that follows. Even though the AMAAIS Deliverable 2 also presented an implementation phase resulting in the AMAAIS prototype 1.0, the implementation phase expected in AMAAIS Phase 3 is planned to include the new functionalities and code re-factoring, as previously mentioned. Therefore, the code re-factoring is a process that will be extended throughout the whole AMAAIS project development.

2 Introduction

A Shibboleth-based Authentication and Authorization Infrastructure (AAI) enables users to access different web resources in a common manner by providing a single-sign-on interface for login. The AAI makes use of the Federated Identity Management concept that allows the use of a single user identity for different services beyond the user's home institution domain. Such characteristic plays an important integration and organizational aspect for institutions: it not only eases the users' access to resources, but also makes the management process more convenient. Pfitzmann et al. [2] provides a comprehensive overview of available federated identity approaches and protocols. Within the scope of the AMAAIS project (Accounting and Monitoring of AAI Services) [3], Shibboleth is used and the considered type of federation includes primarily institutions of higher education in Switzerland.

The goal of the AMAAIS project – a collaboration between the Communications System Group (CSG) at UZH, SWITCH, and ETHZ – is to extend the current Shibboleth-based AAI with accounting and monitoring support, enabling inter-domain accounting and the management of the AAI. During 2009, the AMAAIS project ran the Phase 1. The results of AMAAIS Phase 1 are published in the Deliverable 1 [1] which was primarily concerned with service-independent accounting and monitoring (*e.g.*, Service Provider and Identity Provider), as well as with service-dependent scenarios (*i.e.*, SMS and Printing services) and the composition of a high-level architecture to rely on. The AMAAIS Phase 2 was mainly focused on the refinement of the architecture and the implementation of the architectural components. The results of Phase 2 are published in the Deliverable 2 [4]. One of the main points touched by the Phase 2 is the importance of well-defined interfaces between components to enable previously defined requirements on Phase 1 (*e.g.*, extensibility and reliability). Moreover, Phase 2 highlighted the ease to develop other service-dependent components with the use of the defined interfaces. For example, the Phase 2 implementation enabled users to write their own Collector (*e.g.*, a "log parser") in order to account resource usage of a given resource. In summary, Phase 2 deliverable was useful to explicit the technical implementation details and to show how users can benefit from it, demonstrating how to deploy or even extend project's components.

During AMAAIS Phase 1 and 2, the AMAAIS team gained considerable experience and knowledge in discussing crucial aspects of the AMAAIS implementation, like performance, security, privacy, extensibility, manageability, and usability. Also, aspects like charging and billing were recurrently mentioned, since accounting is a process that precedes charging and billing, and therefore it becomes important to discuss which kind of information will be persisted in the accounting database. Based on these observations, the AMAAIS project was extended to a Phase 3, which is focused in an extension beyond the accounting and monitoring extensions of Phases 1 and 2.

Phase 3 extensions cover (a) re-factoring of AMAAIS implementation to meet Shibboleth requirements, (b) testing and operational process definitions, (c) trials, (d) inter-domain accounting record exchanges, (e) additional use cases and the charging process of the related services, (f) database optimizations for accounting record handling, (g) accounting attributes composed from group-based values, and (h) security and privacy aspects of these extensions. These extensions from (a) to (h) should follow the general concept of the AMAAIS architecture developed in Phases 1 and 2.

More specifically, (a) focuses on adapting the AMAAIS implementation into the Shibboleth software; (b) intends to compile use case workflows and test definitions in order to have a broader view of the normal and abnormal flows that a certain use case has in relation to the whole AMAAIS system. This activity will also support the implementation phase, including the API definition and test cases specific to each developed component; (c) will test AMAAIS with both the SMS and printing use cases using on-production data, meaning that tests will rely on real input provided by ETHZ and UZH. Aspects like reliability and robustness should closely be taken into observation during the trial; (d) looks to the anonymity, scalability, and a common configuration/agreement between domains when exchanging accounting records; (e) focuses on identifying, selecting, and applying additional use cases besides the existent SMS and Printing, like, *e.g.*, network usage and storage; (f) focuses on the investigation into how replication, caching techniques, or other optimization methods can positively influence the scalability of the AMAAIS accounting; (g) aims to develop a way that decrease the overhead in the accounting processing and protocol by enabling the transport of group-based accounting attribute's values; and (h) evaluates and enables the security and privacy of the accounting information, mainly when exchanging accounting information across different network domains – which is prone to exposure.

The *Deliverable 3* touches the items *(a) and (b)* of the ones described above: the re-factoring of AMAAIS implementation to meet Shibboleth requirements *and* the testing and operational process definitions.

In the scope of the item (a), in fact, the re-factoring process is a process that will be present also during the upcoming deliverables – since it should be faced as a continuous process. The experience gained during Phase 1 and 2 about aforementioned crucial aspects, brought the AMAAIS team closer to Shibboleth interests and requirements. In order to couple the AMAAIS implementation to the Shibboleth software, the need for partial re-factoring of the AMAAIS code base became apparent – however, as mentioned before, not breaking conceptual ideas developed in Phase 1 and 2. The re-factoring process aims to support project sustainability. After the conclusion of the AMAAIS project as a whole, the implementation and results should be available to and maintained for the community or federations which are involved in the development, or, actually, using the system. The use of standards, extensible components, documentation, and stable functionalities enable the AMAAIS to be a sustainable project. One of the main points of the AMAAIS Deliverable 3 is to present an enhanced Common Accounting Model and the API definition, part of the AMAAIS re-factored implementation. Moreover, presented as an appendix, a preliminary version of "Accounting Service Messages, Metadata, and Profiles" was included. The accounting message, metadata, and profiles definition is also part of the re-factoring process and it has been developed in a cooperation with Shibboleth and AMAAIS team. The document shows, *e.g.*, how accounting messages should be formed when AMAAIS components invoke some API methods.

In the scope of the item (b) and based on the use cases already discussed on Phase 1 and 2 (SMS and Printing), workflows were compiled to express normal operation of a considered service. High-level test cases were also compiled to, looking from an overview perspective, what should be tested related to functionality, performance, and security. Both workflows and test cases focus on specific aspects of use cases, meaning to be part of service-dependent segment of the project.

This document is organized as follows. Section 3 presents a review of the AMAAIS architecture, just highlighting the main architectural concepts and describing the architecture that the project will rely on for upcoming deliverables – even if it is totally compatible to the architecture presented on Phase 1 and 2. Section 4 presents the use case workflows for the SMS and Printing use cases. Section 5 describes the high-level test cases based on the developed workflows, which will be the solid basis for the implementation test cases and trials to come. Section 6 presents the Common Accounting Model and its use case extensions. Moreover, Section 6 explains in more detail why and what changed from the Common Accounting Model presented on AMAAIS Deliverable 2 to the current version. Section 7 presents the implementation components and its API (Application Programming Interface) for the implementation re-factoring phase to come. Finally, Section 8 presents the Deliverable 3 summary and key conclusions drawn. At the end of this deliverable, in the Appendix section, the first version of “Accounting Service Messages, Metadata, and Profiles” document was included (still under development).

3 Architecture Review

Deliverable 3 is highly focused on the re-factoring of the AMAAIS project, in order to adapt its implementation to the Shibboleth software. However, the architecture presented in AMAAIS Deliverable 1 and 2 has no major changes concerning the interfaces and the system deployment which should be followed to the rest of AMAAIS Phase 3 (even with the re-factoring). Therefore, this section aims to present a review of the AMAAIS architecture, its main components and interfaces, and to briefly describe the accounting process.

In summary, this document contains a section called “Architecture Review” to give the reader an end-to-end view of the whole AMAAIS system, including the modifications. Such broad and complete view of the system is important to guide the reader in the upcoming sections, since the components will be mentioned in many parts along this document. Therefore, for the remainder of this document it is assumed that the reader has a complete knowledge about the AMAAIS architecture.

It is important to highlight that the architectural figures presented in Sections 3.2 and 3.3 contain some optional components. These optional components are service-specific, meaning that they relate to the chosen use cases for the AMAAIS project (*i.e.*, Printing and SMS).

3.1 Considerations on the Changes from AMAAIS Deliverable 2 to Deliverable 3

Despite of no major changes in the AMAAIS architecture of what was presented in Deliverable 1 and 2, the accounting process (Section 3.5) was modified to this deliverable due to a new approach for representing accounting data. *I.e.*, the AMAAIS system will be based on services generating events, which can be – or not – a session event (for more details consult Section 6). The changes made to the accounting process aim to increase adaptability to other services in future, also to turn the integration with Shibboleth more straight forward.

The architecture schema presented in AMAAIS Deliverable 2 is still valid even with the modifications made to the accounting process. All components and their respective functions remain the same.

3.2 Architecture Overview

In this section the AMAAIS architecture is presented. Figure 1 illustrates all the components of the architecture, the interfaces, and the relation between them. Harmonized with the second phase of AMAAIS, efficient extension of the system is supported by defining new service-dependent Meters and Collectors.

Reviewing from Deliverable 2, a Meter is always paired with a unique Collector. Thus, every system and service event corresponds to a dedicated Meter-Collector pair. In more detail, the Meter could be considered as the log data that will be produced after every event occurred. The Collector parses the log data, in order to create the input data for the

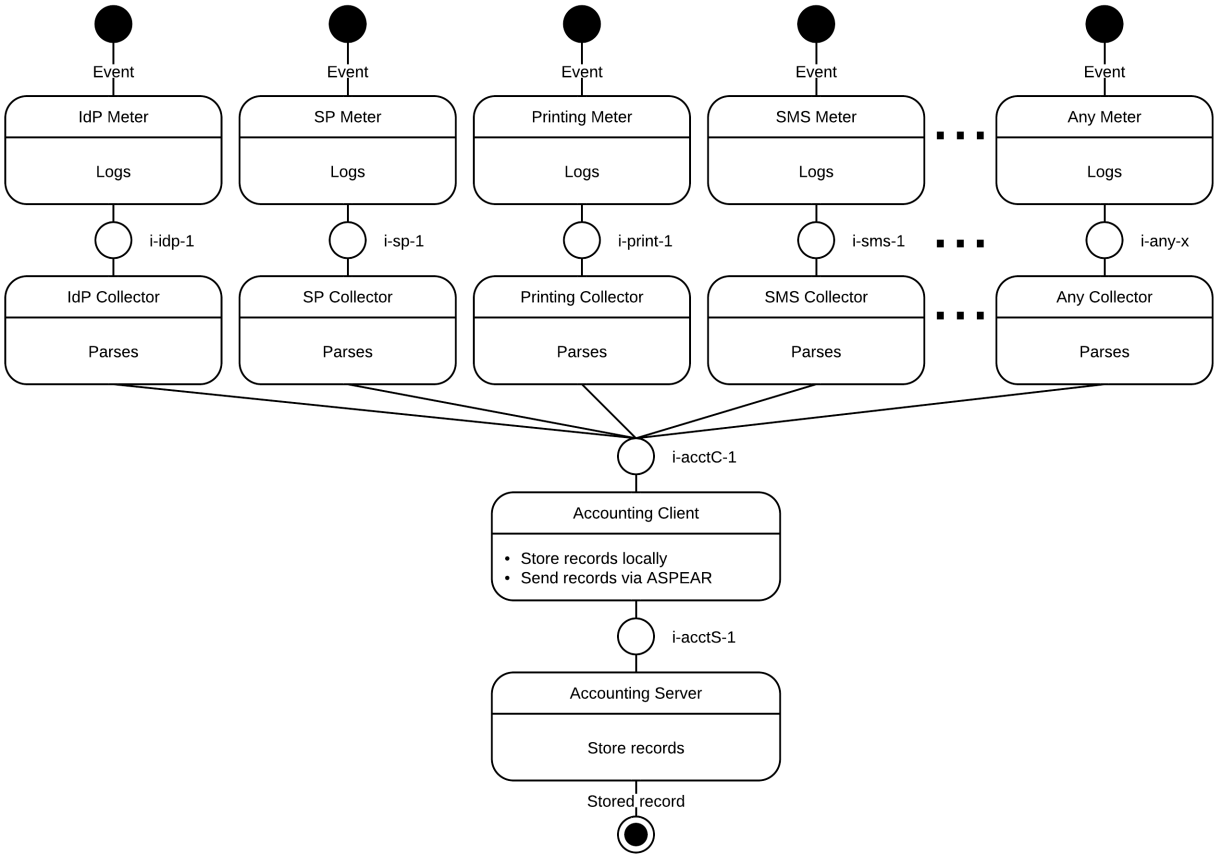


Figure 1: AMAAIS architecture.

accounting client message. This message contains the event's information that need to be accounted like timestamps, type of service, user ID, SP, IdP etc. Finally, the accounting client is using the ASPEAR accounting protocol in order to forward those information to the accounting server, where will be stored permanently. If the accounting server is not available, a copy of the event is stored by the accounting client locally, until the server stays available again. Accounting records could be exchanged between different Accounting Servers if required (e.g. in case of accessing resources offered by another institution), even if the accounting servers belong to different domains. However, during accounting record exchange, privacy as well as accounting policies between the two servers may result some attributes, like the user ID, to be filtered.

3.3 System Deployement

Figure 2 illustrates the AMAAIS architecture on the physical machine level. Every system, as well as service provider dependent services, will be handled on the same machine. In more detail, in the IdP Machine the following processes are executed. Meter implements the event logging procedure. The Collector and the Client together, form another single process which implements the accounting procedure. Equivalent machine level architecture is applied for all services. However, how the infrastructure is organized is not a critical

parameter for AMAAIS. The use of Virtual Machines (VMs) instead of physical machines, and different Data Base (DB) servers for the accounting DB is also possible.

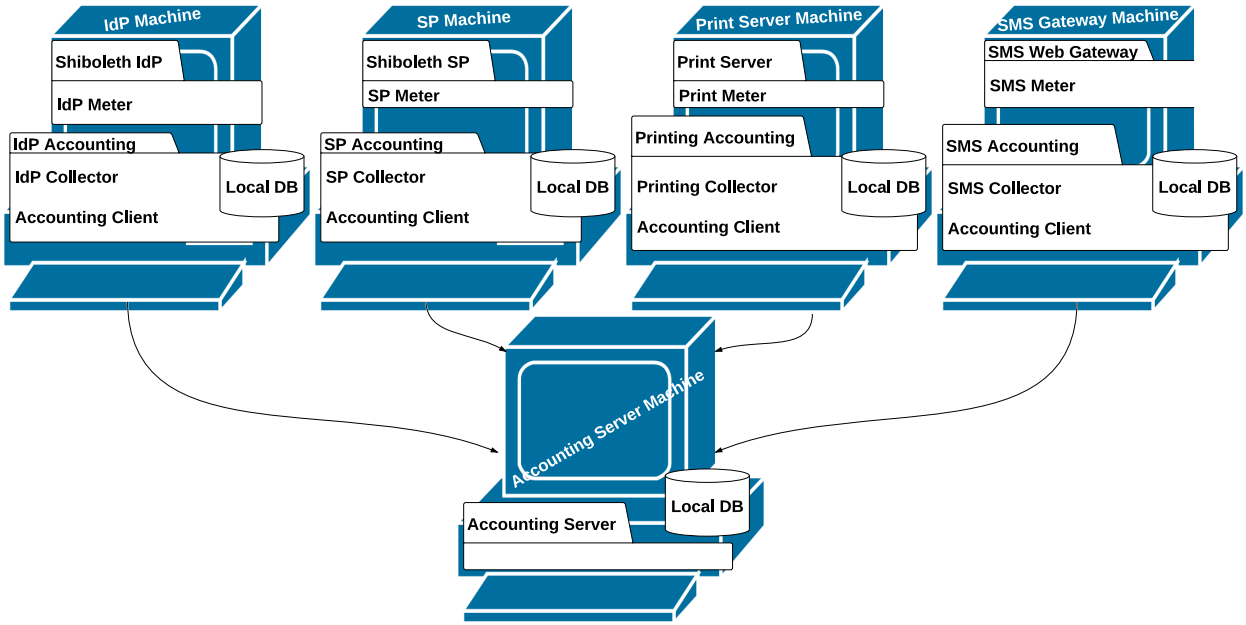


Figure 2: AMAAIS Deployment Covering Service-independent and Use Case-specific (SMS, Printing) Components.

3.4 Interfaces

Table 1 lists all interfaces of the AMAAIS architecture. A detailed description of those interfaces can be found on paragraphs 3.3.1 - 3.3.6 of D2 [4]. Note, that the interfaces define methods which need to be provided in every additional releases of each interface (*e.g.*, i-acctc-1, i-acctc-2, etc), regardless the programming language, or the communication type used during the implementation.

Interface ID	**Component providing the interface**	**Component using the interface**	**Description**	**Type/Protocol**
i-idp-1	IdP Collector	IdP Meter	To send IdP-related metering data to the collector.	File based, using Shibboleth IdP log files
i-sp-1	SP Collector	SP Meter	To send SP-related metering data to the collector.	File based, using Shibboleth SP log files
i-print-1	Printing Collector	Printing Meter	To send metering data about print jobs to the collector.	File based, using log files of the print server
i-sms-1	SMS Collector	SMS Meter	To send SMS-related metering data to the collector.	Database based, using log information of the SMS web interface
i-acctc-1	Accounting Client	Collectors	To create and terminate accounting sessions, to create accounting events, and to send accounting events/sessions to the server.	Java API
i-accts-1	Accounting Server	Accounting Client and Server	To send/forward accounting events to a server. Both the client and the server can use this interface to send accounting events to a server.	Based on ASPEAR

Table 1: Interfaces of AMAAIS.

3.5 Accounting Process

The Meters are independent from the rest of AMAAIS components. The Meter “daemon” runs constantly and creates log messages. Therefore, the log messages are parsed by the corresponding collectors. As more detailed in Section 6.2, log events should be interpreted as accounting events or session-based accounting events, depending on the service and how the Collector component is developed. Thus, if there is an accounting event that need to be proceeded, the collector will create and send the record to the accounting client. Later, the accounting client will forward the accounting records to the accounting server which stores them in a DB. Figure 3 presents the high-level sequence diagram of the procedure described above.

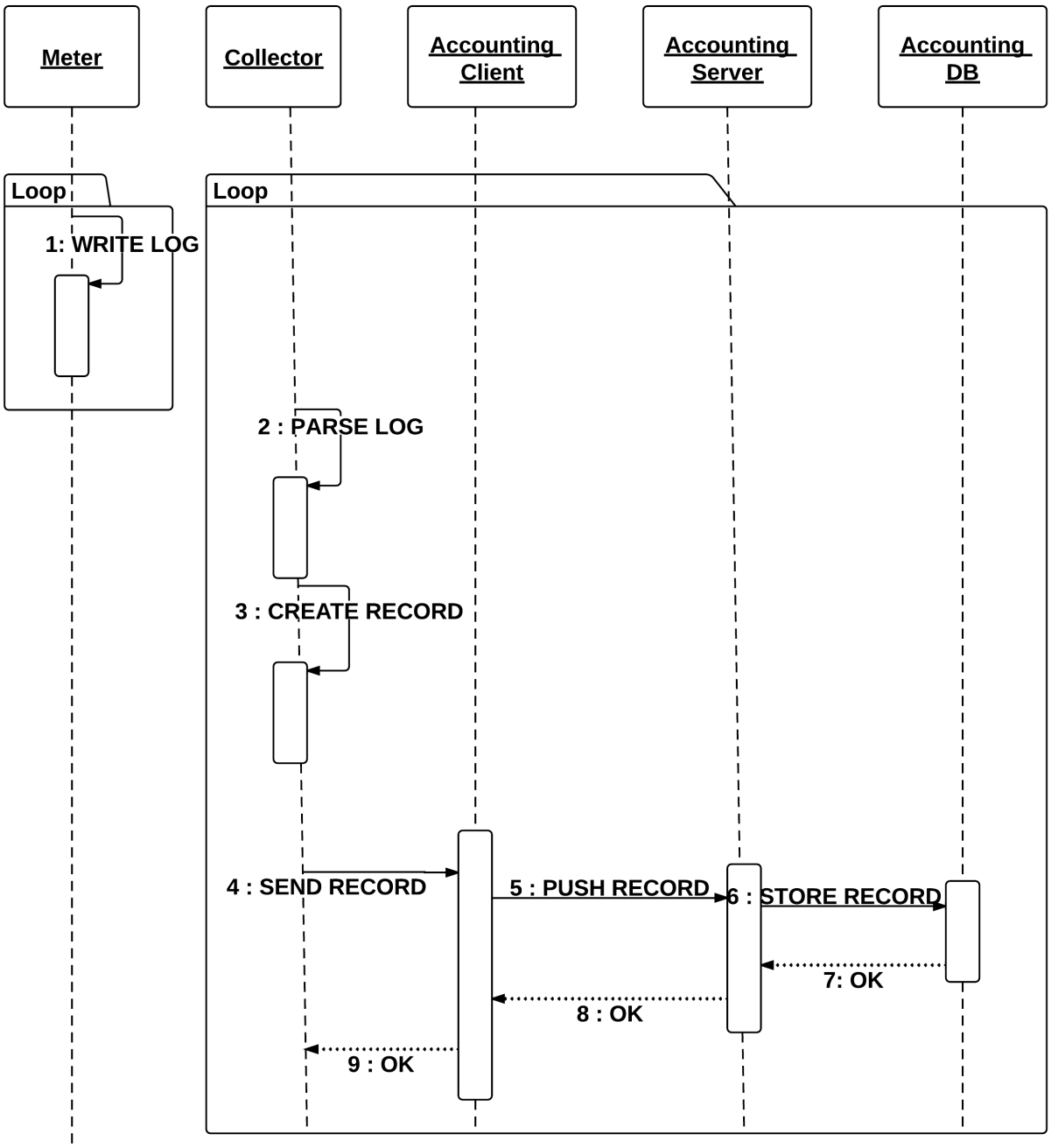


Figure 3: Accounting process

4 Workflow Definition

This section presents the workflow definitions for the AMAAIS use cases: Printing and SMS. The workflow definitions could be totally different if applied to other use cases, *e.g.*, video streaming or network bandwidth. Here, the meaning of a workflow is the "description of flow or progress of work considering a given scenario". The reason to come up with workflows definition for each of the AMAAIS use cases is to better understand what are the normal and abnormal flows of the use cases processes. For example, understanding how the SMS use case behaves in a normal condition, it would be possible to infer implementation test cases that could lead the system to an abnormal state.

From a more high-level perspective, the workflow definition is important to enhance AMAAIS quality. In the end of AMAAIS Phase 3 the system is intended to be deployed in on-production environments for the AMAAIS use cases. Therefore, the workflows definition presented in this section is beneficial mainly to guide the test cases definition (Section 5) and the implementation phase – along the whole project duration.

4.1 General infrastructure

The general infrastructure assumptions for the use cases are:

- The ETHZ IdP log files are available and can be read from any machine (i.e., the accounting server can be installed on any machine);
- The accounting server is installed on a dedicated machine.

4.2 SMS

In the next subsections the workflow definition and its assumptions based on the SMS use case are presented.

4.2.1 Components and infrastructure

The components and infrastructure that are specific to the SMS use case are listed below:

- A web application (SP) sends the SMS and generates the entry in the log file;
- The SMS send log is directly stored in a database table;
- Accounting client and collector can be run from any machine.

4.2.2 Workflows

The workflows are distinguished between a standard workflow (see Section 4.2.2.1) and a workflow with more than one web application (see Section 4.2.2.2).

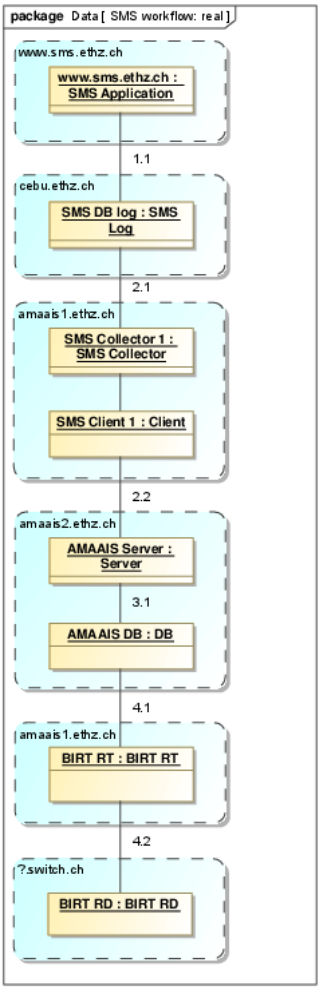


Figure 4: Standard SMS Workflow.

4.2.2.1 Standard SMS workflow

In the standard SMS workflow access to the application is granted through a single web application which generates log entries in the SMS Database log.

SMS events are generated twice per minute while the AMAAIS accounting workflow should be executed in short intervals (*e.g.*, every two minutes) to give the users a quick accounting and billing feedback.

See Table 2 and Figure 4 for the flow and component details.

4.2.2.2 Multi-client SMS workflow

The multi-client mimics a configuration with four web applications and four AMAAIS clients deployed different machines. Steps 1.1, 2.1 and 2.2 in Table 2 would then be performed in parallel on the four machines. See Figure 5 for an overview of the workflow components.

Table 2: Standard SMS Workflow.

Step	Action	Machine	Resources	Data	Rate	Encr.	AA[a]
1.1	SMS sent entry in the log database	www.sms.ethz.ch	DB Log	Source Destination Timestamp SMS length User ID	2/min.	—	User/password IP-filter
2.1	Collector fetches data	amaais1.ethz.ch	DB Log	SMS length Country	8 every 2 min.	—	User/password IP-filter
2.2	Client sends record to server	amaais1.ethz.ch		SMS length Country	8 every 2 min.	SSL	Certificate
3.1	Server stores record	amaais2.ethz.ch	Accounting DB	SMS length Country	8 every 2 min.	SSL	Certificate
4.1	BIRT Report Designer create report and upload to BIRT server	(Admin's machine)	BIRT Report Designer	rpt design file	(on demand)	SSH	SSH credentials
4.2	BIRT runtime component fetches data from the DB	amaais2.ethz.ch	Accounting DB BIRT runtime component	rpt design file report data	(on demand)	SSL	Shibboleth

[a]Authentication and Authorization

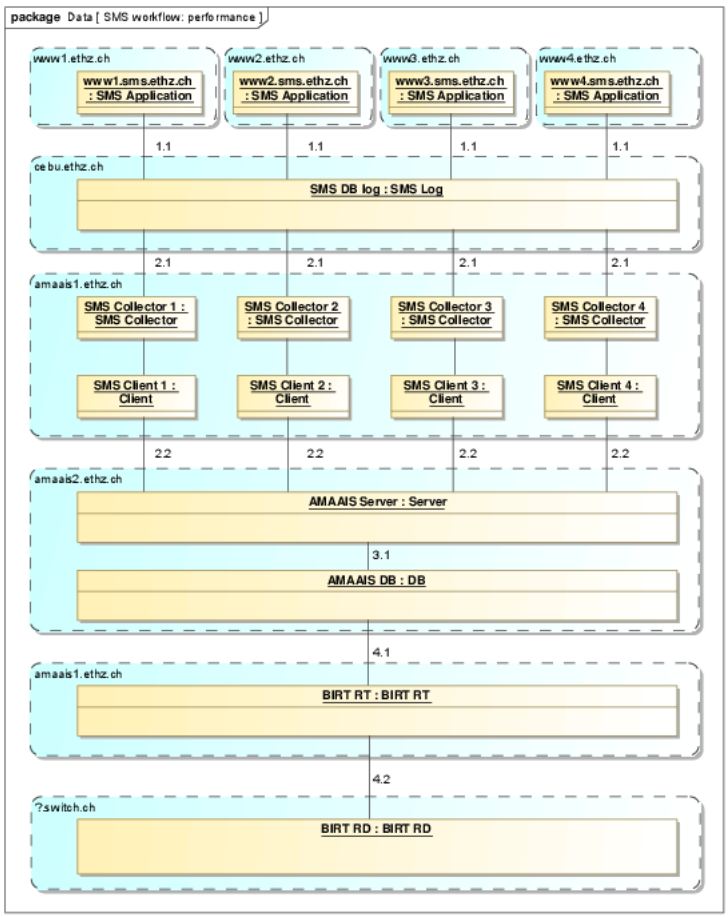


Figure 5: Multi-client SMS Workflow.

4.2.3 SMS Reports

Based on a poll, ETHZ requires the following information in SMS usage reports.

1. Number of SMS users
2. Top-20 users by number of SMS
3. Average number of SMS per user
4. Percentage of SMS sent to foreign countries
5. Average number of SMS sent to foreign countries per user
6. Number of different countries SMS are sent to (by country code)
7. Top-20 countries by number of SMS
8. Top-20 users by amount of money spent on SMS
9. Average amount of money spent on SMS overall

10. Average number of SMS sent per month / year
11. Average number of SMS sent per carrier per month / year[1]
12. Average number of SMS sent to foreign countries per month / year

For each SMS sent over the gateway, information about that event will be recorded in the database. The data about the event has to include the following information (accounting attributes) in order to create the reports mentioned above:

eventTimestamp The date and time of an event

userID A persistent identifier of the user

serviceName The name of the service, the accounting record contains information about (e.g., authentication, authorization, printing, SMS)

characterCount The number of characters the SMS contains

totalNumber The total number of SMS

recipientTelephoneNumber The (mobile) telephone number of the recipient

Additional to the information above, mapping tables will be needed to produce reports that include country information and money: one table to map from country codes to country names and one table for the tariff which maps from carrier to price per SMS.

4.3 VPP

In the next subsections it is presented the workflow definition and its assumptions based on the Printing use case (VPP).

4.3.1 Components and infrastructure

The components and infrastructure that are specific to the VPP use case are listed below:

- There are around 50 web applications to release a print job (i.e., VPP print release stations).
- There are 7 VPP print servers each one with an accounting client. If needed the accounting clients can be installed on other machines and the log files made available (e.g. via NFS or by synchronization)

[1] The gateway infrastructure can't provide the information about the carrier. Therefore as an approximation, the carrier will be derived from the prefix of the destination phone number.

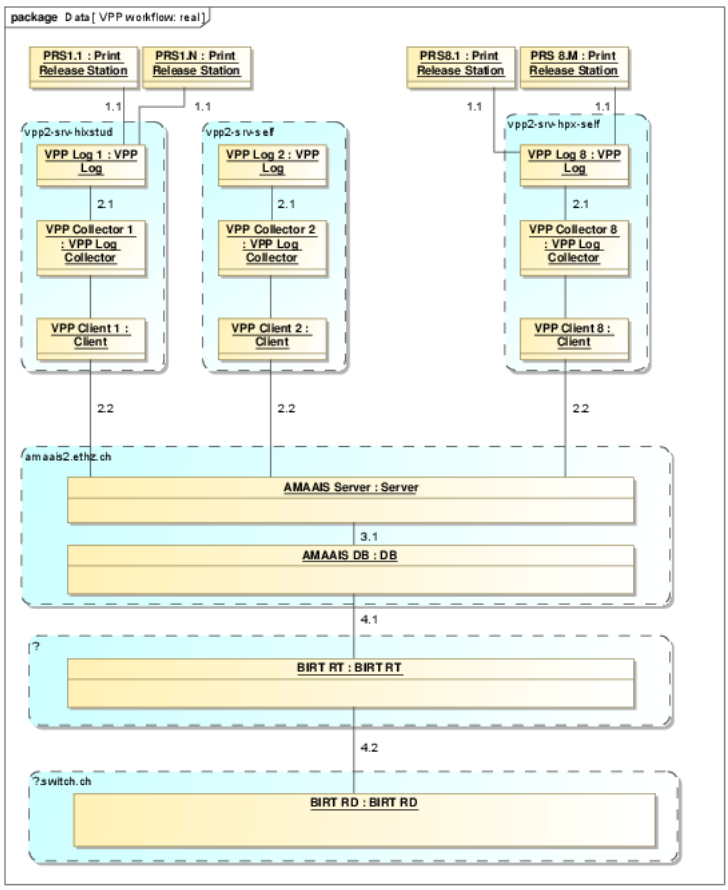


Figure 6: Standard printing workflow.

Machine	Log
vpp2-srv-hixstud.ethz.ch	`/var/log/vpp/VPPYYMM.acc`
vpp2-srv-self.ethz.ch	`/var/log/vpp/VPPYYMM.acc`
vpp2-srv-hg-self.ethz.ch	`/var/log/vpp/VPPYYMM.acc`
vpp2-srv-hcistud.ethz.ch	`/var/log/vpp/VPPYYMM.acc`
vpp2-srv-hpxstud.ethz.ch	`/var/log/vpp/VPPYYMM.acc`
vpp2-srv-hg-old.ethz.ch	`/var/log/vpp/VPPYYMM.acc`
vpp2-srv-hpx-self.ethz.ch	`/var/log/vpp/VPPYYMM.acc`

4.3.2 Workflow

The following workflow is triggered by a timer. Although it could be executed at arbitrary intervals, it is often needed to update the billing records to avoid support requests (*i.e.*, users calls to the support center, complaining about a more up-to-date view of what was used). The current custom implementation executes the accounting and billing process every 10 minutes.

See Table 3 and Figure 6 for a detailed description.

Table 3: Standard VPP workflow.

Step	Action	Machine	Resources	Data	Rate	Encr.	AA[a]
1.1	Job printed entry in the the local log file	VPP print server	`/var/log/vpp/VPPYYMM.acc`	number of pages format size printer station color timestamp User ID	5/min.	—	File permission
2.1	Collector fetches data	amaais1.ethz.ch	`/var/log/vpp/VPPYYMM.acc`	format size printer station color timestamp User ID	8 every 2 min.	—	File permission
2.2	Client sends record to server	amaais1.ethz.ch		format size printer station color timestamp User ID	8 every 2 min.	SSL	Certificate
3.1	Server stores record	amaais2.ethz.ch	Accounting DB	format size printer station color timestamp User ID	8 every 2 min.	SSL	Certificate
4.1	BIRT Report Designer create report and upload to BIRT server	(Admin's machine)	BIRT Report Designer	rpt design file	(on demand)	SSH	SSH credentials
4.2	BIRT runtime component fetches data from the DB	amaais2.ethz.ch	Accounting DB BIRT runtime component	rpt design file report data	(on demand)	SSL	Shibboleth

[a]Authentication and Authorization

5 Test Cases Definition

Driven by the worklows modelled, the key set of test cases is determined in this section according to the use case design dimensions considered (*cf.* Section 5.1). These use cases form the solid basis for the testing during the upcoming implementation and evaluation phase. The use cases enable the project to explore the way components interact with each other and which kind of interaction may break the functionality of the whole system given, *e.g.*, a non-valid input.

5.1 Design Dimensions

The presented test cases cover three test dimensions: functionality, performance, and security. These dimensions were chosen due to the fact that typical accounting systems should guarantee data integrity. Data integrity refers to the trustworthiness of system resources (in this case, accounting records) over their entire life cycle – from the data generated by the Meter component until such data be persisted in the Accounting Data Base. The trustworthiness of a system is measured by these core attributes: completeness, timeliness, accuracy (or correctness), and validity. This documents explores the "completeness", "timeliness", and "accuracy/correctness". The validity will be explored in more specific test cases during the system's implementation phase.

In the functionality test cases, for each case the tester should observe anomalies (as components crashing, or generating a wrong output) in order to evaluate if the system is not working as it should be. Moreover, the tests within this dimension should show if the data collected is fully persisted in the accounting database (completeness).

Related to performance, the tester should observe if the system is accomplishing its functions even with the generation of an abnormal volume of data. The normal volume of data was described as shown in the column "rate" of tables presented in Section 4. Besides the test cases described in the following sections, it should be observed what is the delay (time-wise) to persist Accounting Events. Moreover, the performance test cases also focus to observe peaks and if they affect the system somehow. The performance dimension covers the "timeliness".

Security presents aspects inter-relating functionality and/or performance test cases. Software bugs that affect the functionality (*e.g.*, if a component crashes, like the Accounting Server) can also represent a security threat – and therefore being a subject to be tested. However, these cases of overlapping are generalized here, and should be carefully observed by the tester. Besides that, the security test cases described here focus more in attacks that come from the outside. Therefore, the assumption taken is that there are no infiltrated attackers inside the AMAAIS system's network (*i.e.*, no compromised machines). In summary, the security tests aim to help investigate if the system protects data and maintains functionality as intended, covering the "accuracy/correctness".

It is essential to understand some terms in order to fully understand the test cases. Here, "unknown" means a collected value that the system does not consider normal. A typical example of an unknown value for the attribute "number of pages" would be "123abc", since number of pages inside the system is supposed to be represented by an integer (variable

type mismatching). Another example would be the value “A990” for the attribute page size, since well-known page size values by the system is, for example, “A4”, “A3”, etc.

5.2 SMS

This section presents test cases in relation to the SMS workflow. All steps mentioned here should be related to Table 2. In the Sections 5.2.1, 5.2.2, and 5.2.3, the test cases are presented as items. Each item should be faced as a strong base for a more detailed and technical implementation use case.

It is important to highlight that, even if it is very subjective, the AMAAIS system is considered in a normal state when the collector component collects information from a meter and persists the accounting information (as collect in the input) to the Accounting DB (described in the Section 4.2). Any deviation of such workflow can be considered an anomaly that should be further investigated in the implementation test phase.

5.2.1 Functionality

Below, the high-level functionality test cases to be considered are listed:

- For each SMS event in the SMS DB log (SMS Meter) given a UserID, check if the number of SMS in SMS DB log is equal to the number of SMS in the Accounting DB.
- In the SMS DB Log, generate a set of SMS events with an absurd number of characters. Therefore, check if the SMS events were persisted in the Accounting DB right after.
- In the SMS DB Log, generate a set of SMS events with an unknown country ID.
- In the SMS DB Log, generate a set of SMS events with a wrong (negative number, or incredibly big) time stamp.
- In the SMS DB Log, generate a set of SMS events with no size but with the information of a high number of SMSs.
- In the SMS DB Log, generate a set of SMS events with invalid destination and/or source. It can be an invalid phone number, with *e.g.*, other characters than numbers.
- In the Accounting Protocol, inject a non-valid message (*e.g.*, a malformed SAML message, with wrong XML structure) when exchanging messages from the Accounting Client to the Accounting Server.
- Check if the SMS collector component continues to parse/collect data if the SMS DB log is not reachable for a period of time (*e.g.*, no connection) and, after a while, the SMS DB Log gets reachable again.
- Check if the SMS Collector will resume the action of pushing data to the Accounting Client if the Client becomes unavailable for a period of time and, after a while, becomes available again.

- Observe if the AMAAIS system is not persisting two times the same Accounting Records (*i.e.*, the same SMS event).

5.2.2 Performance

Below, the high-level performance test cases to be considered are listed:

- Generate 100 SMS events per minute in the SMS DB Log to check if the SMS Collector performs well to forwards all events to the Accounting Server. After all, check if all SMSs were persisted in the Accounting DB.
- Check if all data persisted in the Accounting DB until the exact moment of the report request reflects the information displayed in each report.
- Client generating 1000 SMS events per minute to the server. Check if the events are persisted in the Accounting DB.
- Check the inter-component delay when the AMAAIS system is under stress.
- Check the delay of generating the accounting reports.

5.2.3 Security

Below, the high-level security test cases to be considered are listed:

- Evaluate if the SMS DB Log releases SMS data with correct username and password, but with a non-authorized source IP address.
- Evaluate if the SMS DB Log releases SMS data with non-authorized Username and Password, but with an authorized source IP address.
- Check if the Server accepts to push Accounting Records (Table 2, Step 2.2) sent by a Client with a non-authorized certificate.
- Evaluate if BIRT fetches SMS data from the Accounting DB with wrong credentials (username and password).
- Check if the Accounting Client responsible for another service is able to push SMS Accounting Events and be persisted in the Accounting DB.

5.3 VPP

This section presents test cases related to the VPP Workflow (Printing Use Case). All steps mentioned here should be related to Table 3. In the Sections 5.3.1, 5.3.2, and 5.3.3, the test cases are presented as items. Each item should be faced as a strong base for a more detailed and technical implementation use case.

It is important to highlight that, even if it is very subjective, the AMAAIS system is considered in a normal state when the collector component collects information from a meter and persists the accounting information (as collect in the input) to the Accounting DB (described in the Section 4.2.2.1). Any deviation of such workflow can be considered an anomaly that should be further investigated in the implementation test phase.

5.3.1 Functionality

Below, the high-level functionality test cases to be considered are listed:

- For each Job event of a set of Job events in the Accounting DB – for a given UserID –, all the attribute's values (number of pages, format, size, printer, station, color, timestamp, UserID) must be equal to the attribute's values in the VPP log files.
- Generate print events with a high "number of pages" (*e.g.*, more than the integer limit).
- Generate print events with an unknown "format" (*e.g.*, .odt).
- Generate print events with an unknown "size" (*e.g.*, instead of A4 or A3, inject "A9999").
- Generate print events with an unknown "printer" (*e.g.*, instead of a well-formed name like "HP2600", inject some symbols or non-ASCII characters).
- Generate print events with an unknown "station" (*e.g.*, instead of a well-formed name like "Station22", inject some symbols or non-ASCII characters).
- Generate print events with an unknown "color" (*e.g.*, instead of a value like "yes" or "no", inject symbols, non-ASCII characters, or simply "yessss", "noooo", and numbers).
- Generate print events with an unknown "timestamp" (*e.g.*, inject a value bigger than a long variable type, also varying the test case with letters and non-ASCII symbols).
- Generate print events with an unknown "UserID" (*e.g.*, inject a extremely big value – even being a string – and use non-ASCII symbols).
- In the Accounting Protocol, inject a non-valid message (*e.g.*, a malformed SAML message, with wrong XML structure) when exchanging messages from the Accounting Client to the Accounting Server.
- Check if the VPP collector component continues to parse/collect data if the VPP log file is not reachable for a period of time (*e.g.*, wrong file permissions, or non-existent file) and, after a while, the VPP log file gets reachable again.
- Check if the VPP Collector will resume the action of pushing data to the Accounting Client if the Client becomes unavailable for a period of time and, after a while, becomes available again.
- Observe if the AMAAIS system is not persisting two times the same Accounting Records (*i.e.*, the same Job print event).

5.3.2 Performance

Below, the high-level performance test cases to be considered are listed:

- Generate 1000 Job events per minute into the VPP log files to check if, after 10 minutes, the Accounting DB have persisted all generated events.
- Check if all data persisted on the Accounting DB until the exact moment of the report request reflects the information displayed in each report.
- Check if the Accounting Client can support to push more than 1000 events per minute from the VPP Collector.
- Check the inter-component delay when the AMAAIS system is under stress.
- Check the delay of generating the accounting reports.

5.3.3 Security

Below, the high-level security test cases to be considered are listed:

- Check if the collector can fetch data (read) from the VPP log files (Table 3, Step 2.1) using a user out of the group which has permissions to read the VPP log files.
- Check if the Server accepts to push Accounting Records (Step 2.2) sent by a Client with a non-authorized certificate.
- Evaluate if the BIRT fetches Printing data from the Accounting DB with wrong credentials (username and password).
- Check if the Accounting Client responsible for another service is able to push Printing Accounting Events and be persisted in the Accounting DB.

5.4 Test Cases Considerations

One of most common problems foreseen at this stage of the system is related to the variety of different data provided by Meter components. Meters can make information available to Collectors using unexpected characters and, therefore, possibly breaking the Collector. *E.g.*, instead of providing a telephone number (in the SMS use case), a user can use several characters that can be valid for the GUI (Graphical User Interface) but invalid (not expected) for the SMS Collector. This can be critical and requires a careful Collector development aligned to all possible inputs from the Meter.

Another common problem foreseen is the unavailability of components. *E.g.*, Accounting Clients should identify that an Accounting Server is unavailable and therefore not discard accounting records. The same situation may occur with Collectors and Accounting Clients. During the AMAAIS developing phase buffer sizes and databases should be properly designed and evaluated to support data that cannot be sent in a given moment.

What could be noted when compiling the test cases is that performance and security are aspects that involve a lot of factors. The performance of the whole AMAAIS system depends on how many records will be generated, the network capacity where the system is deployed, and the processing power of the machines that are handling the accounting. Together with the workflow definition for use cases, it is important to analyse what is the delay tolerance for each scenario. The delay value for each use case will define if the system is performing well for a given scenario. *E.g.*, for the printing use case, is it acceptable to have a delay of 5 minutes from the moment a user printed something to the moment the accounted information got persisted to the accounting database?

In the security test cases, it depends where the AMAAIS system is deployed and if the environment is considered “secure”. *E.g.*, if it is assumed that the AMAAIS system is deployed in a secure network, attacks like man-in-the-middle or spoofing should not be taken into consideration. However, since AMAAIS provides functionality to exchange accounting events between Accounting Servers from different domains, the protocol should be tested against information forgery at both ends (client and server). The use of a certificate-based solution is an option to overcome the forgery issue along the network path.

6 Common Accounting Model and Use Case Extensions

This section aims to present the Common Accounting Model (Section 6.2) and its Use Case Extensions (Section 6.3) that will be used in the AMAAIS system. Moreover, this section aims to highlight key changes (Section 6.1) – with the main reasons on why the changes were made – from the model presented on AMAAIS Deliverable 2 [4] to the Common Accounting Model presented in this deliverable. As mentioned before, the changes made focuses on a refactoring process to align the AMAAIS project to Shibboleth [5]. For the sake of simplicity, the terminology "D2" will be used throughout this document to describe the AMAAIS Deliverable 2 [4], and "D3" to describe this document, the AMAAIS Deliverable 3.

In the following sections a brief description of every attribute, in every class under the `commonModel` package is given, Figure 7 presents the full package structure of the AMAAIS project. The two `commonModel` sub packages `commonAcctModel` and `useCasesExtensions` are described in detail in Section 6.2 and 6.3.

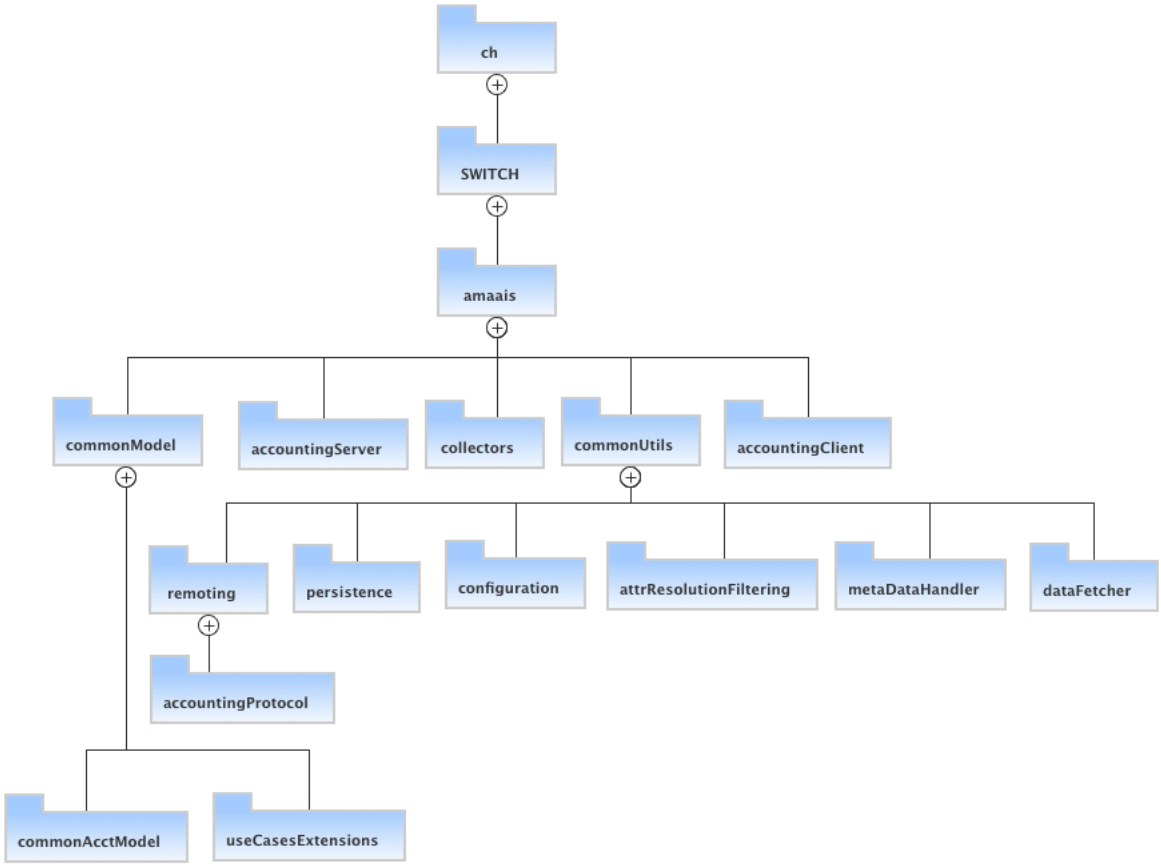


Figure 7: The package structure of the AMAAIS project.

It is extremely important to mention that the Common Accounting Model package (`commonAcctModel`) is intended, during the next implementation phase to come, to be

packed and distributed as a JAR[2] file. The Common Accounting Model package (`commonAcctModel`) JAR file is intended to be the core component of the AMAAIS implementation since all services within a federation should extend it to represent accounting information in a full compatible manner to other system components. The Use Cases Extensions package (`useCasesExtensions`) is intended to serve as an example on how the `commonAcctModel` package classes are extended, forming the service-dependent part of the model.

6.1 Considerations on the Changes from AMAAIS Deliverable 2 to Deliverable 3

During AMAAIS phase 2, the project relied on the Common Accounting Model described in the D2 [4], Figure 4. However, analyzing the previous Common Accounting Model, the AMAAIS members encountered some drawbacks during the implementation phase and therefore decided to pass through a refactoring phase to (1) ease its use, (2) align the AMAAIS model to the Shibboleth implementation, and (3) diminish the abstraction levels to avoid database performance issues.

Below, there are some punctual considerations and reasons on why the Common Accounting Model was modified from the D2 to the D3:

- The model documented in D2 was relying on a totally generic model, with many layers of abstraction. *E.g.*, it was being assumed that new attribute types would be able to be created, groups of attributes could be formed, etc. However, it was agreed that, *e.g.*, new attribute types or how the attributes could be grouped, should be coupled to the service itself. *I.e.*, the printing service should be responsible on how the attributes are described or organized.

- Still touching the many layers of abstraction within the model developed in D2, it was clear that one single concept to represent accounting data was assumed: based on sessions that contains accounting records with its attributes. However, some services like printing or SMS do not have an intuitive notion of accounting sessions as the example of a video stream service has (further exemplified on Section 6.2). This analysis led the team to think on a Common Accounting Model with co-existing concepts: the notion of events – which most of the services rely on –, and also with the notion of sessions. Depending of the nature of the service to be accounted for, one of these two concepts should be better explored.

- In preliminary tests made using the Common Accounting Model presented in D2, it was noted that database performance issues could be significant in future (as the data grows). One of the main reasons for that is the quantity of SQL JOINs that would be necessary to recover data from the accounting database, assuming that one Java Class would represent one table. Therefore, the Common Accounting Model presented in D3 was intended to have more generalization through the

[2]JAR (for Java Archive) is an archive file format typically used to aggregate many Java class files and associated metadata and resources (text, images and so on) into one file to distribute application software or libraries on the Java platform.

different levels, and therefore tending to materialize less tables in the accounting database (due to less instantiated Java classes).

- The use of a global and single dictionary of attributes for the whole Shibboleth federation was one assumption made in the D2 model developed. The previous model would require a central point (that would take care of the global dictionary of attributes) that every time a new service gets released within the federation, all SPs and IdPs would have to update their system in order to cope with the new changes (mainly if the exchanging of attribute records would happen between two domains). However, this assumption was faced as unrealistic after better analyzing the complexity of such system. Therefore for the model developed in D3, agreements between domains allowing the exchange of accounting records were assumed. It means that, each time a service get released by any domain (*e.g.*, UZH), this domain should extend the Common Accounting Model package (`commonAcctModel`) with the service to be released (placed in `useCasesExtensions`), and then distribute the new JAR to the other domains that are supposed to exchange accounting information related to it. Since the release of new services do not happen very frequently within a federation, it justifies to adopt something simple, punctual, but functional (D3 model), and not something very generic with a high management effort (D2 model).
- If new services would be added in the D2 model, two places would be affected: the global dictionary of attributes and possible the new types (if required) to the model.
- Since cross-event SQL queries are assumed to be not very common (*e.g.*, all users that used service X **and** Y for more than X hours), the D3 model was designed in consideration of the database design phase that will happen in future: one table per event/service. The AMAAIS members agreed that intra-event SQL queries happen more frequently than cross-event ones.

6.2 Common Accounting Model

In more detail, as shown in Figure 8 the `commonAcctModel` package contains all the service-independent classes used by the AMAAIS Accounting model.

The `AcctEvent` class represents an accounting event that contains the following attributes:

- `eventId` is of type `String` and identifies an event in a unique format for a given service. The format of the eventID attribute is the UUID[3].
- `eventInstant` is of type `Long` and identifies the instant time that the AcctEvent was originally created by the resource itself (*e.g.*, when an SMS was sent). The format of the eventInstant attribute is the POSIX time[4].

[3]UUID stands for Universal Unique Identifier, and in this case, it will be used the implementation from the package "java.util.UUID".

[4]POSIX time is a representation for describing points in time, defined as the number of seconds elapsed since midnight Coordinated Universal Time (UTC) of January 1, 1970, not counting leap seconds.

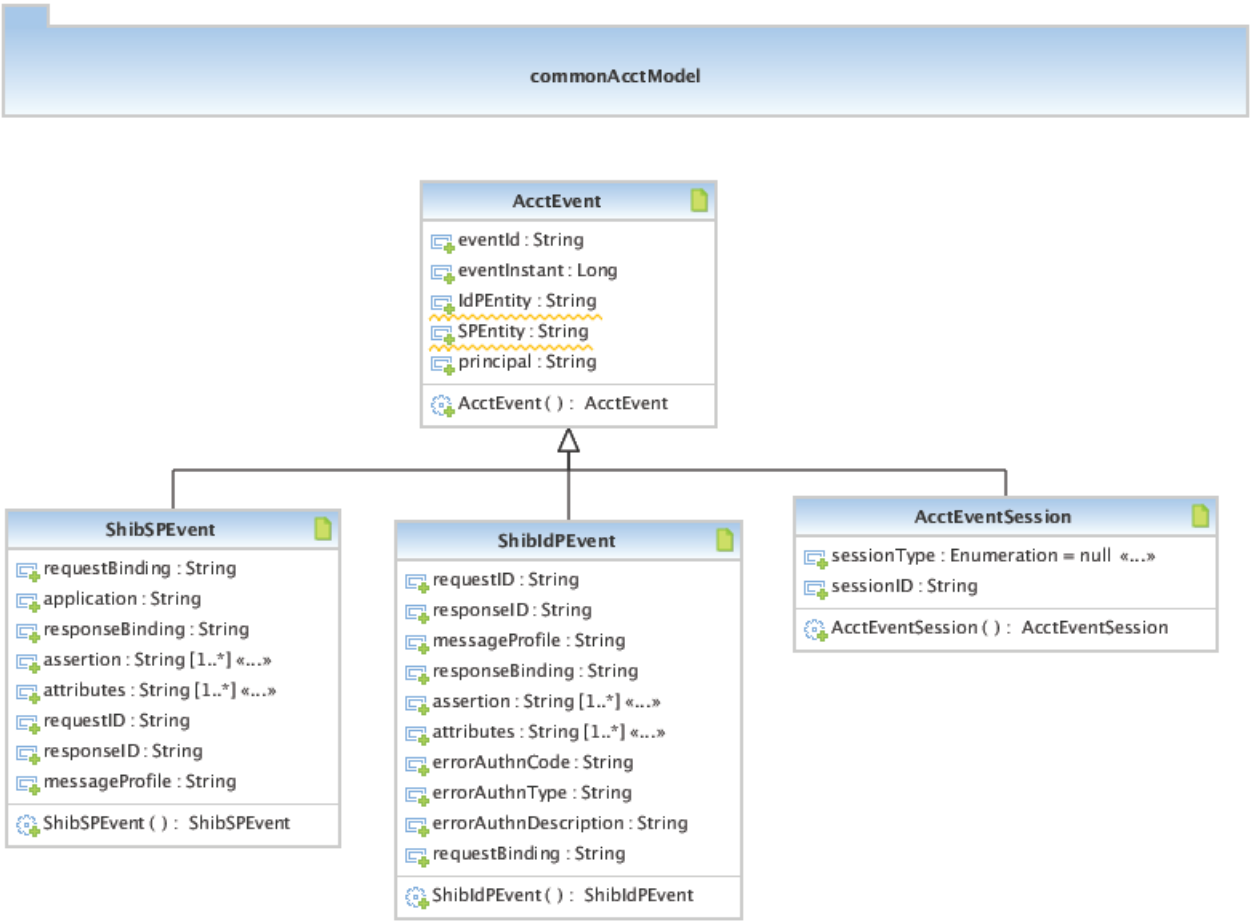


Figure 8: The contents of the package `commonAcctModel` under the package `commonModel`.

- `IdPEntity` is of type `String` and represents a global identification (inside one federation), which identifies the service which a certain resource relies on. *I.e.*, should be expressed the Identity Provider URI of who used the service.
- `SPEntity` is of type `String` and represents a global identification (inside one federation), which identifies the resource that generated the event, and/or of which resource the AcctEvent belongs to. *I.e.*, should be expressed the Service Provider URI associated to the AcctEvent.
- `principal` is of type `String` and represents an identification, that can or not be globally unique, representing who used the resource. *E.g.*, opaqueID from Shibboleth.

Some observations about the design of the `AcctEvent` base class:

- The used nomenclature of the `AcctEvent` is meant to be generic (*e.g.*, to be used by other Authentication and Authorization systems, such as RADIUS or Diameter) but also fully compatible with Shibboleth IdP and SP events – which AMAAIS focuses on.

- The AMAAIS team observed that these attributes are common to other Authentication and Authorization systems, and should be present to any kind of accounting event generated by services that aims to be accounted using AMAAIS.

The `AcctEvent` should be extended in case to incorporate service-specific event classes. Section 6.3 shows two use cases where the `AcctEvent` class were extended.

The `AcctEventSession` class represents an accounting event session and extends the `AcctEvent` class. This class is also located under the *commonModel.CommonAcctModel* package and contains the following attributes:

- `sessionType` is of type `Enumeration`, which is used instead the primitive type `enum`. The three possible values of sessionType are the following strings: `START_SESSION`, `INTERMEDIATE_SESSION` and `END_SESSION`. The attribute `sessionType` represents what kind of session action an accounting event is performing. *E.g.*, a video streaming service can start a given video session, therefore generating a session event `START_SESSION`. During the video stream, the service can generate one or more `INTERMEDIATE_SESSION` session events, informing bandwidth consumed, quality changes during the transmission, etc. In the end, an `END_SESSION` session event is generated to indicate that the service consumption ended.
- `sessionID` is of type `String` and globally identifies a session. One accounting session can be composed by one or more accounting event session's actions. *E.g.*, a video stream session is composed by accounting event session's actions of different session types (sessionType).

It is important to highlight here that, like the `AcctEvent` class, the `AcctEventSession` is an abstract class. The `AcctEventSession` was designed to incorporate the notion of sessions for certain events. For some services, it is desired to created one level of abstraction on top of "events", expressing a period of time devoted to a specific activity. Therefore, sessions were introduced based on the concept presented in the terminology section in RFC 2866 [6] and AMAAIS Deliverable 2 [4]. This deliverable does not present any concrete example to extend the `AcctEventSession` class, however aforementioned services like video streaming or network usage are use cases that can explore the concept of session events.

The next class under the `commonModel.commonAcctModel` package is `ShibIdPEvent` class, which also extends the `AcctEvent` class. The `ShibIdPEvent` represents a Shibboleth IdP event, therefore having the following attributes:

- `requestID` is of type `String` and identifies the global unique identifier of an exchanged request message for the IdP.
- `responseID` is of type `String` and identifies the global unique identifier of an exchanged response message from the IdP.
- `messageProfile` is of type `String` represents which SAML Profile [7] was used in the accounted message event

- `requestBinding` is of type `String` represents the SAML binding used in the accounted request message to the IdP. The existent bindings are described in SAML Bindings Document [8].
- `responseBinding` is of type `String` represents the SAML binding used in the accounted IdP response message. The existent bindings are described in SAML Bindings Document [8].
- `assertion` is of type `String[]` and represents the `<Assertion>` element of SAML [9] messages. Therefore, for each position of this String array, one `<Assertion>` sub element is represented (*e.g.*, Version, Issuer, AuthnStatement, AttributeStatement, etc).
- `attributes` is of type `String[]` and represents the `<Attribute>` element of SAML [9] messages. Therefore, for each position of this String array, the `<Attribute>` would be added (as a String).
- `errorAuthnCode` is of type `String` and represents, if existent, the authentication error code of a given authentication request.
- `errorAuthnType` is of type `String` and represents, if existent, the authentication error type related to the `errorAuthnCode`.
- `errorAuthnDescription` is of type `String` and represents, if existent, the authentication error description related to the `errorAuthnCode` and `errorAuthnType`.

The last class which also extends the `AcctEvent` class under the `commonModel.commonAcctModel` package is `ShibSPEvent`. It represents a Shibboleth SP event and contains the following attributes:

- `requestID` is of type `String` and identifies the global unique identifier of an exchanged request message for the SP.
- `responseID` is of type `String` and identifies the global unique identifier of an exchanged response message from the SP.
- `messageProfile` is of type `String` represents which SAML Profile [7] was used in the accounted message event.
- `requestBinding` is of type `String` represents the SAML binding used in the accounted request message to the SP. The existent bindings are described in SAML Bindings Document [8].
- `responseBinding` is of type `String` represents the SAML binding used in the accounted IdP response message. The existent bindings are described in SAML Bindings Document [8].
- `assertion` is of type `String[]` and represents the `<Assertion>` element of SAML [9] messages. Therefore, for each position of this String array, one `<Assertion>` sub element is represented (*e.g.*, Version, Issuer, AuthnStatement, AttributeStatement, etc).

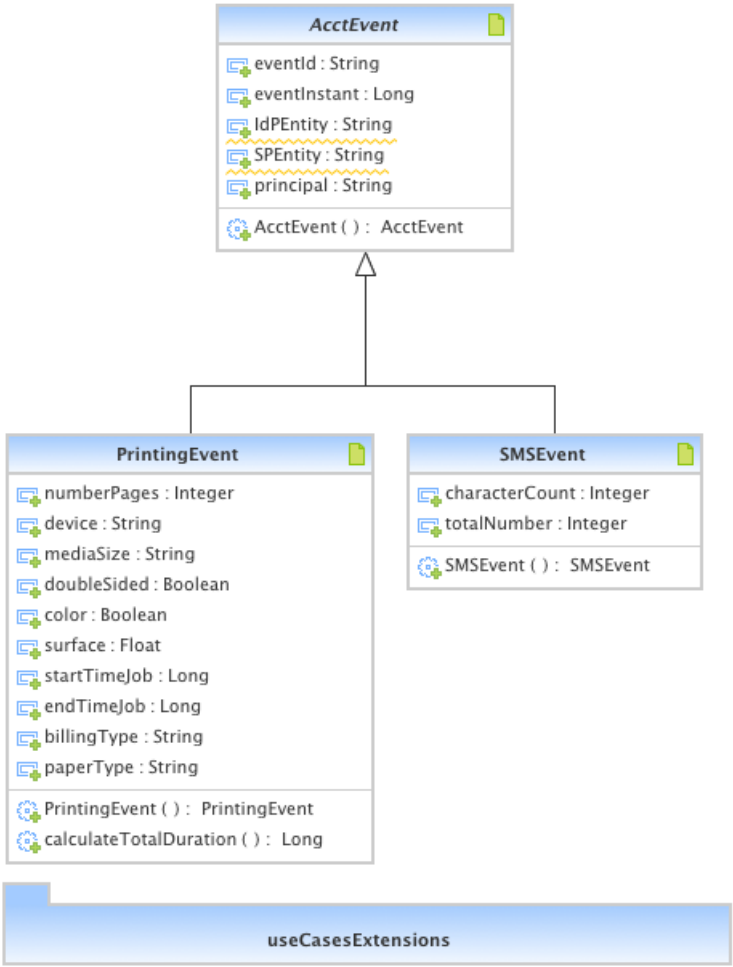


Figure 9: The contents of the package `useCasesExtensions` under the package `commonModel`.

- `attributes` is of type `String[]` and represents the `<Attribute>` element of SAML [9] messages. Therefore, for each position of this `String` array, the `<Attribute>` would be added (as a `String`).
- `application` is of type `String` and identifies the service (expressed by an URI) that the event relates to.

6.3 Use Cases Extensions

The `useCasesExtensions` package Figure 9 contains the service-dependent accounting event classes. Those classes should be developed by the Service Provider. The classes of the two main case studies services, printing and SMS provided at ETHZ, follows. Note that both `PrintingEvent` as well as `SMSEvent` class are extending the `AcctEvent` class of

the `commonModel.commonAcctModel` package. Thus, it is important to mention that every class located under the `commonModel.useCasesExtensions` package, as they are service dependent classes, should extend the `AcctEvent` class.

The `PrintingEvent` class is located under the `commonModel.useCasesExtensions` package, it represents a VPP printing event and contains the following attributes:

- `numberPages` is of type `Integer` and identifies the number of pages printed.
- `device` is of type `String` and identifies the printer that the job was printed.
- `mediaSize` is of type `String` and identifies the size of the media (*i.e.*, the paper of a print job).
- `doubleSided` is of type `Boolean` and represents if the print job was printed in both front and back side of the media (paper).
- `color` is of type `Boolean` and identifies if the job was printed with the color option turned on.
- `surface` is of type `Float` and represents, in square meters, how much of the media was used in the job printed.
- `startTimeJob` is of type `Long` and represents the time in *POSIX* that a job started.
- `endTimeJob` is of type `Long` and represents the time in *POSIX* that a job finished.
- `billingType` is of type `String` and represents if the job to be printed is intended to be for business or private purposes.
- `paperType` is of type `String` and identifies what the type of media was used (*e.g.*, thin paper, normal paper).

The `SMSEvent` class represents an SMS event. Being a service dependent class the location of `SMSEvent` class is also under package `commonModel.useCasesExtensions`. `SMSEvent` has the following attributes:

- *characterCount* is of type `Integer` and represents the number of characters in the SMS.
- *totalNumber* is of type `Integer` and identifies the total number of SMS.

7 API Definition

This section presents the Application Programming Interface (API) for each implementation component expected in the AMAAIS system.

In the AMAAIS project, several implementation packages were created in order to divide responsibilities and deliver a modularized system. Figure 7 depicts the organizational view of the AMAAIS implementation packages, as mentioned in Section 6.

7.1 API

Subsequently, the current version of the textual description of each Java class inside each AMAAIS implementation package is presented. For each class from this diagram, it is presented below the class description, the method list, methods description, method's return description, and the method's parameters description.

7.1.1 ch.SWITCH.amaais.accountingClient

- **Class: Accounting.java**
- **Description:** This is the interface for the Accounting Client.
- **Methods:**
  - `public Boolean init();`
    Description: Initialize the Accounting object
    Return: True if initialization was successful and False if not
  - `public Boolean start();`
    Description: Starts the Accounting component
    Return: True if Accounting component started successfully and False if not
  - `public Boolean stop();`
    Description: Stops the Accounting component
    Return: True if Accounting component is stopped successfully and False if not

- **Class: AccountingClient.java**
- **Description:** The AccountingClient component
- **Methods:**
  - `public Boolean pushAcctEvent(AcctEvent event);`
    Description: Pushes an AcctEvent to the Accounting Client component. Observe that this method just stores the AcctEvent locally.

Parameter: *event* – The AcctEvent to be pushed
Return: True if the AcctEvent pushed successfully (locally). False if any error occurred.

- `public Boolean pushAcctEvent(List events);`
  Description: Pushes an AcctEvent List to the Accounting Client component. Observe that this method just stores the AcctEvent locally.
  Parameter: *events* – The AcctEvent List to be pushed
  Return: True if the AcctEvent List is pushed successfully (locally). False if any error occurred.

7.1.2 ch.SWITCH.amaais.accountingServer

- **Class: AccountingServer.java**
- **Description:** The AccountingServer component

7.1.3 ch.SWITCH.amaais.collectors

- **Class: PrintingCollector.java**
- **Description:** This is the Collector for the VPP Printing Use Case (ETHZ) It parses the data received from the VPP Printing meter and push AcctEvents to the Accounting Client.

- **Class: SMSCollector.java**
- **Description:** This is the Collector for the SMS Use Case. It parses the data received from the SMS meter and push AcctEvents to the Accounting Client.

- **Class: IdPCollector.java**
- **Description:** This is the Collector for the Shibboleth IdP. It parses the data received from the IdP meter and push AcctEvents to the Accounting Client.

- **Class: SPCollector.java**
- **Description:** This is the Collector for the Shibboleth SP. It parses the data received from the SP meter and push AcctEvents to the Accounting Client.

- **Interface: Collector.java**
- **Description:** This is the interface for Collector components. Developers that implement new Collectors should strictly implement this interface.

- **Methods:**
  - `public Boolean init();`
    Description: Initializes the Collector component
    Return: True if the Collector initialized successfully and False if not
  - `public Boolean stop();`
    Description: Stop the Collector component
    Return: True if the Collector stopped successfully and False if not
  - `public Boolean start();`
    Description: Start the Collector component
    Return: True if the Collector started successfully and False if not

7.1.4 ch.SWITCH.amaais.commonModel.commonAcctModel

- **Class: AcctEventSession.java**
- **Description:** The object that represents an Accounting Event Session
- **Methods:**
  - `public String getSessionID();`
    Description: This is a getter for the attribute sessionID
    Return: the sessionID
  - `public void setSessionID(String sessionID);`
    Description: This is a setter for the attribute sessionID
    Parameter: *sessionID* – the sessionID to set
  - `public Enumeration getSessionType();`
    Description: This is a getter for the attribute sessionType
    Return: the sessionType
  - `public void setSessionType(Enumeration sessionType);`
    Description: This is a setter for the attribute sessionType
    Parameter: *sessionType* – the sessionType to set

- **Class: ShibIdPEvent.java**
- **Description:** The object that represents a Shibboleth IdP Event
- **Methods:**

- `public String getRequestID();`
  Description: This is a getter for the attribute requestID
  Return: the requestID

- `public void setRequestID(String requestID);`
  Description: This is a setter for the attribute requestID
  Parameter: *requestID* – the requestID to set

- `public String getResponseID();`
  Description: This is a getter for the attribute responseID
  Return: the responseID

- `public void setResponseID(String responseID);`
  Description: This is a setter for the attribute responseID
  Parameter: *responseID* – the responseID to set

- `public String getMessageProfile();`
  Description: This is a getter for the attribute messageProfile
  Return: the messageProfile

- `public void setMessageProfile(String messageProfile);`
  Description: This is a setter for the attribute messageProfile
  Parameter: *messageProfile* – the messageProfile to set

- `public String getResponseBinding();`
  Description: This is a getter for the attribute responseBinding
  Return: the responseBinding

- `public void setResponseBinding(String responseBinding);`
  Description: This is a setter for the attribute responseBinding
  Parameter: *responseBinding* – the responseBinding to set

- `public String getAssertion();`
  Description: This is a getter for the attribute assertion
  Return: the assertion

- `public void setAssertion(String assertion);`
  Description: This is a setter for the attribute assertion
  Parameter: *assertion* – the assertion to set

- `public String getAttributes();`
  Description: This is a getter for the attribute attributes
  Return: the attributes

- `public void setAttributes(String attributes);`
  Description: This is a setter for the attribute attributes
  Parameter: *attributes* – the attributes to set

- `public String getErrorAuthnCode();`
  Description: This is a getter for the attribute errorAuthnCode
  Return: the errorAuthnCode

- `public void setErrorAuthnCode(String errorAuthnCode);`
  Description: This is a setter for the attribute errorAuthnCode
  Parameter: *errorAuthnCode* – the errorAuthnCode to set

- `public String getErrorAuthnType();`
  Description: This is a getter for the attribute errorAuthnType
  Return: the errorAuthnType

- `public void setErrorAuthnType(String errorAuthnType);`
  Description: This is a setter for the attribute errorAuthnType
  Parameter: *errorAuthnType* – the errorAuthnType to set

- `public String getErrorAuthnDescription();`
  Description: This is a getter for the attribute errorAuthnDescription
  Return: the errorAuthnDescription

- `public void setErrorAuthnDescription(String errorAuthnDescription);`
  Description: This is a setter for the attribute errorAuthnDescription
  Parameter: *errorAuthnDescription* – the errorAuthnDescription to set

- `public String getRequestBinding();`
  Description: This is a getter for the attribute requestBinding
  Return: the requestBinding

- `public void setRequestBinding(String requestBinding);`
  Description: This is a setter for the attribute requestBinding
  Parameter: *requestBinding* – the requestBinding to set

- `public AcctEvent copy();`
  Description:

- **Class: ShibSPEvent.java**
- **Description:** The object that represents a Shibboleth SP Event
- **Methods:**

- `public String getRequestBinding();`
  Description: This is a getter for the attribute requestBinding
  Return: the requestBinding

- `public void setRequestBinding(String requestBinding);`
  Description: This is a setter for the attribute requestBinding
  Parameter: *requestBinding* – the requestBinding to set

- `public String getApplication();`
  Description: This is a getter for the attribute application
  Return: the application

- `public void setApplication(String application);`
  Description: This is a setter for the attribute application
  Parameter: *application* – the application to set

- `public String getResponseBinding();`
  Description: This is a getter for the attribute responseBinding
  Return: the responseBinding

- `public void setResponseBinding(String responseBinding);`
  Description: This is a setter for the attribute responseBinding
  Parameter: *responseBinding* – the responseBinding to set

- `public String getAssertion();`
  Description: This is a getter for the attribute assertion
  Return: the assertion

- `public void setAssertion(String assertion);`
  Description: This is a setter for the attribute assertion
  Parameter: *assertion* – the assertion to set

- `public String getAttributes();`
  Description: This is a getter for the attribute attributes
  Return: the attributes

- `public void setAttributes(String attributes);`
  Description: This is a setter for the attribute attributes
  Parameter: *attributes* – the attributes to set

- `public String getRequestID();`
  Description: This is a getter for the attribute requestID
  Return: the requestID

- `public void setRequestID(String requestID);`
  Description: This is a setter for the attribute requestID
  Parameter: *requestID* – the requestID to set

- `public String getResponseID();`
  Description: This is a getter for the attribute responseID
  Return: the responseID

- `public void setResponseID(String responseID);`
  Description: This is a setter for the attribute responseID
  Parameter: *responseID* – the responseID to set

- `public String getMessageProfile();`
  Description: This is a getter for the attribute messageProfile
  Return: the messageProfile

- `public void setMessageProfile(String messageProfile);`
  Description: This is a setter for the attribute messageProfile
  Parameter: *messageProfile* – the messageProfile to set

- `public AcctEvent copy();`
  Description:

- **Abstract class: AcctEvent.java**
- **Description:** The object that represents an Accounting Event.
- **Methods:**
  - `public String getEventId();`
    Description: This is a getter for the attribute eventId
    Return: the eventId

  - `public void setEventId(String eventId);`
    Description: This is a setter for the attribute eventId
    Parameter: *eventId* – the eventId to set

  - `public Long getEventInstant();`
    Description: This is a getter for the attribute eventInstant
    Return: the eventInstant

  - `public void setEventInstant(Long eventInstant);`
    Description: This is a setter for the attribute eventInstant
    Parameter: *eventInstant* – the eventInstant to set

- `public String getIdPEntity();`
  Description: This is a getter for the attribute IdPEntity
  Return: the idPEntity

- `public void setIdPEntity(String idPEntity);`
  Description: This is a setter for the attribute IdPEntity
  Parameter: *idPEntity* – the idPEntity to set

- `public String getSPEntity();`
  Description: This is a getter for the attribute sPEntity
  Return: the sPEntity

- `public void setSPEntity(String sPEntity);`
  Description: This is a setter for the attribute sPEntity
  Parameter: *sPEntity* – the sPEntity to set

- `public String getPrincipal();`
  Description: This is a getter for the attribute principal
  Return: the principal

- `public void setPrincipal(String principal);`
  Description: This is a setter for the attribute principal
  Parameter: *principal* – the principal to set

- `public abstract AcctEvent copy();`
  Description: Method to copy an AcctEvent class
  Return: The AcctEvent that is a copy from the original class

7.1.5 ch.SWITCH.amaais.commonModel.useCasesExtensions

- **Class: PrintingEvent.java**
- **Description:** The object that represents a VPP Printing Event
- **Methods:**
  - `public Integer getNumberPages();`
    Description: This is a getter for the attribute numberPages
    Return: the numberPages

  - `public void setNumberPages(Integer numberPages);`
    Description: This is a setter for the attribute numberPages
    Parameter: *numberPages* – the numberPages to set

- `public String getDevice();`
  Description: This is a getter for the attribute device
  Return: the device

- `public void setDevice(String device);`
  Description: This is a setter for the attribute device
  Parameter: *device* – the device to set

- `public String getMediaSize();`
  Description: This is a getter for the attribute mediaSize
  Return: the mediaSize

- `public void setMediaSize(String mediaSize);`
  Description: This is a setter for the attribute mediaSize
  Parameter: *mediaSize* – the mediaSize to set

- `public Boolean getDoubleSided();`
  Description: This is a getter for the attribute doubleSided
  Return: the doubleSided

- `public void setDoubleSided(Boolean doubleSided);`
  Description: This is a setter for the attribute doubleSided
  Parameter: *doubleSided* – the doubleSided to set

- `public Boolean getColor();`
  Description: This is a getter for the attribute color
  Return: the color

- `public void setColor(Boolean color);`
  Description: This is a setter for the attribute color
  Parameter: *color* – the color to set

- `public Float getSurface();`
  Description: This is a getter for the attribute surface
  Return: the surface

- `public void setSurface(Float surface);`
  Description: This is a setter for the attribute surface
  Parameter: *surface* – the surface to set

- `public Long getStartTimeJob();`
  Description: This is a getter for the attribute startTimeJob
  Return: the time a job started in POSIX format (seconds)

- `public void setStartTimeJob(Long startTimeJob);`
  Description: This is a setter for the attribute startTimeJob
  Parameter: *startTimeJob* – the time a job started in POSIX format (seconds)

- `public Long getEndTimeJob();`
  Description: This is a getter for the attribute endTimeJob
  Return: the time a job ended in POSIX format (seconds)

- `public void setEndTimeJob(Long endTimeJob);`
  Description: This is a setter for the attribute endTimeJob
  Parameter: *endTimeJob* – the time a job ended in POSIX format (seconds)

- `public String getBillingType();`
  Description: This is a getter for the attribute billingType
  Return: the billingType

- `public void setBillingType(String billingType);`
  Description: This is a setter for the attribute billingType
  Parameter: *billingType* – the billingType to set

- `public String getPaperType();`
  Description: This is a getter for the attribute paperType
  Return: the paperType

- `public void setPaperType(String paperType);`
  Description: This is a setter for the attribute paperType
  Parameter: *paperType* – the paperType to set

- `public Long calculateTotalDuration();`
  Description: This method calculates the total duration, in seconds, that the device took to print the job
  Return: The total duration, in seconds, that the device took to print the job

- `public AcctEvent copy();`
  Description:

- **Class: SMSEvent.java**
- **Description:** The object that represents an SMS Event
- **Methods:**
  - `public Integer getCharacterCount();`
    Description: This is a getter for the attribute characterCount

Return: the characterCount

- `public void setCharacterCount(Integer characterCount);`
  Description: This is a setter for the attribute characterCount
  Parameter: *characterCount* – the characterCount to set

- `public Integer getTotalNumber();`
  Description: This is a getter for the attribute totalNumber
  Return: the totalNumber

- `public void setTotalNumber(Integer totalNumber);`
  Description: This is a setter for the attribute totalNumber
  Parameter: *totalNumber* – the totalNumber to set

- `public AcctEvent copy();`
  Description:

7.1.6 ch.SWITCH.amaais.commonUtils.attrResolutionFiltering

- **Class: AcctAttributeResolution.java**
- **Description:**
- **Class: AcctAttributeFiltering.java**
- **Description:**

7.1.7 ch.SWITCH.amaais.commonUtils.configuration

- **Class: Configuration.java**
- **Description:** Provides an interface to read configuration parameters in .conf files
- **Methods:**
  - `public static String getValue(String key);`
    Description: Method to get a configuration parameter value
    Parameter: *key* – The parameter
    Return: The value
  - `public static Boolean loadConfigFile(String configFile);`
    Description: Load the configuration file
    Parameter: *configFile* – The specified configuration file
    Return: True if the configuration file was loaded

7.1.8 ch.SWITCH.amaais.commonUtils.dataFetcher

- **Interface: DataFetcher.java**
- **Description:** The DataFetcher is an interface to concrete implementation classes that has the knowledge on how to fetch data from a Meter. E.g., reading data from log files with rotation enabled, reading data from proprietary data bases (with specific drivers), etc.

- **Class: LogFileFetcher.java**
- **Description:** Fetches data from a log file

- **Class: JDBCDataFetcher.java**
- **Description:** Read the data from a DataBase

7.1.9 ch.SWITCH.amaais.commonUtils.metaDataHandler

- **Class: AcctMetadataProvider.java**
- **Description:** This class extends the AbstractMetadataProvider from OpenSAML library. This class intends to get the accounting metadata making available to all other components that wants to read any specific information from it.

- **Class: AcctMetadataFilter.java**
- **Description:** This class implements the MetadataFilter from OpenSAML library. This class intends to filter the accounting metadata to grab just relevant information for the components that request a specific part of the metadata.

7.1.10 ch.SWITCH.amaais.commonUtils.persistence

- **Class: AcctEventDAO.java**
- **Description:** The AcctEventDAO
- **Methods:**
  - `public Boolean storeAcctEvent(AcctEvent event);`
    Description: Stores an AcctEvent
    Parameter: *event* – The AcctEvent to be stored
    Return: True if the AcctEvent stored successfully and False if not

- `public AcctEvent getAcctEvent(String acctEvent);`
  Description: Gets the AcctEvent
  Parameter: *acctEvent* – The AcctEvent String
  Return: The AcctEvent

- `public List getAcctEvents(Date from, Date to);`
  Description: Gets a List of AcctEvents
  Parameter: *from* – The initial Date to collect the AcctEvents
  Parameter: *to* – The end Date to collect the AcctEvents
  Return: The collected AcctEvents

7.1.11 ch.SWITCH.amaais.commonUtils.remoting

- **Class: Remoting.java**
- **Description:** The Remoting component takes care of the networking. It sends data between components through the network.
- **Methods:**
  - `public Integer sendAcctEvent(AcctEvent event, String destination);`
    Description: Sends an AcctEvent to a destination
    Parameter: *event* – The AcctEvent to be sent
    Parameter: *destination* – The destination that the AcctEvent should be sent
    Return: An Integer

  - `public Integer sendAcctEvent(List events, String destination);`
    Description: Sends an AcctEvent List to a destination
    Parameter: *events* – The AcctEvent List to be sent
    Parameter: *destination* – The destination that the AcctEvent List should be sent
    Return: An Integer

7.1.12 ch.SWITCH.amaais.commonUtils.remoting.accountingProtocol

- **Class: AccountingProtocol.java**
- **Description:** The Accounting Protocol class intends to pack and unpack objects, from SAML strings to AcctEvent objects (and vice-versa).
- **Methods:**
  - `public AcctEvent pack(String samlMsg);`
    Description: Packs a SAML message

Parameter: *samlMsg* – The SAML message to be packed
Return: An AcctEvent

– `public String unpack(AcctEvent event);`
Description: Unpacks an AcctEvent
Parameter: *event* – The AcctEvent to be unpacked
Return: A String

7.2 API Considerations

It is important to highlight that, naturally, the API of software components can change due to different requirements during the development phase. In case of changes, Deliverable D3 will be updated accordingly reflecting the implementation code at the SVN[5].

The API definition has a link to the document entitled "Accounting Service Messages, Metadata, and Profiles" which is included as an appendix of this deliverable. Basically, the document presented in the appendix describes the elements that can be present in (a) accounting messages, (b) accounting metadata, and (c) accounting profiles. Considering (a), (b), and (c), the following methods/classes have a link, respectively:

- In (a), the methods `public AcctEvent pack(String samlMsg)` or `public String unpack(AcctEvent event)` (Section 7.1.12) should be aware of how to compose accounting messages in a unique manner.
- In (b), the classes `AcctMetadataProvider` and `AcctMetadataFilter` should be aware of which kind of information the metadata can have and how to handle it.
- In (c), the method `public Integer sendAcctEvent(AcctEvent event, String destination)` (Section 7.1.11) should be aware of how to send the accounting message to, *e.g.*, an Accounting Server, and therefore should follow a beforehand agreed standard.

Last but not least, readers can note that some classes do not have any methods (*e.g.*, `AcctMetadataProvider` or `JDBCDataFetcher`). This can happen because of three reasons: either the class has inherited methods only (and therefore, AMAAIS did not define any method as part as its own API), or it is an interface without any methods, or the AMAAIS team did not yet reach a consensus (at the moment of the deliverable publication) of how the API for the given class should be.

[5] https://www.csg.uzh.ch/svn/amaais/code/

8 Summary and Conclusions

In summary, the main achievements of AMAAIS Deliverable 3 are (1) the implementation design re-factoring process aligned to Shibboleth standards, and (2) the use cases workflow with its test cases. As a result, the activities performed in (1) brought a better organization of the implementation components related to its function methods (API), and the activities in (2) brought a better and clear view of which are the normal or abnormal situations considering the AMAAIS use cases in a production scenario.

Specifically, the implementation design re-factoring process resulted either in the creation of new implementation components (*i.e.*, Java classes which should be implemented from scratch) or the adaptation of existent ones. For example, in AMAAIS D2 components like "remoting" or "AttributeFiltering" were embedded into other classes, in diverse parts of the code. In AMAAIS D3, "remoting" and "attributeFiltering" packages are explicitly and separated components that can be used for different purposes, in different classes. In addition, the new "dataFetcher" package brings an extensible manner to fetch data from different Meters: from a simplistic log file to a database that has a specific DB driver. The existence of a "metadataHandler" package also brings the functionality of a central metadata in the Shibboleth standards. It has the responsibility to grab important information (*e.g.*, accounting server certificates) from a central authority and make it available to internal AMAAIS components (*e.g.*, the "remoting"). In summary, the project will reach a higher maturity level by means of its code base being closely aligned with Shibboleth standards and goals.

The use cases workflow with its test cases resulted in a better view of what the AMAAIS project is aiming to achieve when deployed in a production environment. The wokflow definition showed that, *e.g.*, under normal conditions the system should process *X* quantity of SMSs per minute, and therefore the Accounting Server should persist that in a way to not cause any service disruption. The test cases, on the other hand, based on the workflow definition, aim to predict in which situations the system can crash or suffer a disruption. Therefore, the test cases definition – which are high-level described – tried to support the implementation and fine-grained test cases to come.

The AMAAIS D3 intended to form a solid base for the Phase 3 implementation phase that follows. Even though the AMAAIS D2 also presented an implementation phase resulting in the AMAAIS prototype 1.0, the implementation phase expected in AMAAIS Phase 3 is planned to include the new functionalities and code re-factoring, as previously mentioned. Therefore, the code re-factoring is a process that will be extended throughout the whole AMAAIS project development.

Acknowledgement

This deliverable was made possible due to the large and open help of the members of the AMAAIS project. Many thanks to all of them. Moreover, thanks to Chad LaJoie (Shibboleth Java Components Lead at Internet2), Phil Smart and Rhys Smith (Identity and Access Management at Cardiff University) for the ideas and discussions in the AMAAIS development mailing list.

References

[1] AMAAIS Project. *Accounting and Monitoring of AAI Services – Deliverable Phase 1*, October 2009. Available at: http://www.csg.uzh.ch/research/amaais. Visited on: Dec. 2010.

[2] B. Pfitzmann and M. Waidner. Federated identity-management protocols. *LECTURE NOTES IN COMPUTER SCIENCE*, 3364:153, 2005.

[3] AMAAIS Project. *Accounting and Monitoring of AAI Services – Project's Website*, June 2009. Available at: http://www.csg.uzh.ch/research/amaais. Visited on: Dec. 2010.

[4] AMAAIS Project. *Accounting and Monitoring of AAI Services – Deliverable Phase 2*, January 2011. Available at: http://www.csg.uzh.ch/research/amaais. Visited on: Oct. 2011.

[5] Shibboleth: an Internet 2 Project. *Shibboleth Project's Website*, June 2009. Available at: http://shibboleth.internet2.edu/. Visited on: Dec. 2010.

[6] IETF – Internet Engineering Task Force. *RFC 2866 – RADIUS Protocol*, June 2000. Available at: http://www.ietf.org/rfc/rfc2866.txt. Visited on: Dec. 2010.

[7] OASIS. *Security Assertion Markup Language (SAML) – Profiles*, March 2005. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf. Visited on: Oct. 2011.

[8] OASIS. *Security Assertion Markup Language (SAML) – Bindings*, March 2005. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf. Visited on: Oct. 2011.

[9] OASIS. *Security Assertion Markup Language (SAML) – Core Document*, March 2005. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf. Visited on: Oct. 2011.

Appendix

Accounting Service Messages, Metadata, and Profiles, Version 1.0

Working Draft 02

[15 August 2011]

Related Work:

This specification defines extensions for use with SAML V2.0 Core [SAML2Errata], Profile [SAML2Profile], and Metadata [SAML2Meta] specifications.

Declared XML Namespace(s):

TODO

Abstract:

This document describes the methods by which entities acting as accounting record producers and consumers may exchange accounting records. It provides a conceptual overview of this account model, descriptions of extensions to the SAML core and metadata elements, and communication profiles for a model where producers push records to a consumer and a model where consumers pull records from a producer.

Status:

This document is a Working Draft and as such is likely to change, perhaps drastically, with time.

Table of Contents

1 Introduction

This document describes the methods by which entities acting as accounting record producers and consumers may exchange accounting records. It provides a conceptual overview of this account model, descriptions of extensions to the SAML core and metadata elements, and communication profiles for a model where producers push records to a consumer and a model where consumers pull records from a producer.

1.1 Conceptual Overview

Center to this work is the notion of an *accounting record* which represents data about events (e.g., logging in to a site, printing documents, sending SMS messages) that are to be monitored for accounting/reporting/metric purposes. Each accounting record is strongly typed and its types indicates what information it carries. A service which is being monitored in this manner will be referred to as an *accounted service* within this document.

Accounting record producers, of *producer* for short, are actors which interact with a service and monitor it for the events of interest. This monitoring may be done by analyzing log files, database transaction logs, hooking to service-specific APIs, etc. Once such an event is identified the producer is then responsible for creating the appropriate accounting record.

Accounting record consumers, or *consumer* for short, are actors which take in accounting records and operate on upon them. The consumer may also store the records, provide various reporting and visualization services, and offer other features. A consumer may also be able to act as a producer by aggregating consumed data and making it available to other consumers.

1.2 Terminology and Notation

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 [RFC 2119].

Conventional XML namespace prefixes are used throughout the listings in this specification to stand for their respective namespaces as follows, whether or not a namespace declaration is present in the example:

Prefix	XML Namespace	Comments
`saml:`	urn:oasis:names:tc:SAML:2.0:assertion	This is the SAML V2.0 assertion namespace defined in the SAML V2.0 core specification [SAML2Errata].
`samlp:`	urn:oasis:names:tc:SAML:2.0:protocol	This is the SAML V2.0 protocol namespace defined in the SAML V2.0 core specification [SAML2Errata].
`md:`	urn:oasis:names:tc:SAML:2.0:metadata	This is the SAML V2.0 metadata namespace defined in the SAML V2.0 metadata specification [SAML2Meta].
`acct:`	TODO	A namespace defined by this document.
`xsd:`	http://www.w3.org/2001/XMLSchema	This namespace is defined in the W3C XML Schema specification [Schema1]. In schema listings, this is the default namespace and no prefix is shown.
`xsi:`	http://www.w3.org/2001/XMLSchema-instance	This is the XML Schema namespace for schema-related markup that appears in XML instances [Schema1].

This specification uses the following typographical conventions in text: `<ns:Element>`, `Attribute`, **Datatype**, `OtherCode`.

This specification uses the following typographical conventions in XML listings:

```
Listings of XML schemas appear like this.
```

```
Listings of XML examples appear like this.  These listings are non-normative.
```

1.3 Normative References

[RFC 2119] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt.

[SAML2Errata] OASIS Approved Errata, *SAML V2.0 Errata*, October 2009. http://docs.oasis-open.org/security/saml/v2.0/sstc-saml-approved-errata-2.0.pdf.

[SAML2Binding] OASIS Standard, *Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0*, March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf.

[SAML2Profile] OASIS Standard, *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf.

[SAML2Meta] OASIS Standard, *Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0*. March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf.

[Schema1] H. S. Thompson et al. XML Schema Part 1: Structures. World Wide Web Consortium Recommendation, May 2001. See http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.

[Schema2] Paul V. Biron, Ashok Malhotra. *XML Schema Part 2: Datatypes*. World Wide Web Consortium Recommendation, May 2001. See http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

2 Accounting Service Messages and Profiles

The following section describes extensions to [SAML2Errata] and [SAML2Profile] and the use of the messages defined herein.

2.1 Element <acct:PublishRecordRequest>

The `<acct:PublishRecordRequest>` message is used to push accounting records from an accounting record producer to an accounting record consumer.

This element extends **RequestAbstractType** with the following elements:

`<saml:Assertion>` or `<saml:EncryptedAssertion>` [One or More]

Each `<saml:Assertion>` or `<saml:EncryptedAssertion>` carries one or more accounting records for a given accounted service. Multiple assertions may be included in a single request as a means of batching up multiple accounted service record sets in to a single request.

```
<element name="PublishRecordRequest" type="acct:PublishRecordRequestType">
<complexType name="PublishRecordRequestType">
    <complexContent>
        <extension base="samlp:RequestAbstractType">
            <choice minOccurs="1" maxOccurs="unbounded" >
                <element ref="saml:Assertion" />
                <element ref="saml:EncryptedAssertion" />
            </choice>
        </extension>
    </complexContent>
</complexType>
```

2.2 Element <acct:RecordQuery>

The <acct:RecordQuery> message is used by an accounting record consumer to pull accounting records from an accounting record producer.

This element extends RequestAbstractType with the following element:

`<saml:Subject>` [Optional]

If present, specifies the entity about which all returned records must apply. For example, if the requester specifies the identity provider identified by the entity `https://idp.example.org` then only those records that are about that identity provider are returned.

`<acct:RecordType>` [Zero or More]

If present, specifies the types of records to be returned.

```
<element name="RecordQuery" type="acct:RecordQueryType">
<complexType name="RecordQueryType">
    <complexContent>
        <extension base="samlp:RequestAbstractType">
            <sequence>
                <element ref="saml:Subject" minOccurs="0" />
                <element ref="acct:RecordType"
                         minOccurs="0" maxOccurs="unbounded" />
            </sequence>
        </extension>
```

```
    </complexContent>
</complexType>

<element name="RecordType" type="anyURI">
```

2.3 Message Profile

2.3.1 <acct:PublishRecordRequest> Usage

The `<saml:Issuer>` element of the `<acct:PublishRecordRequest>` MUST be omitted.

The use of the `Consent` attribute on the `<acct:PublishRecordRequest>` SHOULD be omitted. The use of this attribute is outside the scope of this document and thus its use is undefined.

The `<saml:Issuer>` element of each of `<saml:Assertion>` and `<saml:EncryptedAssertion>` element MUST identify the producer. The `Format` attribute MUST be omitted or have the value `urn:oasis:names:tc:SAML:2.0:nameid-format:entity`.

The `<saml:Subject>` element of each `<saml:Assertion>` or `<saml:EncryptedAssertion>` identifies the accounted service. The `Format` attribute of the included `<saml:NameID>`, or its encrypted form, SHOULD have the value `urn:oasis:names:tc:SAML:2.0:nameid-format:entity`. Other attributes of the `<saml:NameID>` as well as the `<saml:SubjectConfirmation>` SHOULD be omitted. The use of these other attributes and elements is outside the scope of this profile and thus their use is undefined.

Each `<saml:Assertion>` or `<saml:EncryptedAssertion>` SHOULD omit the `<saml:Condition>` and `<saml:Advice>` elements. The use of these elements is outside the scope of this profile and thus their use is undefined.

Each `<saml:Assertion>` or `<saml:EncryptedAssertion>` MUST only include `<saml:AttributeStatement>` statements. These attribute statements MUST conform to the accounting record attribute statement profile given in section 2.3.3.

2.3.2 <acct:RecordQuery> Usage

The `<saml:Issuer>` element MUST be present and MUST identify the record requester. The `Format` attribute MUST be omitted or have the value `urn:oasis:names:tc:SAML:2.0:nameid-format:entity`.

The use of the `Consent` attribute on the `<acct:RecordQuery>` SHOULD be omitted. The user of this attribute is outside the scope of this document and thus its use is undefined.

If present, the `<saml:Subject>` MUST only contain either a `<saml:NameID>` or `<saml:EncryptedID>`. The `Format` attribute of the included `<saml:NameID>`, or its encrypted form, SHOULD have the value `urn:oasis:names:tc:SAML:2.0:nameid-format:entity`. Other attributes of the `<saml:NameID>` as well as the `<saml:SubjectConfirmation>` SHOULD be omitted. The use of these other attributes and elements are outside the scope of this profile and thus their use is undefined.

2.3.3 <saml:AttributeStatement> Usage

When used in accounting service protocol messages one attribute statement represents one accounting record. Such attribute statements MUST include the following SAML attributes:

Record Type

This attribute specifies the type of this accounting record. This SAML attribute MUST have a `Name` of `XXX/RecordType` and a `Format` of `urn:oasis:names:tc:SAML:2.0:attrname-format:uri`. This attribute MUST contain only a single value whose type is **anyURI**. The given URI SHOULD be a URL that resolved to a human readable description of the record type. This document does not define any concrete record types.

3 Accounting Record Publication Profile

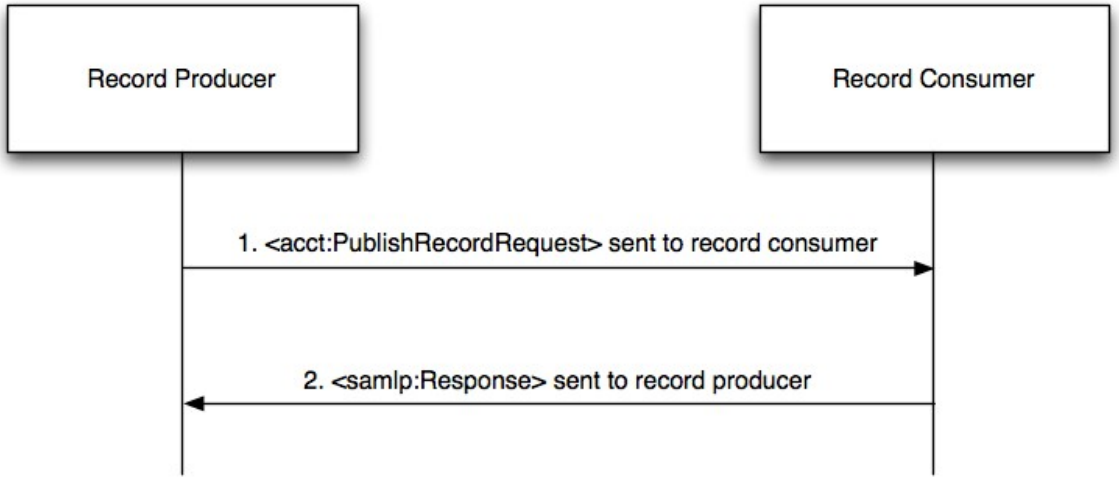
3.1 Required Information

Protocol Identification: TODO

Profile Identification: TODO

3.2 Profile Overview

The message exchange described here provide a means for an accounting record producer to send accounting records to an accounting record consumer and receive acknowledgement that the records have been accepted.



3.3 Profile Description

3.3.1 <acct:PublishRecordRequest> issued to Consumer

To initiate this profile, the producer sends an `<acct:PublishRecordRequest>` message containing accounting records to the accounting record consuming endpoint of the consumer. Metadata MAY be used to determine the endpoint location and bindings supported by the consumer.

The producer MUST use a synchronous binding such as the SOAP binding, [SAML2Binding], to send the request directory to the consumer. The producer SHOULD authenticate itself to the consumer either by signing the `<acct:PublishRecordRequest>` or by using a mechanism supported by the binding in use.

3.3.2 <samlp:Response> issued to Producer

The consumer, after accepting and processing the record request, MUST return a `<samlp:Response>` message containing the appropriate status code in order to complete the exchange.

The consumer MUST authenticate itself to the producer, either by signing the `<samlp:Response>` or using any other binding-supported mechanism.

4 Accounting Record Query Profile

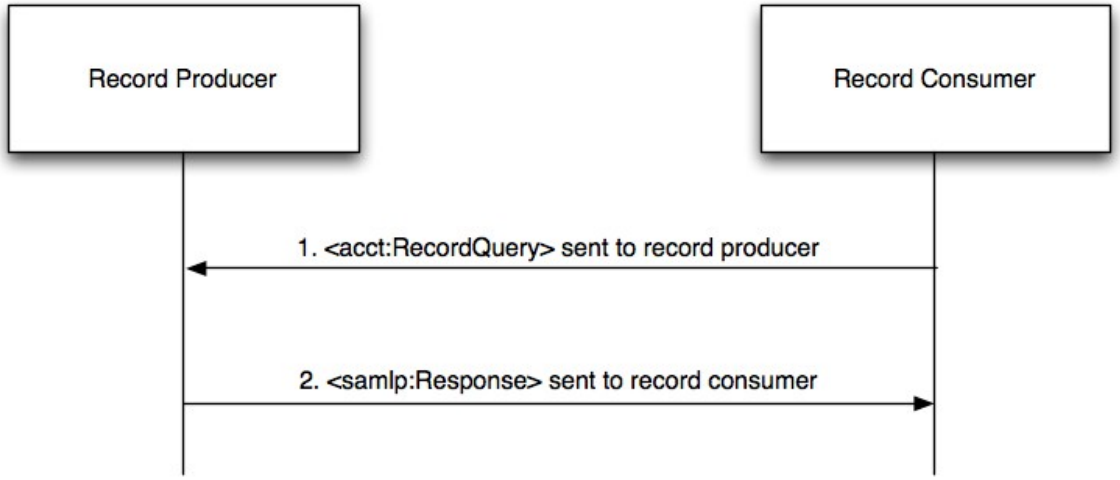
4.1 Required Information

Protocol Identification: TODO

Profile Identification: TODO

4.2 Profile Overview

The message exchange described here provides a means for an accounting record consumer to query for accounting records, and receive matching records, from an accounting record producer.



4.3 Profile Description

4.3.1 <acct:RecordQuery> issued to Producer

To initiate this profile, the consumer sends an `<acct:RecordQuery>` message to the record query service endpoint of the producer. Metadata MAY be used to determine the endpoint location and bindings supported by the consumer.

The consumer MUST use a synchronous binding such as the SOAP binding, [SAML2Binding], to send the request directory to the producer. The consumer SHOULD authenticate itself to the producer either by signing the `<acct:RecordQuery>` or by using a mechanism supported by the binding in use.

4.3.2 <samlp:Response> issued to Consumer

The producer, after accepting and processing the record query, MUST return a `<samlp:Response>` message containing the appropriate status code in order to complete the exchange. In addition, if any records matched the given query request the `<samlp:Response>` will include one or more `<saml:Assertion>` or `<saml:EncryptedAssertion>` containing the accounting records.

The producer MUST authenticate itself to the consumer, either by signing the `<samlp:Response>` or using any other binding-supported mechanism.

5 Accounting Service Metadata Extensions

5.1 Complex Type AccountingRecordConsumer

The **AccountingRecordConsumer** extends **RoleDescriptorType** and indicates that an entity is capable of receiving and processing accounting records. An instance of this type SHOULD include at least one of either `<acct:RecordConsumingService>` or `<acct:SupportedRecordType>`.

`<acct:RecordConsumingService>` [Zero or More]

> Zero or more elements of type **EndpointType** that describe the endpoints that support the accounting record publication protocol defined in this document.
>
> Listing more than one endpoint with the same binding allows for the querying entity to “fail over” to one of the other endpoints in the event that the chosen endpoint is unavailable. In such cases, the client SHOULD choose one of the other available endpoints at random in order to distribute the workload across all endpoints in the event of failures.

`<acct:SupportedRecordType>` [Zero of More]

> Zero of more elements of type **anyURI** that indicate the accounting record types that are supported by this entity.

`wantAssertionsSigned` [Optional]

> Optional attribute that indicates that this entity requires the `<md:Assertion>` carrying the accounting records to be signed. If omitted, the value of this attribute is assumed to be `false`. This requirement is in addition to any requirement for signing derived from the use of a particular profile/binding combination.

The following schema fragment defines the **AccountingRecordConsumer** complex type:

```
<complexType name="AccountingRecordConsumer">
    <sequence>
        <element ref="acct:RecordConsumingService"
                 minOccurs=”0” maxOccurs=”unbounded”/>
        <element ref="acct:SupportedRecordType"
                 minOccurs=”0” maxOccurs=”unbounded”/>
    </sequence>
    <attribute name=”wantAssertionsSigned” type=”boolean” use=”optional” />
</complexType>
```

5.2 Complex Type AccountingRecordProvider

The **AccountingRecordProvider** extends **RoleDescriptorType** and indicates that an entity is capable of providing accounting records. An instance of this type SHOULD include at least one of `<acct:RecordQueryService>`, `<acct:AffiliatedEntity>`, or `<acct:SupportedRecordType>`

`<acct:RecordQueryService>` [Zero or More]

> Zero or more elements of type **EndpointType** that describe the endpoints that support the accounting record query protocol defined in this document.
>
> Listing more than one endpoint with the same binding allows for the querying entity to “fail over” to one of the other endpoints in the event that the chosen endpoint is unavailable. In such cases, the client SHOULD choose one of the other available endpoints at random in order to distribute the workload across all endpoints in the event of failures.

`<acct:AccountedService>` [Zero or More]

Zero or more elements of type **entityIDType** that indicate the accounted services for which this provider can provide accounting information.

`<acct:SupportedRecordType>` [Zero of More]

Zero of more elements of type **anyURI** that indicate the accounting record types that are supported by this entity.

`wantQueryRequestsSigned` [Optional]

Optional attribute that indicates the requirement that <acct:AccountingRecordQuery> messages received by this entity be signed. If omitted, the value is assumed to be `false`. This requirement is in addition to any requirement for signing derived from the use of a particular profile/binding combination.

The following schema fragment defines the **AccountingRecordProvider** complex type:

```
<complexType name="AccountingRecordProvider">
    <sequence>
        <element ref="acct:RecordQueryService"
                 minOccurs="0" maxOccurs="unbounded"/>
        <element ref="acct:AccountService"
                 minOccurs="0" maxOccurs="unbounded"/>
        <element ref="acct:SupportedRecordType"
                 minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="wantQueryRequestsSigned" type="boolean" use="optional" />
</complexType>
```

Appendix A. Acknowledgments

The editor would also like to acknowledge the following contributors:

- Patrik Schnellmann, SWITCH
- Phil Smart, Cardiff University
- Rhys Smith, Cardiff University
- Ian Young, Edinburgh University

Appendix B. Revision History

Changes made to produce Working Draft 2:

- Section 1.1: note that record consumers may also persist record data
- Section 2.1: Attempt to clear up why a record request might carry multiple assertions
- Section 2.3.1: Explicitly prohibit the `<saml:Issuer>` in the request
- Section 5.1: Note that clients should randomly select endpoints in order to help balance the load across endpoints
- Typographical and grammatical edits