CLOUD
ACCOUNTABILITY
PROJECT

D:D-6.2: Accountability Toolset Validation Report

Deliverable Number: D46.2 (D:D-6.2)
Work Package: WP 46
Version: Final
Deliverable Lead Organisation: Armines
Dissemination level: PU
Contractual Date of Delivery (release): 31/03/2016
Date of Delivery: 11/04/2015

Editors

Ali Kassem (Armines), Mario Südholt (Armines)

Contributors

Walid Benghabrit (Armines), Stefan Berthold (KAU), Carmen Fernandez-Gago (UMA), Christian Frøystad (SINTEF), Giorgos Giotis (ATC), Ali Kassem (Armines), David Núñez (UMA), Andersen Santana De Oliveira (SAP), Tobias Pulls (KAU), Jean-Claude Royer (Armines), Thomas Rübsamen (HFU), Philip Ruf (HFU), Mario Südholt (Armines), Vasilis Tountopoulos (ATC)

Reviewers

Frederic Gittler (HP), Andersen Santana De Oliveira (SAP), Christoph Reich (HFU)

A4Cloud
www.a4cloud.eu
Accountability For Cloud and Other Future Internet Services

FP7-ICT-2011-8-317550-A4CLOUD

SEVENTH FRAMEWORK PROGRAMME

Executive summary

The main objective of the A4Cloud project is the provision of new processes, methods, techniques, and tools for the construction of accountable Cloud architectures that increase the awareness and level of enforcement of accountability properties in future Cloud architectures.

A main contribution to this end is the different (preventive, detective and corrective) accountability-related tools designed and implemented as part of the project. Some of these accountability tools, such as the Accountability Primelife Policy Engine (A-PPLE), the Audit Agent System (AAS), Data Transfer Monitoring Tool (DTMT) and the Incident Management Tool (IMT), play a particular role because they handle accountability properties at runtime in an automated fashion. The accountability properties are expressed using policies that are used to configure the tools. These tools achieve a high degree of automation, all the while being able to enforce diverse accountability properties at very different levels of Cloud architectures: they include high-level properties such as intrusion attempts in personal data of Cloud Subjects, as well as low-level properties, such as the identification of sensitive data in virtual machine snapshots at the infrastructure level.

Due to the diversity of the tools, the different problems they resolve, the interactions between actors (data subjects, data processors, etc.) and the data involved, the correct enforcement of their accountability properties is difficult to validate. This is the case, in particular, because most high-level accountability properties, for example “Make explicit by whom a subject’s data is processed” (a transparency property in the sense of the attributes defined as part of WP C-2), cannot be expressed in terms of the low-level accountability properties directly supported by any single tool, that is the policies of any tool; instead, they have to be expressed by a combination of low-level properties of different tools that correspond to chains of events handled by the tools during runtime. The high-level property above, for example, can be handled by the A-PPLE tool that can detect data processing operations, the AAS tool that constructs the evidence of which actor performed the processing and and the IMT tool that sends corresponding notifications.

For these reasons, the runtime accountability tools have been chosen as subject of validation in workpackage D-6. The validation has to satisfy three main objectives.

1. The validation methodology should support the validation of accountability properties that are expressible in terms of accountability attributes, such as transparency and responsiveness, over the actions on tools and their effects on subjects and their resources.
2. It should be effective and practical, thus being useful to perform concrete validation tasks on tool compositions that execute in real Cloud environments.
3. It should be extensible in order to accommodate new accountability properties, tools and services.

Currently, there are no commercial or academic tools for the validation of accountability properties that are enforced by different tools. Existing validation approaches and tools (in a large sense, covering formal verification techniques as well as general testing tools) are either specific to individual tools, and support a small set of accountability properties or both. Moreover, no existing tool meets the requirements stated above.

The main goals of WP D-6 are the provision of a validation methodology for A4Cloud tools, corresponding tool support and its application to the validation of diverse accountability properties. Concretely, we have developed a validation methodology for accountability tools based on accountability scenarios and the monitoring of temporal logic formulas. Our validation method is

mainly geared for tool developers and Cloud providers that want to integrate new accountability tools into their ecosystems; it is, however, also useful to actors that have to audit or supervise Cloud ecosystems.

As part of our work, two validation tools have been designed and implemented. First, the Assertion Tool which enables validation scenarios to be executed and validated through interactions with the A4Cloud tools. Concretely, the Assertion Tool supports the three objectives through the following main characteristics:

- Accountability properties can be expressed in terms of a set of accountability predicates that capture basic accountability-related actions of the A4Cloud tools and are used to built complex accountability assertions that validate event chains handled by tool combinations.
- It allows the effective and practical execution of concise accountability validation scenarios in order to validate flexible compositions of A4Cloud tool actions and accountability properties.
- The validation method includes extensible interfaces for tools that may be used to integrated tools external to A4Cloud.

Second, the Accountability Monitor, an extension of the Accountability Lab [dOSP$^+$15], that allows the distributed verification at runtime of temporal logic formulas. This tool is complementary to the Assertion Tool in that it supports more general specifications but lacks specific support for fine-grained accountability properties of the A4Cloud tools.

The Assertion Tool and the Accountability Monitor are the first accountability validation tools that are, respectively, based on validation scenarios and distributed temporal logic formulas.

To illustrate the work of the Assertion Tool, we have defined five validation scenarios that allow different accountability properties of the A4Cloud tools to be validated. The defined validation scenarios are derived from the wearable scenario which is developed and deployed by the A4Cloud project as part of WP D-7. This scenario constitutes a realistic and topical scenario, in which the involved business actors have to take the appropriate actions to ensure that the collection and processing of the customers' personal data are handled responsibly, based on the established regulations and declared security organizational policies. For each scenario, we provide a detailed description of how the scenario is used for validation of specific accountability properties using the Assertion Tool. We also describe, for each scenario, the related accountability metrics, which are defined in WP C-5 to assist the assertion process, in particular, by providing assurance of compliance. Metrics can provide a qualitative or quantitative overview of how this assurance is achieved. These scenarios have then be used to automatically validate the accountability properties of the tools in two concrete deployments of the supply chain provided by partners ATC and HFU.

Concerning AccMon, we demonstrate how it can be used in collaboration with the Assertion Tool to validate execution traces against a behavioral formula. More precisely, we provide first-order distributed temporal formulas, which are related to some of the validation scenarios, and we then check these formulas, using AccMon, on global execution dummy traces.

An advantage of AccMon over the Assertion Tool is that it can produces formulas which express what could happen if new events are appended in the future to the trace under verification. While the Assertion Tool can only checks a trace of events up to the current state, so in case of runtime validation it re-collects the the potential events every a given amount of time and re-checks whether the related assertions are satisfied or not.

Contents

List of figures

List of tables

Glossary

A4Cloud toolkit

Subset of the A4Cloud tools that is subject to validation by the Assertion framework. The A4Cloud toolkit consists of the runtime policy-based accountability tools, notably the Accountability Primelife Policy Engine (A-PPLE), the Audit Agent System (AAS), Data Transfer Monitoring Tool (DTMT) and the Incident Management Tool (IMT).

A4Cloud tools

Complete set of A4Cloud tools developed as part of the A4Cloud project.

Accountability (A4cloud definition from WP C-2)

Accountability for an organization consists of accepting responsibility for data with which it is entrusted in a cloud environment, for its use of the data from the time it is collected until when the data is destroyed (including onward transfer to and from third parties). It involves the commitment to norms, explaining and demonstrating compliance to stakeholders and remedying any failure to act properly.

Accountability assertion

A boolean expression representing an accountability property that can be validated using the Assertion tool and the AccMon tool. Assertions are built up from predicates..

Accountability Monitor tool (AccMon)

The AccMon tool is one of the main components of the Assertion framework. It enables the distributed verification of first-order temporal formulas at run time.

Predicates

Boolean term used to express the lower-level accountability properties directly supported by the A4Cloud tools and the environment.

Accountability property

Property representing accountability obligations. Accountability properties range from high-level properties over the accountability attributes to low-level tool-specific or environment-specific properties.

Assertion framework

Software framework enabling the validation of the A4Cloud toolkit.

Assertion Tool

One of the main components of the Assertion framework. It orchestrates the validation of an A4Cloud toolkit, in particular by running or observing validation scenarios and checking assertions.

Evidence

Evidence is a collection of data, metadata, routine information and formal operations performed on data and metadata which provide an attributable and verifiable account of the fulfillment of relevant obligations. Evidence can be used to support an argument shown to a third party about the validity of claims about the appropriate and effective functioning (or not) of an observable system.

Validation scenario

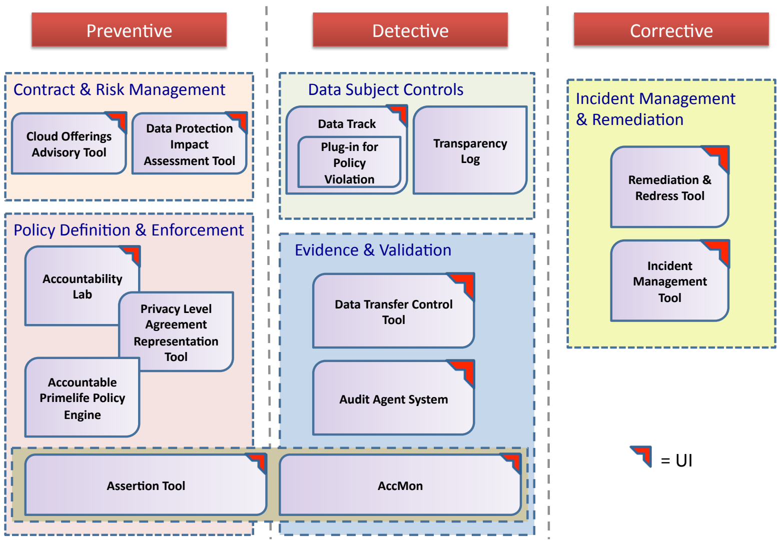
A validation scenario is a narrative of foreseeable interactions between the A4Cloud tools (including the Assertion Tool) and the environment. Scenarios are used to validate some predefined accountability properties.

Chapter 1

Introduction

The A4Cloud project strives for new processes, methods, techniques, and tools for the construction of accountable Cloud architectures that increase the awareness and level of enforcement of accountability properties in the future Cloud. Hence, the project has developed models for the accountable Cloud and designed significant software support for accountability in the Cloud.

The different accountability-related tools designed and implemented as part of the project are a cornerstone of this notion of the accountable Cloud. Figure 1.1 shows a high-level representation of the A4Cloud tools, classifying them by their type of effects (columns) and by functionality (blocks). Some of the tools, such as the contract and risk management tools are intended to be used manually as advisory tools, while most can be used in an automated fashion as part of Cloud ecosystems and infrastructures. Some enforcement and auditing tools, such as the Accountability Primelife Policy Engine (A-PPLE), the Audit Agent System (AAS), Data



Preventive
Detective
Corrective
Contract & Risk Management
Cloud Offerings Advisory Tool
Data Protection Impact Assessment Tool
Data Subject Controls
Data Track
Plug-in for Policy Violation
Transparency Log
Incident Management & Remediation
Remediation & Redress Tool
Incident Management Tool
Policy Definition & Enforcement
Accountability Lab
Privacy Level Agreement Representation Tool
Accountable Primelife Policy Engine
Evidence & Validation
Data Transfer Control Tool
Audit Agent System
Assertion Tool
AccMon
= UI

Figure 1.1: High level view of the A4Cloud tools [GKP+15a]

Transfer Monitoring Tool (DTMT) and the Incident Management Tool (IMT), play a particular role because they handle accountability properties at runtime in an automated fashion.

The properties are expressed using accountability policies that allow actions to be defined that are executed in order to meet obligations triggered, for example, when resources of data subjects are manipulated. Policies are used to configure the tools that then identify policy violations, construct evidence for such violations and notify them, for instance to the data subject whose data has been modified.

The A4Cloud tools achieve a high degree of automation, all the while being able to enforce diverse accountability properties at very different levels of Cloud architectures. They encompass high-level properties, such as a data subject knows about all locations at which her data is stored or intrusion attempts in personal data of Cloud Subjects. They also have to satisfy low-level properties, such as the identification of sensitive data in virtual machine snapshots at the infrastructure level.

Due to the diversity of the tools, the interactions between actors and the data involved, the correct enforcement of accountability properties by the A4Cloud tools is difficult to be validated. This is the case, in particular, because most high-level accountability properties, for example "Make explicit by whom a subject's data is processed" (a transparency property in the sense of the attributes defined as part of WP C-2), cannot be expressed in terms of the low-level accountability properties directly supported by any single tool, that is the policies of any tool; instead, they have to be expressed by a combination of low-level properties of different tools that correspond to chains of events handled by the tools during runtime. The high-level property above, for example, can be handled by the A-PPLE tool that can detect data processing operations, the AAS tool that constructs the evidence of which actor performed the process

For these reasons, the runtime accountability tools have been chosen as subject of validation in workpackage D-6. The validation has to satisfy three main objectives.

1. The validation methodology should support the validation of accountability properties that are expressible in terms of accountability attributes, such as transparency and responsiveness, over the actions on tools and their effects on subjects and their resources.

2. It should be effective and practical, thus being useful to perform concrete validation tasks on tool compositions that execute in real Cloud environments.

3. It should be extensible in order to accommodate new accountability properties, tools and services.

Software validation, notably for the Cloud, can be performed in many different ways, notably on the basis of automatized or manual formal approaches, such as formal proofs of properties [BGRS15, BM14, CEH$^+$05], or systematic runtime verification and testing methods [BB13a, GBT11, GI09]. However, currently, there are no commercial or academic tools for the validation of accountability properties that are enforced by different tools. Existing validation approaches and tools (in a large sense, covering formal verification techniques as well as general testing tools) are either specific to individual tools, support a small set of accountability properties or both. Moreover, no existing tool meets the three requirements stated above.

The main goals of WP D-6 are the provision of a validation methodology for A4Cloud tools, corresponding tool support and its application to the validation of diverse accountability properties. Concretely, we have developed a validation methodology for A4Cloud using runtime verification based on accountability scenarios and temporal logic based formulas. Our validation method is mainly geared towards tool developers and Cloud providers that want to integrate new accountability tools into their ecosystems. It requires, in particular, the definition of validation scenarios and logic formulas that must be tailored towards the tools and execution ecosystem in which the tools run. The method can, however, also be helpful to auditors and regulatory/supervisory authority of Cloud ecosystems: in this case, predefined scenarios and formulas can be harnessed that represent the accountability properties

As part of our work, two validation tools have been designed and implemented. First, the *Assertion Tool* (AT) which enables validation scenarios to be executed and validated through interactions with the A4Cloud tools. Concretely, the Assertion Tool supports the three objectives through the following main characteristics:

- Accountability properties can be expressed in terms of a set of accountability predicates that capture basic accountability-related actions of the A4Cloud tools and are used to built complex accountability assertions that validate event chains handled by tool combinations.
- It allows the effective and practical execution of concise accountability validation scenarios in order to validate flexible compositions of A4Cloud tool actions and accountability properties. Accountability properties involving multiple tools are verified by checking accountability assertions that are built from accountability predicates.
- The validation method includes extensible interfaces for tools that may be used to integrated tools external to A4Cloud. The tool interfaces include dedicated validation interfaces to pass information in a controlled, notably secure and sanitized manner, directly from the A4Cloud tools to the Assertion Tool.

Second, the Accountability Monitor, an extension of the Accountability Lab [dOSP+15], that allows the distributed verification at runtime of temporal logic formulas. This tool is complementary to the Assertion Tool in that it supports more general specifications but lacks specific support for fine-grained accountability properties of the A4Cloud tools.

The Assertion Tool and the Accountability Monitor are the first accountability validation tools that are, respectively, based on validation scenarios and distributed temporal logic formulas.

To illustrate the work of the Assertion Tool, we have defined five validation scenarios that allow different accountability properties of the A4Cloud tools to be validated. The defined validation scenarios are derived from the wearable scenario which is developed and deployed by the A4Cloud project as part of WP D-7. This scenario constitutes a realistic and topical scenario, in which the involved business actors have to take the appropriate actions to ensure that the collection and processing of the customers' personal data are handled responsibly, based on the established regulations and declared security organizational policies.

For each scenario, we provide a detailed description of how the scenario is used for validation of specific accountability properties using the Assertion Tool. We also describe, for each scenario, the related accountability metrics, which are defined in WP C-5 to assist the assertion process, in particular, by providing assurance of compliance. Metrics can provide a qualitative or quantitative overview of how this assurance is achieved. These scenarios have then be used to automatically validate the accountability properties of the tools in two concrete deployments of the supply chain provided by partners ATC and HFU.

Overview of the validation methodology A high-level overview of the A4Cloud tools validation methodology is shown in Figure 1.2.

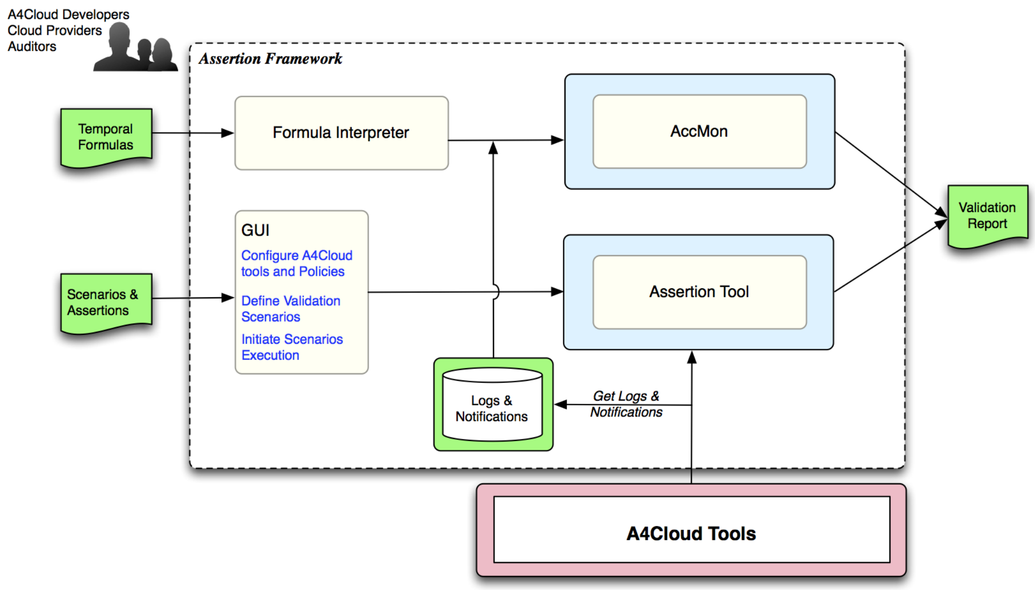


Figure 1.2: Overview of the validation methodology

The validation of the A4Cloud tools consists in providing guarantees on the correctness of these tools once combined together. As part of our validation methodology, high-level accountability properties are expressed in an extensible set of predefined assertions. They come equipped with implementations in terms of lower-level and often tool-specific accountability predicates that capture the accountability properties directly supported by the A4Cloud tools. The set of tool-specific accountability predicates has been derived from a detailed analysis of the accountability properties that can be directly enforced by the A4Cloud tools.

The validation itself consists in runtime verification of the executing A4Cloud system performed on the basis of validation scenarios and validation specifications in form of temporal logic formulas. The validation scenarios are declarative and executable (Java-based) specifications that are built from blocks representing tool interactions and accountability assertions. The scenarios are then run by the Assertion Tool (AT). Logic-based formulas are monitored continuously by the AccMon component of the AccLab tool suite.

During a run of a validation scenario, the AT tool will gather information from the A4Cloud tools, essentially in the form of logs and notifications. This information provides the basis for the decision whether accountability assertions are satisfied or not. Similarly, the AccMon component gathers information about events generated by the A4Cloud tools in order to decide whether its logic-based specifications are met during system runs. Finally, the tools provide validation reports over the assertions or logic-based properties that have been satisfied or not.

Validation scenarios and demo As part of WP D-7 a wearable scenario has been developed and deployed in order to demonstrate the accountability framework. This scenario involves commercial and institutional actors, which have to take the appropriate actions to ensure that the collection and processing of the customers' personal data are handled responsibly, based on the established regulations and declared security organizational policies.

In WP D-6, we have defined five validation scenarios, derived from the D-7 wearable scenario, in order to demonstrate how the Assertion Tool can be used to validate the accountability properties of the A4Cloud tools. For each scenario, we provide a detailed description of how the scenario is used for validation of specific accountability properties using the Assertion Tool. We also describe, for each scenario, the related accountability metrics, which are defined in W PC-5 to assist the assertion process, in particular, by providing assurance of compliance. Metrics can provide a qualitative or quantitative overview of how this assurance is achieved. These scenarios have then be used to automatically validate the accountability properties of the tools in two concrete deployments of the supply chain provided by partners ATC and HFU.

AccMon and logical formulas We show how to use AccMon in order to validate the scenarios and how to write the corresponding logical formula for each test. The formula are written is FODTL (First order distributed temporal logic) and are checked on a global trace collected by AccMon on different tools.

Relation with other workpackages The validation facilities developed in WP D-6 are related to several other workpackages as detailed later in this deliverable.

First, the different A4Cloud tools that have subjected for validation in D-6 have been developed as part of other workpackages in streams C and D.

Furthermore, the Assertion Tool has been tested and used for validation within two OpenStack-based Cloud environments, notably the environment developed for the A4Cloud demonstrator as part of WP D-7.

Finally, the validation methodology is integrated with the A4Cloud architecture as defined by WP D-2, harnesses accountability properties following the accountability model developed as part of WP C-2, and uses metrics for validation developed in WP C-5.

Organization of the deliverable This deliverable is organized as follows.

We present our validation methodology, related work on the validation of accountability properties, as well as our notion of accountability assertions in Chapter 2.

Chapter 3 introduces the assertion framework and its architecture. It also presents the two tools we have developed for validation, the Assertion Tool and Accountability Monitor. Finally, we also review the A4Cloud tools that we have subjected to validation.

In Chapter 4, we provide the validation scenarios for tools validation. Then, we present the A4Cloud tools validation we have performed, based on these validation scenarios, on the Cloud infrastructures provided within the A4Cloud project, notably the demo infrastructure developed as part of Workpackage D-7.

Chapter 5 presents a conclusion.

Finally, Appendix A shows the concrete examples of the data exchanged between the A4Cloud tools.

Chapter 2

Accountability Validation

One of the main objectives of A4Cloud is to "*enable Cloud service providers to give their users appropriate control and transparency over how their data is used*" [PTC$^+$12]. The A4Cloud partners provide different tools to assist Cloud stakeholders to meet this objective. In order to be sure that the correctness of the A4Cloud tools is met once combined together, their correct enforcement of accountability properties has to be validated. Validation is defined in the IEEE standard glossary of software engineering terminology as: "*The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements*" [IEE90].

As part of the WP D-6, the partners have developed a validation methodology based on the runtime verification of accountability properties based on two different kinds of specification. First, declarative, executable validation scenarios that define tool interactions, as well as, accountability assertions support the verification of accountability properties. Second, temporal logic formulas can be monitored and verified during execution. The validation methodology can be harnessed by tool developers at the end of the tool development phase as well as Cloud service providers during the setup or validation runs of their Cloud ecosystems.

This chapter is structured as follows. The validation methodology is presented in Section 2.1. Related work on the validation of accountability properties is discussed in Section 2.2. Accountability assertions that are used to express accountability properties in validation scenarios are introduced in Section 2.3. Finally, a summary is presented in Section 2.4.

2.1 The validation methodology

An overview of the A4Cloud tools validation methodology is shown in Figure 2.1.

The validation of the A4Cloud tools consists in providing guarantees that the composition of these tools preserve accountability properties. Generally speaking, accountability properties range from high-level accountability properties over the accountability attributes defined in work-package C-2 [FP$^+$14], notably transparency, responsiveness, responsibility and remediability to low-level tool-specific or environment-specific properties, for example, relative to the OpenStack Cloud infrastructure environment.

As part of our validation methodology high-level accountability properties are expressed in terms of an extensible set of predefined accountability predicates that represent lower-level and

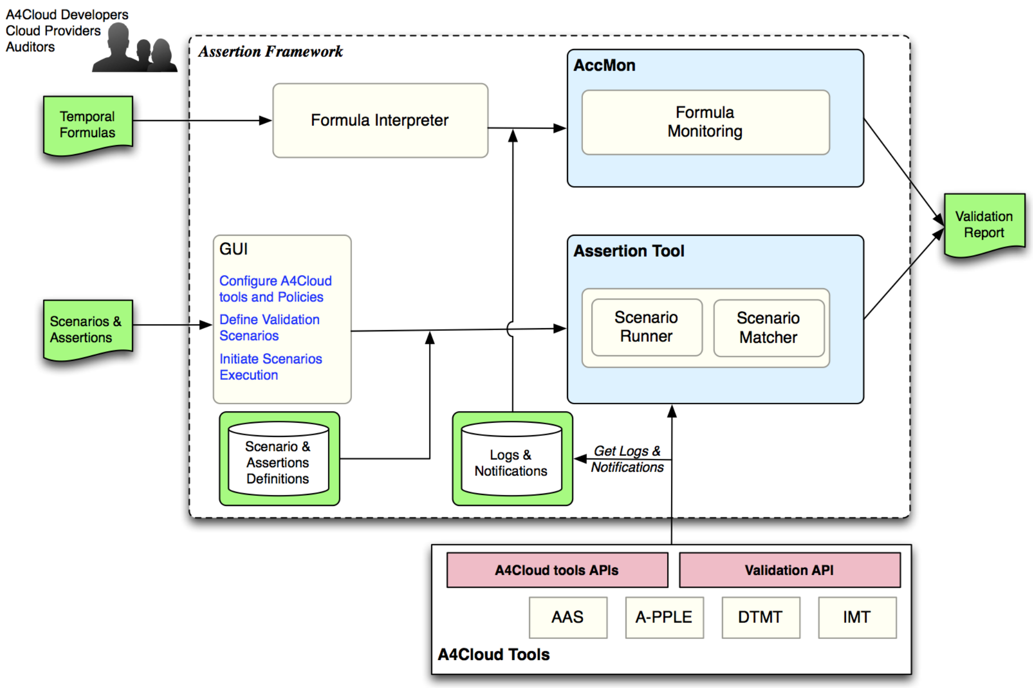


Figure 2.1: Overview of the validation methodology

often tool-specific accountability predicates. These predicates3 capture the accountability properties directly supported by the A4Cloud tools. The predicates enable, in particular, the formulation of the accountability properties that are defined using policies by the runtime accountability tools AAS, A-PPLE, DTMT and IMT. The set of tool-specific, notably policy-defined, accountability predicates has been derived from a detailed analysis of the accountability properties that can be directly enforced by the four A4Cloud runtime accountability tools.

The validation itself consists in runtime verification of the executing A4Cloud system performed on the basis of validation scenarios and validation specifications in form of temporal logic formulas. The validation scenarios are declarative and executable (Java-based) specifications that are built from blocks representing tool interactions and accountability assertions. The scenarios are then applied by the Assertion Tool in one of two modes. In the "scenario runner" mode, the AT executes the tool interactions that are part of a scenario and validates the accountability assertions when they are encountered as part of the scenario. In the "scenario matching" mode, the AT verifies the accountability assertions continuously while observing the autonomously-executing system. Logic-based formulas are monitored continuously by the AccMon component of the AccLab tool suite.

At the end or during the execution and matching of a validation scenario, the AT tool will gather information, essentially in the form of logs and notifications, from the A4Cloud tools. This information provides the basis for the decision whether accountability assertions are satisfied or not by the run of the validation scenarios. Similarly, the AccMon component gathers infor-

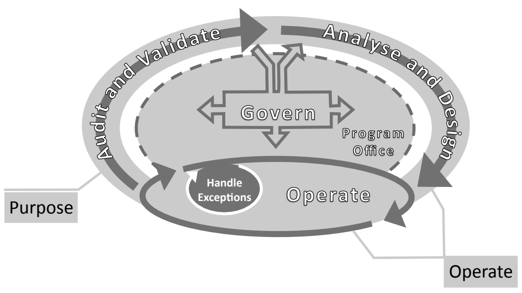


Figure 2.2: A4Cloud organizational lifecycle (from WP D-2 [GKP+15b])

mation about events generated by the A4Cloud tools in order to decide whether its logic-based specifications are met during system runs.

Finally, during execution and at the termination of validation scenarios as well as during the monitoring of logic-based accountability properties the two tools provide validation reports over the assertions or logic-based properties that have been satisfied or not.

Validation and the A4Cloud architecture and lifecycle

The AT and AccMon tools serve to validate tools during their development by tool developers or during their execution by Cloud providers or auditors. Both tools integrate smoothly into the organizational lifecycle, see Figure 2.2. They operate at the end of the analysis and design phase and during the operational phase, in particular, in order to detect exceptional situations; both then feed into a later analysis and design phase in order to improve the accountability properties of the Cloud ecosystem.

In terms of the A4Cloud conceptual reference architecture, see Figure 2.3, the AT and AccMon tools principally need direct input about the resources and computations performed by the Cloud service provider, as well as, information about Cloud subjects and Cloud service customers. Information from regulatory authorities, as well as, contracts established using Cloud brokers is also relevant as long as it is represented in machine-readable form, for example, using policies.

Metrics for accountability validation

Metrics can assist the assertion process, in particular, by providing assurance of compliance. Metrics can provide a qualitative or quantitative overview of how this assurance is achieved.

Supplementary to the validation methodology, it is possible to further assess the fulfillment of assertion requirements through the evaluation by means of accountability metrics. In the context of A4Cloud, measurement techniques for assessing the concept of Accountability have been developed as part of WP C-5, which led to a catalog of accountability metrics [NFGA+14]. The catalog is composed of 39 metrics, organized in three categories:

- Verifiability and Compliance. This category groups metrics devoted to demonstrating compliance to good practices and regulations.

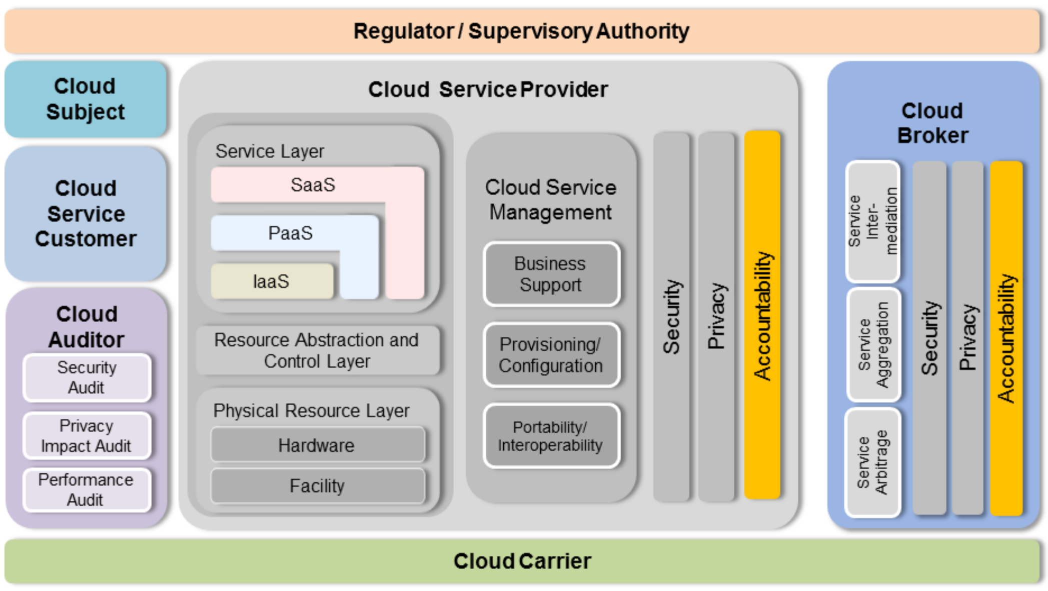


Figure 2.3: A4Cloud conceptual reference architecture (from WP D-2 [GKP$^+$15b])

- Transparency, Responsibility and Attributability. This group of metrics is related to measuring the characteristics about the internal processes that provide Accountability.
- Remediability and Incident Response. This category groups metrics encompasses metrics related to remediation, redress, and incident response.

This catalog supports the demonstration that proper practices and mechanisms for privacy, security and information governance are in place.

Once suitable metrics are identified they can in principle be used in the process of accountability validation. For each validation scenario, relevant metrics can be defined. This includes defining the criteria of the results of the metric that support the validation of accountability properties. By using data derived from the execution of the validation scenario we could proceed to apply the metrics and determine whether the validation scenario are validated.

However, it is important to remark that the nature of the metrics in WP C-5 is oriented towards the evaluation of organizational and governance aspects, rather than automated measurements. For this reason, most of them are not directly applicable in the validation scenarios. These scenarios model several expected outcomes of the A4Cloud tools, following a “Detect and Notify” pattern: a particular set of tools detects some deviation of the normal behavior and notifies the corresponding stakeholder. The Metrics catalog of WP C-5, because of its nature, is not concerned with the Detection part. However, there are relevant metrics for the Notification part, which can be useful as a complement to the performed validation.

For example, the following metric describes quantitatively how timely the notifications are delivered to interested stakeholders (e.g., end-users, administrators, etc)

- Input: This metric is computed using the following parameters:

  - N – Total number of notifications for a given period of time
  - A_i – Time of the i-th notification during the given period of time, expressed in UNIX time
  - B_i – Time of the i-th incident during the given period of time, expressed in UNIX time

- Formulation and output: Output = $\frac{\sum_{i=1}^{N}(A_i - B_i)}{N}$

This metric can be used in runtime-based validations, such as our validation methodology. We will describe the metrics related for each validation scenario in Chapter 4.

Finally, it should be noted that the metrics described in this document are not intended to be integrated in the runtime execution of the validation process, but are rather meant to be means to enrich the validation results.

2.2 Related work

The assertion tool uses a validation scenario-based methodology to validate at runtime accountability properties of the policy-enforcement A4Cloud tools. Currently, no tools (commercial or otherwise) exist for the validation of tool sets for accountability. Existing approaches for the testing and monitoring of accountability properties are limited to individual tools. Furthermore, the Assertion tool is unique in providing a flexible assertion language that allows the definition of a wide range of accountability properties, including transparency, responsiveness and responsibility properties.

In this section, we present related work which deals with the validation of accountability properties and scenario testing. We classify the different approaches in three groups:

- Scientific approaches concerned with the verification of accountability properties, see Section 2.2.1.
- Accountability-aware tools which collect logs and evidence to support accountability or verify accountability properties, see Section 2.2.2.
- Runtime verification and monitoring of logical formula, see Section 2.2.3.

2.2.1 Scientific approaches to accountability validation

Scientific approaches to the validation of accountability properties either come in the form of formal approaches or runtime verification techniques. However, all academic approaches are limited with respect to the validation requirements of the A4Cloud tools because of two main reasons. First, none of them considers the accountability properties of tool chains and diverse sets of accountability properties. All existing approaches only consider single tools and small sets of properties. Second, as to be best of our knowledge, none of the formal approaches to accountability properties has been applied to large-scale real-world systems as those considered in the A4Cloud project. Similarly, the scientific approaches to the validation of accountability properties have not been applied to real-world systems but have only been evaluated on some small systems or system components.

Formal approaches. In the context of work on formal approaches to accountability, Küsters *et al.* [KTV10] have been proposed formal definitions for accountability and verifiability and recognized verifiability as a weaker variant of accountability. The authors provide two definitions of accountability: a symbolic one in the Dolev-Yao model and a computational one with a cryptographic model, and demonstrate the applicability of their approach by analyzing several cryptographic protocol specifications. However, their framework needs to be instantiated for each application by identifying relevant goals that are to met and is therefore very unwieldy.

Mazza *et al.* [EM11] have proposed a formal framework for specifying and reasoning about logs as electronic evidence. The authors consider a decentralized logs, and an analysis method, defined prior to legal disputes, to determine liability of the parties for predefined misbehaviors. Moreover, they study how previous results can be used in the incremental analysis of larger inputs to obtain precise or approximated results, and illustrate their approach with an example of a travel arrangement service. The presented framework allows to specify liability, claims, and logs as electronic evidence. It uses the B-method, a methodology that focuses both on data and behaviors, and which is referenced as a standard in industry. Note that, the authors assume that origin, authentication, and integrity properties hold for messages; however, they claim that the framework can easily be adapted to relax these assumptions.

Work based on runtime validation. Several researchers have investigated runtime validation for accountability or related properties.

Yao *et al.* [YCW$^+$10, YCW$^+$12] have proposed an approach for bringing accountability to the cloud. The proposed approach consists of a novel design to achieve trustworthy Service Oriented Architectures, identifying and associating failures and misbehaviors with the corresponding responsible entities. Administrated by a new service the system promises accountability as a service. The authors focused on evidence provision for accountability of cloud services. They introduced their concept of accountability, which includes attribution as a major task, and with the core functionalities: logging, monitoring and auditing, and dispute resolution. System monitoring and auditing includes a logic mechanism that verifies compliance based on analysis of pre-established Service Level Agreements (SLAs) as well as a business operation logic that defines the correct flow between services. Policy definitions are also introduced to fill the gap between SLA and business logic definitions and monitoring mechanisms used to evaluate service legitimacy. However, they only addressed business exchanges at the SaaS level, and accountability issues related to personal data handling are not considered.

Wang and Zhou [WZ10] have proposed a collaborative monitoring mechanism to support accountability for a multi-tenant database with a centralized external service. They proposed an accountability service, namely, a third party service which allows clients to verify the correctness of the data and the execution of their business logic in the cloud platform. The third-party service is responsible for intercepting endpoints interfacing the client and the service. It captures data relevant to the SLA contract and uses the Merkle B-tree method [LHKR08] to authenticate the data stored in the database. The paper concentrates on the centralized service, however it also considers a distributed version. This approach was discussed and evaluated with through experiments in Amazon EC2 cloud service.

More recently, Masmoudi *et al.* [MLK14] have proposed a multi-tenant services monitoring approach that keeps monitoring service execution at runtime and detecting privacy violations. The proposed approach based on patterns as abstract model which provide reusable and suitable description of multi-tenant services based on accountability rules. The considered rules are specified using simple Event Condition Action model, and are enforced using Aspect Oriented

Programming. In addition, a Pattern Matching algorithm is defined for the appropriate multi-tenant accountability pattern selection. The authors also have proposed a middleware layer to implement their approach into cloud architecture. Later, they have extended their approach in [MSLK15] by enabling the post-violation diagnosis of multi-tenant services accountability. It is based on service patterns as an abstract model to provide reusable and suitable description of the different characteristics of the analysis process, and an algorithm for pattern selection and instantiation.

Sundareswaran *et al.* [SSL12] have proposed an end-to-end decentralized accountability framework, namely Cloud Information Accountability (CIA), to keep track of the user data usage in the cloud. The authors suggest an object-centered approach that packs the logging mechanism together with users' data and policies. Thus a data owner can track whether or not the service-level agreements are satisfied, but also can enforce access or usage control. They consider two possible modes for auditing: push mode (where owners receive logs) and pull mode (where auditors extract logs). The authors have tested their framework by setting up a small cloud, using the Emulab testbed [Tes]. The results show the efficiency, scalability and granularity of the approach. However, they do not propose an explicit accountability definition; the paper focuses on data accountability and a technical framework to achieve it.

Naskos *et al.* [NSKG15, NSG$^+$15] presents a model-driven approach for the dynamic provisioning of cloud resources in an autonomic manner. The proposed approach is based on the probabilistic model checking of Markov Decision Processes (MDPs) at runtime. The MDPs are dynamically instantiated at runtime using logs and measurements, then the data are verified using the PRISM model checker for the provisioning/deprovisioning of cloud resources. The authors have been experimentally evaluated their approach using a NoSQL database cluster running on a cloud infrastructure.

2.2.2 Accountability-aware tools

While there are no tools for the validation of accountability properties of tool chains as enabled by the A4Cloud validation methodology, a small number of tools helps in collecting logs and evidences, or in performing data monitoring and auditing.

HyTrust [HyT] is a commercial virtual appliance tool focusing on cloud auditing and policy enforcement. It is a log report and policy enforcement tool, which sits between the virtual infrastructure and its administrators. More precisely, HyTrust intercepts all the administrative requests for a virtual infrastructure, and determines if these requests are in accordance with the policies defined by the virtual infrastructure manager, before permitting or denying it accordingly. By logging all requests, records are produced that can be used for regulatory compliance and auditing, troubleshooting, and forensic analysis. HyTrust addresses the low-level system layer of accountability in the cloud. This tool does not perform any validation by itself but provides information which may be useful for validation. Furthermore, it focuses on the virtualization layers and does not support higher-level accountability properties.

Haeberlen *et al.* [HARD10] have been introduced the accountable virtual machines (AVMs), which allows to audit the execution of a system in a distributed context. Precisely, AVMs can execute binary software images in a virtualized copy of a computer system, and record non-repudiable information that allows auditors to subsequently check whether the software behaved as intended. To demonstrate the effectiveness of AVMs in practice, the authors have designed and implemented a prototype monitor based on VMware Workstation, and used it to detect several existing cheats. This tool's runtime verification capabilities are much more limited in scope than what is required by the A4Cloud tools.

Massonet *et al.* [MNP$^+$11] have presented a monitoring and audit logging system for data location compliance in federated cloud infrastructure. The proposed system allows the Data Controller to specify his requirements for data geolocation, enabling the Virtual Execution Environment Manager to place the virtual machines according to these requirements. However, this system only addresses the IaaS layer and does not provide a way to map the data transfers across the different layers. Indeed, data movements at the PaaS and SaaS levels are not detectable by this system. This tool provides services similar to A4Cloud's DTMT tool but does not help in validating accountability tools.

In context of cloud accountability, we can find projects such as, the currently suspended project, CloudAudit[1] which aims at providing the technical foundation to enable transparency and trust in private and public cloud systems. To this end, CloudAudit targets a common interface that allows enterprises and cloud providers to automate the audit, assertion, and assurance of their infrastructure (IaaS), platform (PaaS), and application (SaaS) environments and allow authorized consumers of their services to do likewise via an open, extensible and secure interface and methodology.

2.2.3 Runtime verification

AccMon is a monitoring framework for asynchronous distributed systems which can be (in-line and out-line). AccMon is based on a logic called FO-DTLF (First Order Linear Distributed Temporal Logic over Finite states), which is a mix between LTLFO [BKV15] and DTL [SVAR04]. FOTL is very expressive language however it is an undecidable logic [DFK06]. Just as FOTL, FO-DTLF is not decidable and even not recursively enumerable (due to the use of functions and interpreted predicates). FO-DTLF is more expressive than FOTL since it includes distributed operators.

In [SVA$^+$06] authors present a specific temporal logic `MTTL`, to express properties of asynchronous multi-threaded systems. The proposed monitoring procedure takes as input a formula and a partially ordered execution of a parallel asynchronous system. Then, it determines whether there are executions that violate the formula. However, this procedure is restricted to safety formulas, while AccMon can also monitors liveness formulas (which are the main foucus of AccMon). Another difference is that the authors of [SVA$^+$06] assume a global trace, while we consider the possibility that a system may not collect a global trace.

Other works like [GMM06] target distributed systems, but do not focus on the communication overhead that may be induced by the monitoring. In [SS14], the authors tackle the runtime verification in distributed asynchronous systems. The proposed approach is an extension of PTLTL [SVAR04] and LTL3 [BLS11]. They monitor LTL formula over a distributed system using bushi automaton. While, AccMon monitors formulas with the first order using progression technique. For more details on runtime verification the reader can refer to the surveys [GP10, BB13b].

2.3 Accountability assertions

In this section, we present a notion of accountability assertions that supports the declarative definition of accountability properties enforced by the A4Cloud tools and the environment they

[1] http://cloudaudit.org/CloudAudit/Home.html

are executing in. Accountability assertions are constructed from atomic ones that reify accountability properties that are directly supported by the A4Cloud tools and that can be combined to construct higher-level properties. To this end, we have performed a detailed analysis of the accountability properties of the tools related to policy enforcement, resulting in a complete set of predicates capturing these properties.

We first motivate our notion of accountability assertions in Section 2.3.1. Based on the accountability attributes from workpackage C-2, we then discuss fundamental high-level accountability properties that we would like to express in terms of assertions in Section 2.3.2. In Section 2.3.3 we introduce a general framework for the definition of accountability assertions. General accountability predicates and the tool-specific predicates are presented in Section 2.3.4. Finally, we illustrate in Section 2.3.5 how informal high-level properties can be expressed in terms of low-level ones.

2.3.1 Motivation

Accountability statements often consist of high-level informal properties, for example, that a user being informed about all locations where her data is stored (a transparency property), that a data removal request will be fulfilled after at most a certain time period (a responsiveness property), or that Cloud actors responsible for the manipulation of data can be identified (a responsibility property). Definitions of accountability properties at such a high-level of abstraction are frequently mandatory in order to inform many Cloud actors about accountability-related issues and enable them to make informed decisions concerning their data. This is frequently the case notably for most Cloud subjects that use the Cloud without having any technical background and Cloud auditors that have to assess the general accountability properties of Cloud ecosystems.

However, accountability properties have to be defined, configured, analyzed and enforced at the level of architectures, applications and infrastructures with the help of tools that operate at these levels. This implies that accountability properties have to be expressed frequently using rather low-level existing mechanisms and techniques that ensure security and privacy properties, such as access control, encryption of communication and data stores, and the creation of copies of data bases or virtual machines. Similarly, evidence relevant for determining whether accountability properties are satisfied has to be gathered and expressed in terms of the resources manipulated at the same levels.

Consequently, an approach to the validation of accountability properties that are enforced using tools operating over real software infrastructures should at least enable the definition of accountability properties directly at the level of tools and infrastructures and allow for the composition of those accountability properties in order to express higher-level properties.

Our approach meets these requirements respectively by means of the following concepts:

- A set of high-level accountability properties that are grounded in the accountability attributes defined in WP C-2. They are (informally) defined without reference to tool-specific or implementation-specific properties.
- A set of precisely-defined low-level accountability predicates that make explicit specific properties of tools, applications and infrastructures.
- Means for the definition of the high-level predicates in terms of the low-level ones.

The construction of high-level accountability assertions in terms of accountability predicates is done at two levels:

- Accountability predicates can be composed with one another.
- Accountability predicates can be used as part of validation scenarios (see Chapter 4) that enable complex accountability properties to be defined using a programming-based representation. At this level, predicates and predicate compositions can be orchestrated as part of sequential compositions and loops. Furthermore, additional orchestration operations, such as parallel compositions, can be defined and added to the assertion library harnessed by the Assertion tool.

From a programming language viewpoint this framework satisfies the three following design principles.

- It provides uniform means to capture basic, that is tool-specific and infrastructure-specific, accountability properties.
- it reuses well known mechanisms for the definition of high-level properties: predicate composition, object-oriented abstraction mechanisms and the program-based orchestration.
- it provides a very flexible definition of accountability properties, notably through the simple manner in which new accountability predicates can be defined, in which new complex compositions can be programmed, and the possibility to extend the orchestration possibilities of accountability predicates.

In the remainder of this chapter we discuss, in Section 2.3.2, high-level accountability properties conforming to the accountability model introduced in WP C-2. In Section 2.3.3 we introduce accountability assertions and how they are defined. Finally, in Section 2.3.4, we present the predicates that directly capture the accountability properties of A4Cloud tools for policy enforcement as well as the infrastructure they are executing in.

2.3.2 High-level accountability properties

As part of WP C-2 [FP$^+$14], a general model for accountability (including a set of fundamental accountability attributes) as well as a general accountability framework has been devised. In the remainder of this section, we present examples of high-level accountability properties.

In C-2 ([FP$^+$14], Section 5), the A4Cloud model has been characterized using four key attributes:

- *Transparency:* provide visibility of norms, behavior and compliance of behavior with respect to norms.
- *Responsiveness:* take into account input from external stakeholders and respond to their requests.
- *Responsibility:* individuals and organizations being assigned to take action to comply with norms.
- *Remediability:* take corrective action or provide a remedy in case of failure of compliance with a norm.

These attributes give rise to a number of high-level accountability properties: Table 2.1 shows a list of examples of fundamental high-level properties whose correct handling by the A4Cloud tools we are interested in.

Attribute	**Property**
Transparency	Are the policies governing data storage and processing accessible and modifiable? Where is a Cloud Subject's data located? Where and by whom is a Cloud Subject's data processed?
Responsiveness	Does evidence include the source of the request? Are system components connected? Is an action performed in a timely manner?
Responsibility	Do policies and evidence make explicit responsibility of accountability-related decisions? Can responsibility for data manipulations be tracked?
Remediability	Are Cloud actors correctly notified of accountability issues? Do policy violations result in corrective actions?

Table 2.1: Main high-level properties

Assertion	=	`true` | `false` | *Predicate* | *Task*
Task	=	*Assertion* ; *Assertion*

Table 2.2: Assertion syntax

Predicate	=	*ID* ({ *Subject* | Resource })
	|	`true` | `false` | not *Predicate*
	|	*Predicate* and *Predicate* | *Predicate* or *Predicate*
	|	*TransparencyP* | *ResponsivenessP* | *ResponsabilityP*

Table 2.3: Assertion predicates

2.3.3 Accountability assertions

Accountability assertions constitute one of the core concepts of our approach to the expression and validation of accountability properties. Built from accountability predicates they are the means to express concrete accountability properties of tools and infrastructures. As, essentially, compositions of predicates, they allow complex properties to be expressed and are called as part of validation scenarios that permit to relate accountability properties to tool interactions.

More precisely, accountability assertions, see Table 2.2[2], are built from atomic accountability predicates (non-terminal *Predicate*) that represent basic mostly tool-specific or infrastructure-specific accountability properties. They may also form tasks, that is, sequences of assertions (that are interpreted as conjunctions of assertions).

Accountability predicates, see Table 2.3, generally relate a name (ID) to a boolean expression over subjects and resources. In order to formulate specific accountability properties we provide general, policy, and tool-specific predicates that can be classified to the accountability attributes (Transparency, Responsiveness and Responsability) as introduced in the previous section, see Section 2.3.4.

[2]Here and in the following syntax definitions we use extended Backus Naur form (EBNF) that expresses repetition by curly brackets and optionality by square brackets.

Subject	=	*Actors* | *Tool* | *Env*
Actors	=	`subject` | `consumer` | `provider` | `carrier`
	|	`auditor` | `supervisor` | `controller` | `processor`
Tool	=	`a-pple` | `aas` | `dtmt` | `imt` | ...
Env	=	`OpenStack` | `Service` | `OS`

Table 2.4: Subjects

Resource	=	*ToolRes* | *EnvRes* | *Permission*
ToolRes	=	*PolicyFile* | *DB* | *Evidence* | *Notification* | ...
EnvRes	=	*Location* | *Time* | *OpenStackRes* | ...
OpenStackRes	=	*VM* | *DataVolume* | *Snapshot*
Permission	=	`read(`*Resource*`)` | `write(`*Resource*`)` | `exec(`*Resource*`)`

Table 2.5: Resources

Assertions are defined over operations that are performed by subjects, that are Cloud actors, tools or computational services provided from some (OS or service-oriented) environment, see Table 2.4.

Finally, subjects use and manipulate resources, which include tool resources, environmental resources, such as resources from the underlying OS, as well as permissions, see Table 2.5.

2.3.4 Accountability predicates

Accountability predicates capture basic general, policy or tool-specific accountability properties. In the following these predicates are classified according to the accountability attributes they provide information about.

Some predicates are general in that they are meaningful independent from specific tools and may be used in the context of several tools. This is the case, for instance, for predicates providing information about access of subjects to resources, or predicates that limit the satisfaction of other predicates over time.

However, several predicates are meaningful only when considered in relation to actions by specific tools or a small number of different tools. The accountability predicates we consider have been derived from a detailed analysis of the accountability-specific extensions provided by the policy-enforcing A4Cloud tools: this analysis has resulted in a list of predicates for the A4Cloud tools.

In the following sections, we note predicates that are specific to a particular A4Cloud tool by qualifying the predicate name by the name of the corresponding tool. Finally, accountability predicates may refer to rules established by accountability policies (the predominant case in the A4Cloud project).

Predicates for accountability policies

Accountability properties are defined within the A4Cloud project using policies which are extensions of policies defined as part of the PrimeLife EU project which in turn are extensions of XACML policies [dOSoTP+15b].

Policy	=	*ID* $\mid$ `policy`(*Tool*, *ID*, { *Rule* })	
Rule	=	*ID* $\mid$ `rule`(*ID*, { *Target* }, { *ProvAction* }, *Cond*, { *Obligation* }, *Effect*)	
Target	=	*Subject* $\mid$ *Resource*	
ProvAction	=	`pacReveal`(*Resource*) $\mid$ `pacSign`(*Resource*)	
Obligation	=	*Trigger* : *Action*	
Effect	=	`Permit` $\mid$ `Deny`	
Trigger	=	`trPolicyViolation`(*Policy*)	*A4CLOUD*
	$\mid$	`trDataCollection`(*Subject*, *Resource*)	*A4CLOUD*
	$\mid$	`trComplaint`(*Subject*, *Resource*)	*A4CLOUD*
	$\mid$	`trEvidenceRequestReceived`(*Evidence*)	*A4CLOUD*
	$\mid$	`trDataSubjectAccess`(*Resource*, *Resource*)	*A4CLOUD*
	$\mid$	`trPORRequestReceived`(*Subject*, *Resource*, *Resource*)	*A4CLOUD*
	$\mid$	`trUserRegistration`(*Subject*)	*A4CLOUD*
	$\mid$	`trAtTime`(*Time*, *Duration*)	
	$\mid$	`trPersonalDataAccessedForPurpose`(*Resource*, *Resource*, *Duration*)	
	$\mid$	`trPersonalDataDeleted`(*Resource*, *Duration*)	
	$\mid$	`trPersonalDataSent`(*Resource*, *Resource*, *Duration*)	
	$\mid$	`trDataSubjectAccess`(*Resource*, *Resource*)	
Action	=	`acEncrypt`(*Resource*)	*A4CLOUD*
	$\mid$	`acAudit`(*Policy*)	*A4CLOUD*
	$\mid$	`acEvidenceCollection`(*Evidence*)	*A4CLOUD*
	$\mid$	`acPORetrievability`(*Resource*)	*A4CLOUD*
	$\mid$	`acRequestConsent`(*Subject*, *Resource*)	*A4CLOUD*
	$\mid$	`acDeletePersonalData`(*Resource*)	
	$\mid$	`acAnonymizePersonalData`(*Resource*)	
	$\mid$	`acNotifyDataSubject`(*Resource*, *Resource*)	
	$\mid$	`acLog`(*Policy*)	
	$\mid$	`acEncrLog`(*Policy*)	

Table 2.6: Policies

For the purpose of validation, we define predicates over policies as shown in Table 2.6. This definition capture the main concepts of the A4Cloud policies as used by the A-PPLE and AAS tools. Besides standard XACML authorization decisions (`Permit`, `Deny`), these predicates principally enable the expression of the A4Cloud-specific rules, that is, that use the policy elements marked by *A4CLOUD* in the table. These have been defined as part of the policy extensions for the A4Cloud tools as part of workpackages D-3 and C-8. The remaining entries are defined in the reference documents respectively for XACML and the Primelife Policy Language. The actions *ProvActions*, for example, designate the Primelife actions for the divulgation of additional information about a resource (`pacReveal`) and the assertion of an identity through signing (`pacSign`).

Note that the syntax represents policies as one term starting with the constructor `policy`, which also includes an identifier and the corresponding tool. The A4Cloud tools, A-PPLE in particular, can provide information, for example through logs, about almost all internal processes that can be used for validation purposes. The Assertion Tool therefore allows validation scenarios to be defined in terms of all constituent predicates of policies. It is, for instance, possible to refer to the fact that a policy *Pol* has been violated by directly referring to the predicate

ToolP	`toolInstalled`(*Tool* | *Resource*) `toolConfigured`(*Tool* | *Resource*) `userRegistered`(*Tool* | *Resource*, *Subject*)
PolicyP	`policyInstalled`(*Tool*, *Policy*) `policyInstalled`(*Tool*, *Policy*, *Subject*) `policyInstalled`(*Tool*, *Policy*, *Subject*, *Resource*) `policyAaccessible`(*Tool*, *Policy*) `policyModified`(*Tool*, *Policy*) `policyViolation`(*Tool*, *Policy*)
SubjectResourceP	`login`(*Subject, Resource*) `access`(*Subject, Resource*) `located`(*Subject, Location*) `located`(*Resource, Location*) `exists`(*Resource, Duration*)

Table 2.7: Transparency predicates: atomic predicates

`trPolicyViolation`(*Pol*), or that a data *R* (a resource) has been collected for a subject *S* by referring to the predicate `trDataCollection(S, R)` that are part of the A4Cloud triggers defined for policies in Table 2.6. The trigger `trPolicyViolation`(*Pol*) may be followed by the actions `acEvidenceCollection`(*E*) and `acNotifyDataSubject`(*R1*, *R2*) to respectively reflect the facts that the evidence related to the violation *E* has to be collected and the corresponding Data Subject(s) be notified about the violation that occurred while the resources *R1* and *R2* were processed. This will result in the following obligation:

`trPolicyViolation`(*Pol*): `acEvidenceCollection`(*E*)
and `acNotifyDataSubject`(*R1*, *R2*)

General accountability predicates

Apart from the predicates that constitute policies, a set of general predicates capture accountability-relevant actions by the A4Cloud tools.

Table 2.7 shows the predicates related to transparency properties that provide information about three kinds of entities. First, non-terminal *ToolP* includes three predicates that capture whether tools (accountability tools or other tools, services that are launched as resources from the OS for example) are installed, configured and know about some subject. For instance, the composite predicate

`toolInstalled`(*a-pple*) and `toolConfigured`(*a-pple*)

allows to check whether the tool A-PPLE is installed and configured. Second, non-terminal *PolicyP* derives a set of facts capturing whether policies are installed, accessible and modifiable. For instance the following composite predicate reflects the fact that the policy *Pol* is installed on *a-pple*, and accessible but not modifiable by *aas*:

`policyInstalled`(*a-pple*, *Pol*) and
`policyAaccessible`(*aas*, *Pol*) and
(not `policyModified`(*aas*, *Pol*))

TimlinessP	`period`(*Task*, *Duration*) `before`(*Task*, *Predicate*) `after`(*Task*, *Predicate*)
AvailabilityP	`toolAvailable`(*Tool*) `toolAvailable`(*Tool*, *Function*) `resAvailable`(*Resource*, *Resource*)

Table 2.8: Responsiveness predicates

ResponsabilityP	`userAuthenticated`(*Subject*, *Resource*) `responsibleRecorded`(*Subject*, *Task*)

Table 2.9: Responsability predicates

Policies may impose restriction on arbitrary entities or only for specific subjects and resources as defined by the second and third variant. Third, non-terminal *SubjectResourceP* denotes a set of predicates stating basic facts about subjects and resources. For example, the information about the resource *R* that it is located at the location *L* for a duration *D*, and that it is accessible by the Subject *S* can be expressed by the following composite predicate:

`located`(R, L) and `exists`(R, D) and `access`(S, R)

The second group, shown in Table 2.8, consists of two sets of basic responsiveness predicates. The first set (*TimelinessP*) enables tasks to be limited in time or with respect to (the satisfaction of) other predicates. For example, to enable evidence collection when a policy violation occurs, one can use the predicate:

`after`(`acEvidenceCollection`(*E*), `trPolicyViolation`(*Pol*))

The second one (*AvailabilityP*) features predicates discriminating whether tools or resources are available. The last predicate `resAvailable` is satisfied if a resource is available (linked, connected, reachable, etc.) by another one. It can be used to capture, for example, if no network connection can be established because of a wrongly configured firewall.

The third group, see Table 2.9, consists of responsibility predicates that permit to authenticate users and state that a responsible is recorded for some task.

Tool-specific predicates

The final group of predicates, see Table 2.10, consists of tool-specific predicates. Note that the policy-dependent predicates of the first group are, for a large part, specific to the A-PPLE tool because only A-PPLE harnesses the full policy specification, while some tools, in particular AAS, only support parts of it.

2.3.5 Expression of high-level properties in terms of low-level ones

We are now ready to introduce how (informal) high-level properties are formalized in terms of low-low level ones.

EventP	`event`(*Predicate*) `timestamped`(*Predicate,* Duration) `aas.copyCreated`(*Resource*) `aas.evidenceGathered`(*Predicate*) `aas.encryptionNotEnforced`(*Resource*) `aas.dataRelocated`(*Resource*, *Location*, *Location*) `aas.notificationSent`(*Predicate*)
	`dtmt.dataRelocated`(*Resource*, *Location*, *Location*) `dtmt.acceptedLocation`(*Resource*, *Location*)
	`imt.notificationSent`(*Predicate*)

Table 2.10: Transparency predicates: EventP

The high-level property `Pro`: "by whom a subject's data is deleted" can be expressed by a composition of the following four low-level predicates: the A-PPLE tool detecting data deletion operations (`P1`), the AAS tool constructing the evidence of which actor performed the processing (`P2`) and notifying the IMT tool (`P3`), and the IMT tool sending corresponding notifications. These low-level predicates are applied successively as follows, where `R` is some personal data and D is a certain duration:

```
P1 = trPersonalDataDeleted(R, D)
P2 = aas.evidenceGathered(P1)
P3 = aas.notificationSent(P2)
Pro = imt.notificationSent(P3)
```

The property `Pro` can be generalized to "by whom a subject's data is processed". In this case the predicate `P1` has to be expressed as a disjunction of the low-level predicates that reflect the fact that the A-PPLE tool detecting data processing operations: personal data accessed, deleted or sent (more low-level predicates can be added, if any). In the latter case `P1` is expressed as follows, where *R1* and *R2* are some personal data:

```
P1 =  trPersonalDataAccessedForPurpose(R1, R2, D) or
      trPersonalDataDeleted(R1, D) or
      trPersonalDataSent(R1, R2, D)
```

2.4 Summary

In this chapter, we have introduced the A4Cloud validation methodology, which is based on the runtime verification of accountability assertions and temporal logic formulas. We have provided an overview of the methodology that is realized by two corresponding tools, the AT and the AccMon tool. We have shown how the methodology is related to the metrics developed as part of WP C-5 and introduced the notion of general and tool-specific assertions that enable accountability properties to be expressed by the Assertion Tool. Finally, we have illustrated how (informal) high-level properties and (precisely-defined) low-level ones are linked: the latter are used to express the former.

Chapter 3

The Assertion Framework

The validation methodology presented in the previous chapter is supported by a set of tools and interfaces, the assertion framework. This framework mainly comprises the A4Cloud tools to be validated and two validation tools: the Assertion Tool (AT) and the Accountability Monitor (AccMon).

The AT supports runtime verification of A4Cloud tools through the application of validation scenarios to eco-systems in which the A4Cloud tools (and potentially other tools) are running. Validation scenarios can be used in two ways: either the AT runs the senarios by actively exerting control over the A4Cloud tools in order to execute the A4Cloud tool interactions (REST calls, mainly) that are defined as part of the scenario, or the AT observes the autonomously executing A4Cloud tools by just gathering relevant information. In both cases, the AT verifies accountability properties through the execution of accountability assertions that are also defined as part of the validation scenarios. AccMon supports runtime verification through the distributed monitoring of temporal logic formulae. It intercepts events on a set of machines and validates whether their occurrence satisfies a logical formula or not.

In this chapter we first present the architecture of the assertion framework (Section 3.1), followed by the Assertion Tool (Section 3.2) and the AccMon tool (Section 3.3). In Section 3.4, we give a brief overview of the A4Cloud tools that we have subjected to validation. Finally, Section 3.5 presents a summary.

3.1 Architecture of the assertion framework

In this section, we define the assertion framework. Its main goal is the validation of the A4Cloud tools as a whole, that is, to validate whether the A4Cloud tools are cooperating correctly with each other in order to satisfy accountability properties. The assertion framework is not concerned by the validation of the individual tools (which is part of the responsibility of the tool owners). Note that, not all A4Cloud tools are subjected for validation in D-6, but only those that interact together in an automated fashion and can be associated to (at least) one of the accountability properties (see Section 3.4.1). The tool set that is subjected for the validation called the A4Cloud toolkit.

The architecture of our framework is depicted in Figure 3.1. Its main component of is the Assertion Tool that orchestrates the validation of the A4Cloud toolkit, manages the execution

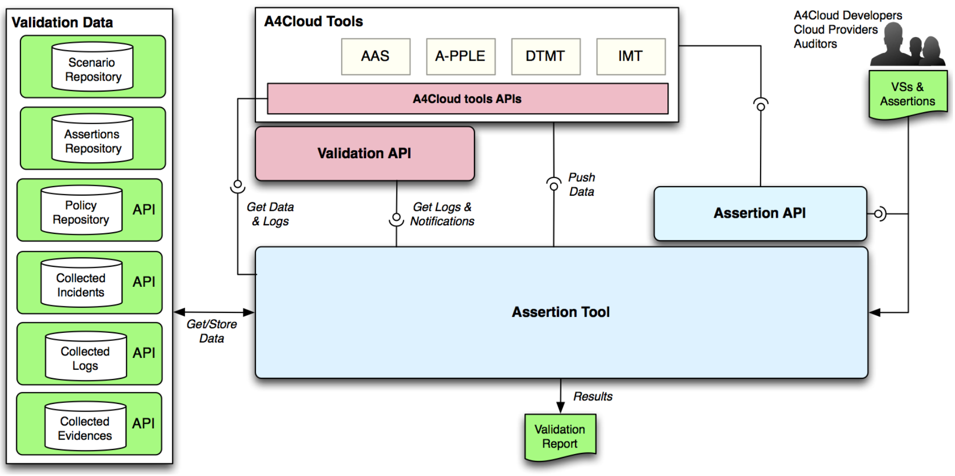


Figure 3.1: Architecture of the assertion framework

of the validation scenarios (VSs), and provides the assertion API, the public interface of the AT offered to tool developers, Cloud providers and Cloud auditors. The assertion API allows human and or software agents to assert that a given A4Cloud toolkit is consistent with the dedicated accountability attributes (i.e., satisfies its associated properties). It provides an abstraction layer allowing tools to be validated in a unified manner, that is using the same set of exposed operations. Note that since the AccMon has not yet been fully integrated with the A4Cloud tools we present it later in Section 3.3.

The validation of the A4Cloud toolkit requires several kinds of data that we call, collectively, `Validation data`. These kinds include definitions of validation scenarios, the definitions of accountability assertions and predicates, policy definitions, as well as runtime data, such as incidents, logs, and evidence. A set of validation scenarios and a set of accountability assertions/predicates are predefined; both may be extended by users of the AT through the assertion API. To the contrary, the other kinds of data are collected at runtime and stored in dedicated repositories by the AT for later use during checking of accountability properties. Concretely, the following kinds of runtime information are harnessed:

- A-PPLE policies which are defined and associated to the validation scenarios.
- Incidents which are collected from the tools, such as the Incident Management Tool, while executing validation scenarios.
- Logs which are collected from the A4Cloud tools, the Audit Agent System in particular, the Data Transfer Monitoring Tool, and the A-PPL Engine.
- Evidence that is generated by tools such as the Audit Agent System.

The runtime data may come from the tools to be validated and/or other sources, such as logs from infrastructure services. They are acquired by the AT by two different means:

1. (Principally public) information, such as installed policies and published logs, through the standard APIs of the A4Cloud tools and other services.

2. Validation-specific information from A4Cloud tools through a an extensible interface, the so-called Validation APIs.

Information may be pushed from the A4Cloud tools to the AT or pulled by the AT. A4Cloud tools push information to the AT using a web server interface that is part of the AT.

In order to control the validation process via the Assertion Tool and manage interactions with the A4Cloud tools, three kinds of API are used: the Assertion, Validation and A4Cloud tools APIs.

Assertion API. This API provides basic services that other tools can employ to communicate with the AT. This API involves the following groups of services:

- **Interaction with other accountability tools.** This subinterface mainly enables the call of functionalities provided through other tools, principally through calls to REST services.
- **Tool declaration and configuration.** This subinterface allows the definition of new accountability tools, the listing of current tools, etc. Such a declaration mainly requires the definition of two entities. First, access methods to the functionality provided by the tools, principally in form of calls to REST services, notably for access to logs and notifications provided by the tool. Second, collector methods that gather runtime information for instance logs.
- **Scenario definition and execution.** This subinterface permits validation scenarios to be loaded, configured and executed. It also provides services for the listing of existing scenarios and their modification.

These interfaces can be used programmatically but some are also accessible to users via the AT's graphical user interface.

Validation API. Validation APIs are offered by some A4Cloud tools in order to provide data that is not publicly offered by the tools but is useful for validation. The validation API may support pushing such information from the external tool to the AT. Furthermore, it can be used to send data in a more secure way, after sanitizing and anonymization. The AAS and IMT tools have been extended by validation interfaces that have been used in concrete validation tasks (see Section 3.4 for info on the validation APIs offered by these two tools).

A4Cloud tools APIs. The AT employs the APIs of the external tools in order to interact with them through REST calls. These interactions include, for example, the installation of A-PPLE policies in order to initialize the execution of a validation scenario or the request of access to a log that is stored within the Transparency Log (TL) tool (see Section 3.4.4.4).

3.2 The Assertion Tool

The Assertion Tool (AT) is the core component of the assertion framework. It enables tool developers and cloud providers to validate the accountability properties of tool combinations, such as A4Cloud toolkit, at runtime using a validation scenario-based methodology.

3.2.1 Features

As part of the validation methodology (see Figure 1.2 on page 13), the AT uses validation scenarios to control or observe the execution of A4Cloud tools and verify accountability properties. It provides the following main features:

Accountability validation scenarios. Validation scenarios can be defined in terms of flexible compositions (essentially orchestrations) of A4Cloud tool actions and accountability assertions. Accountability assertions are built from accountability predicates and executed in order to dynamically verify accountability properties involving multiple tools.

The AT supports the concise definition of accountability scenarios using two features. First, an *automatic mechanism for the application of assertions* enables to automatically execute certain tool-specific assertions based on assertions that express accountability properties involving multiple tools.

Second, the collection of information necessary for validation, such as logs and notifications, is performed internally using so-called information crawlers. Crawlers often have not to be called explicitly as part of validation scenarios because the AT provides *automatic information collection strategies*. These strategies determine when the crawler fetching, for example, the current contents of a log, is called. One of these automatic strategies calls a crawler gathering information from a tool, for example, only when an operation on that tool has been performed.

Two modes for scenario execution. The AT provides two modes of execution of validation scenarios that enable accountability properties to be validated in different ways:

In the *scenario running mode*, the Assertion Tool actively triggers the interactions of the A4Cloud tools as defined in a scenario and also validates the accountability assertions as they are defined in the scenario.

In the *scenario observation mode*, the Assertion Tool observes the interactions of the autonomously executing A4Cloud tools and validates accountability assertions based on available runtime information, such as logs and notifications. In this mode, the AT provides support to automatically update relevant information, such as logs, and to check for the validation of accountability assertions.

Extensible tool set and validation interfaces. The AT includes a *tool interface* that enables the declaration of and interaction with the A4Cloud tools and other external accountability tools. The tool configuration mainly comprises services offered by the tool and the definition of crawlers for accountability-relevant information that is updated regularly.

The tool configuration may also include *validation interfaces* that support the direct, possibly secured and sanitized, communication between the external tool and the AT tool.

3.2.2 Architecture

Figure 3.2 shows the main components of the architecture of the AT that realize the features introduced above. In the following we consider them in more detail.

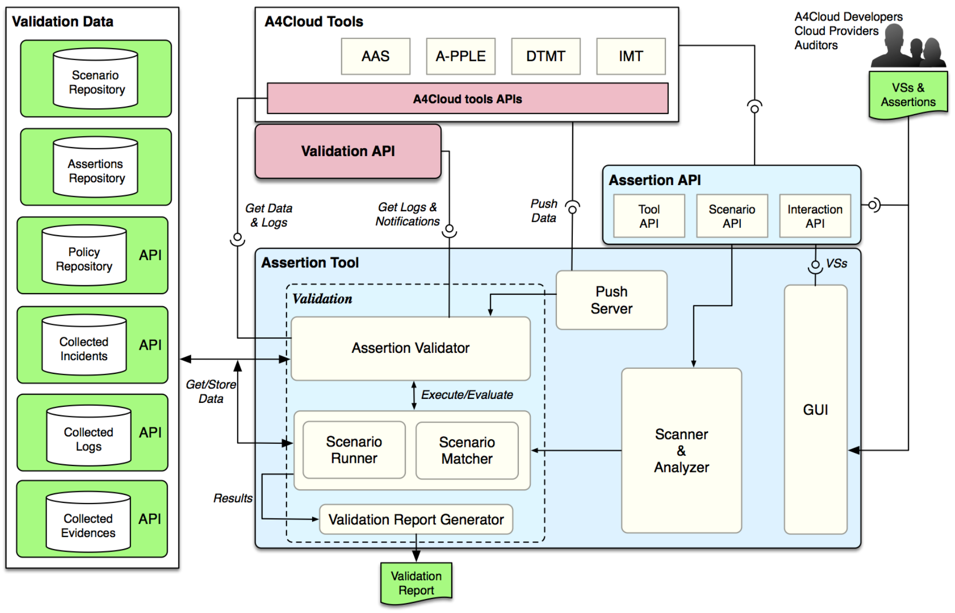


Figure 3.2: Architecture of the Assertion Tool

GUI. A graphical user interface that allows A4Cloud developers and cloud providers to configure accountability tools, select and define validation scenarios and their associated accountability properties/assertions. It also permits to execute validation scenarios according to the different modes.

Scanner & Analyzer. This component scans and interprets validation scenarios provided through the GUI or the assertion API. This component is in charge of transforming the validation scenario that are formulated using a Java-based domain specific language (DSL) into an executable piece of code. It then launches the scenario through the `Scenario Runner` or `Scenario Matcher` components according to whether the mode running mode or matching mode has been chosen.

Scenario Runner. The scenario runner handles the execution of a validation scenario in running mode. This involves executing interactions of the accountability tools (essentially calling services of their REST interfaces) and collecting runtime data according to the configured collection strategy. The scenario runner also interacts with the `Assertion Validator` component to execute the accountability assertions to validate the corresponding accountability properties.

Scenario Matcher. The scenario matcher handles the execution of a validation scenario in matching mode. In this case, the validation scenario does not trigger the interactions with the accountability tools but only observe the execution of the tools and checks the accountability assertions defined in the scenario using the `Assertion Validator` component. In this execution mode, the assertions are checked regularly based on the scenario configuration that may be provided by the user.

Assertion Validator. This component validates the accountability assertions that are part of validation scenarios by analyzing logs, parsing reports, comparing notifications, etc. The corresponding data might come from the validation data repositories which contain the definitions of predicates/assertions as well as data gathered from the running accountability tools. Moreover, information may be gathered from external tools to validate assertions.

Validation Report Generator. The report generator produces validation reports based on the execution state of validation scenarios and the validation of assertions. These reports constitute the output of the AT for a validation request from a user of the tool. The report states whether or not the specified toolkit is validated with respect to the specified properties/predicates and contains details on the execution of the validation scenario. Note that, in running mode a report is provided at the end of scenario execution, while in matching mode reports on the assertion state are provided continuously during a period that can be specified by the user of the AT.

Push Server. The AT implements a REST server which receives information automatically from the A4Cloud tools. The latter can passively interacts with AT by pushing any data which could be relevant to the validation process. The AAS and IMT tools have been extended with this feature which have been used for D-6 Demo. In the next section we present the interfaces offered by the Push Server.

3.2.3 Implementation

The source code of the AT is written in Java (jdk1.8), and the GUI is implemented using HTML/-JavaScript. They can be deployed using a web service in Tomcat 8.

Figure 3.3 shows a detailed (implementation-level) architecture of the Assertion Tool. The web server allows the Assertion Tool interfaces to be accessed remotely through a dedicated GUI. The Loader dynamically loads and compiles the validation scenarios. It implements a dynamic source Java compiler for the GUI. The Scanner provides a concrete representation, based on a Domain Specific Language (DSL), of the abstract validation scenarios provided by the user. The Analyzer mainly generates a term which corresponds to the validation scenario, that is a sequence of tool actions, assertions, and data collections. The Interpreter interprets the term provided by the Analyzer and runs it according to one of the supported execution modes.

To execute a validation scenario the Scanner first parses the validation scenario and provides for each step (tool action, assertion, or evidence collection) an equivalent term. Then the Analyzer analyzes the validation scenario and generates a global term of the form defined by the grammar shown in Figure 3.4 (described below). Finally, the term representing the validation scenario is evaluated following one of the possible execution modes.

In the following, we present the main implementation-level features of the AT.

Runner and matching execution modes. The AT provides two modes of execution: the running mode and matching mode. In the running mode the AT executes the steps, including the

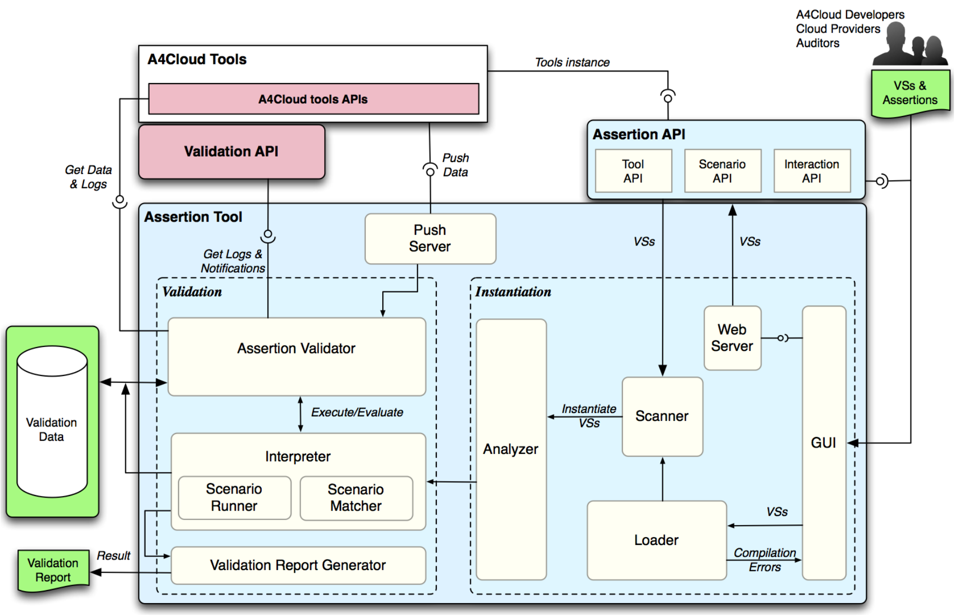


Figure 3.3: Detailed Implementation-level architecture of the Assertion Tool

tool actions and assertions as defined by the provided validation scenario. Assertions are therefore checked only where and when they appear as part of the scenario. In this mode only one validation report is generated.

In the matching mode, the AT runs the provided validation scenario while pushing all the encountered assertions in an assertion queue. As part of the scenario configuration or the system configuration, the user then has to specify the number of times and the time period during which the queued assertions are checked.

Validation scenarios. Validation scenarios are implemented using Java classes that feature a method `scenario` for the scenario definition. The predefined sets of tool interactions and assertions provided by the AT allow the scenarios typically to be defined in a concise and abstract manner.

Listing 3.1 presents, for example, a data retention scenario. In this scenario, the AT first installs policies for the A-PPLE and AAS tools. The A-PPLE policy, as suggested by its name, defines a retention period of information from user Panos of two minutes. The AT than creates an OpenStack snapshot (that violates the policy because it will exist after the retention period), and finally checks whether a retention violation assertion has been recognized and notified.

Validation scenarios are represented internally by terms of the form given by the grammar shown in Figure 3.4. The AT defines a validation scenario as a `ATTerm`. An `ATTerm` can be built based on a set of basic terms: `ATBegin`, `ATCAll`, `ATAssert`, `ATCrawl`, `ATEnd`, that respectively represent REST calls, assertions, data-collection crawlers and return statements. This common representation of a validation scenario allows different interpretations for the same validation scenario, a feature used, for instance, to implement the two scenario execution modes.

Listing 3.1: Data Retention validation Scenario

```
1    public void scenario() {
2      // 1. Create Data Retention Audit Task through AAS
3      aas.createDataRetentionAuditTask();
4
5      // 2. Add a policy to A-PPLE
6      apple.createPII("policies/panosCountryPii2Min.xml");
7
8      // 3. Create OpenStack SnapShot
9      openstack.createSnapshot();
10
11     // 4. Check Retention Violation Assertion
12     ACCOUNT.retentionViolation();
13   }
14 }
```

```
ATTerm   := ATCall ATTerm
          | ATAssert ATTerm
          | ATCrawl ATTerm
          | ATBegin ATTerm
          | ATEnd
```

Figure 3.4: The grammar of ATTerm

Assertion validation. The AT essentially supports two kinds of assertions: the tool-specific ones, and more general ones that are defined in terms of the former. The tool-specific assertions and simple general assertions are predefined and proposed as part of a Java library `ACCOUNT`. In the general case, complex assertions may be defined programmatically. However, complex assertions can frequently be defined as conjunctions of corresponding assertions for the tools involved in a scenario.

For instance, the `ACCOUNT.retentionViolation()` (Line 12 in Listing 3.1) is a complex assertion that is broken down by the AT in assertions for the three tools involved in the scenario:

- An A-PPLE assertion checks whether a data retention policy has been installed and applied.
- An AAS assertion checks whether copies, such as OpenStack snapshots, have been created and exist beyond the retention period, as well as whether a corresponding violation notification has been sent to IMT.
- An IMT assertion that checks whether a notification has been sent to a relevant subject.

In order to support this pattern, the AT checks and executes, if a complex assertion is not defined explicitly, all tool-specific assertions with the same name as the global one. In this

Listing 3.2: The Retention Violation assertion

```
public boolean retentionViolation(Env env) {
  // 1. Get relevant Apple policy  events
  Stream<String> appleEvent = getAppleEvents(env);

  // 2. Identify relevant AAS snapshot information
  Pattern snapIdPattern = getSnapshotId();

  // 3. Get snapshot information
  List<String> snapshotsInfo = getSnapshotInfo(env, snapIdPattern);

  // 4. Get times for deletion policy
  Pattern delTimePattern = getDeletionTime();

  // 5. Check if snapshot exists after deletion time
  boolean res = checkSnapshotExistance(appleEvent,snapIdPattern,
      snapshotsInfo,delTimePattern);

  // 6. Return result
  return res;
}
```

Listing 3.3: The `getAppleEvents` assertion

```
public Stream<String> getAppleEvents(Env env) {

  Stream<String> appleEvent = env
    .of(this)
    .log().stream()
    .filter(s -> s.contains("received_apple_log"));
  return appleEvent;

}
```

case, the Analyzer replaces the general assertion by the corresponding tool-specific ones by modifying the representation of the scenario term, see Figure 3.4.

Furthermore, the AT allows the verification of the assertions during and at the end of the evaluation of validation scenarios in both modes of execution.

As an example of an assertion let us consider Listing 3.2 that shows how a data retention assertion can be defined. It first filters out the relevant A-PPLE policy events, then extracts the information on snapshots obtained from AAS, and the deletion time. Finally, it verifies whether a snapshot has been created during application of the policy and exists beyond the deletion time specified by the policy.

The retention assertion verification executed in five steps involving actions of the A-PPLE and AAS tools. Each step uses a predefined assertion that is part of the `ACCOUNT` package. In the following, we present the implementation of these five assertions.

Listing 3.4: The `getSnapshotId` assertion

```java
  public Pattern getSnapshotId() {

    Pattern snapIdPattern =
      Pattern.compile(".*?NovaImage\\{id=(\\w+).*?"
                      + "<evidenceDetectionTime>"
                      + "(.+)?"
                      + "</evidenceDetectionTime>.*");
   return snapIdPattern;

}
```

Listing 3.5: The `getSnapshotInfo` assertion

```java
  public List<String> getSnapshotInfo(Env env, Pattern snapIdPattern)
      {

    List<String> snapshotsInfo = env
      .of(this)
      .log().stream()
      .filter(s -> !s.contains("received_apple_log"))
      .filter(s -> s.contains("snapshot_detected"))
      .filter(s -> snapInfoPattern.matcher(s).matches())
      .collect(Collectors.toList());
    return snapshotsInfo;

  }
```

Listing 3.6: The `getDeletionTime` assertion

```java
  public Pattern getDeletionTime() {

    Pattern delTimePattern =
      Pattern.compile(".*?PII delete message.*?"
                      + "date: ([\\d-]+) ([\\d:\\.]+).*");
    return delTimePattern;

  }
```

Listing 3.7: The `checkSnapshotExistance` assertion

```
public boolean checkSnapshotExistance(appleEvent,snapIdPattern,
    snapshotsInfo,delTimePattern) {

  return appleEvents.filter(s -> appleEvtPattern.matcher(s).matches
      ())
    .allMatch(evt -> {
        LocalDateTime evtDt = getSnapshotEvtDT(appleEvtPattern, evt)
            ;
        return aasSnapshotsInfo.stream().anyMatch(evd -> {
            LocalDateTime evdDt = getSnapshotEdtDT(snapInfoPattern,
                evd);
            return evdDt.isAfter(evtDt);

}
```

Listing 3.3 presents the `getAppleEvents` assertion which retrieves the A-PPLE events. Listing 3.4 presents the `getSnapshotId` assertion which obtains the snapshots' identities. Listing 3.5 presents the `getSnapshotInfo` assertion which obtains the snapshots' information, based on the identities obtained in the previous step. Listing 3.6 presents the `getDeletionTime` assertion which obtains the deletion time of the snapshots' information. Finally, Listing 3.7 presents the `checkSnapshotExistance` assertion which perform pattern matching in order to check whether a snapshot has been created during beyond the deletion time, based on the information obtained on the previous steps.

Explicit and implicit evidence collection. The AT implements methods which support the explicit and also implicit collection of evidence, such as logs and notifications. The AT supports five different strategies for evidences (e.g., logs, notifications, etc.), namely: `CURRENT`, `TOOL_STEP`, `Scenario_STEP`, `beforeAssert`, `afterCall` The strategy `CURRENT` allows to perform evidence collection explicitly by inserting a corresponding step inside the validation scenario. Using the strategies `TOOL_STEP` and `Scenario_STEP` evidence collection is implicitly performed, respectively, after every tool step and scenario step inside the validation scenario. While using the strategies `beforeAssert` and `afterCall` evidence collection is implicitly performed, respectively, before an assertion verification and after a REST call execution. The `beforeAssert` is in particular important in the matching mode where it is necessary to collect any new evidence before the next verification of the assertions. Note that the AT supports the combination of two or more strategies.

Tool definition. The AT is configured with a set of default tools (A-PPLE, AAS, IMT, and OpenStack), which can be loaded and used to define validation scenarios. However, the user can define new accountability tools and integrate it in the system. AT's tool API provides methods to manage interactions with tools. The tool-defining methods are classified into three groups that are distinguished using the following annotations:

- `@Evidence`: evidence-collecting methods, such as logs and notifications. The evidence annotation takes as an argument a set of evidence-gathering strategies, as described in

Listing 3.8: AAS tool Java implementation

```
public class AASTool {
  //Evidence
  @Evidence(strategy = {CURRENT, TOOL_STEP})
  void getLog(Env env) {...}
  @Evidence(beforeAssert = "successfulAccess")
  void getLogAfter10s(Env env) {...}

  // REST Calls
  @Call
  JSONObject createIntrusionDetectionAuditTask(args)
  { ... }
  @Call
  JSONObject createDataRetentionAuditTask(args) {...}

  // Assertions
  @Assert
  boolean successfulAccess(args, env) {...}
  @Assert
  boolean retentionViolation(args,env) {...}

  // Utils
  String getIntrusionEvtId(Pattern p, String event) {...}
  LocalDateTime getSnapshotEvtDT(p, event) {...}
  LocalDateTime getSnapshotEdtDT(p, evidence) {...}
}
```

the previous paragraph.

- `@Call`: methods that allow the AT to perform REST calls on a tool.
- `@Assert`: methods defining tool-specific assertions.

Listing 3.8 shows how the AAS tool is defined. The AASTool class is defined including methods for evidence collection to get logs, two REST calls which permit to create audit tasks at AAS tool, two assertions `successfulAccess` and `retentionViolation`, and a set of utility methods to facilitate the assertion definitions. The `successfulAccess` assertion verifies whether an *EvidenceRecord* contains all relevant A-PPLE events to accesses of a users' private data. This assertion is used, in particular, in the intrusion detection validation scenario, see Section 4.2.1. The `retentionViolation` assertion verifies whether there is any information related to a certain snapshot after a certain point of time. It is used, in particular, in the data retention scenario, see Section 4.2.2.

Interaction with other tools. The AT interacts with the A4cloud tools by performing calls to the REST webservices of the tools in order to trigger some actions or collect some validation data.

Listing 3.9 presents an example of a REST call by AT to AAS. This REST call initializes the AAS with a data retention audit task.

Listing 3.9: REST call to AAS

```java
public JSONObject assCreateDataRetentionAuditTask(JSONObject args) {
   JSONObject res = new JSONObject();
   String s = Unirest.get(AAS_URL)
        .routeParam("service", "setAuditTask")
        .header("accept", "application/json")
        .queryString(JSONHelper.toMap(args))
        .asString()
        .getBody();
   res = new JSONObject("{" + REST_KEY + ":" + s + "}");
   return res;
 }
```

API Name	Purpose of use	Consumed by	Data
Interaction API	Instantiate and execute validation scenarios. Re-/load, list assertions.	AT user interface	JSON
Scenario API	Re/load, list validation scenarios	AT user interface	JSON
Tool API	Re/load, list tool definitions	AT user interface	JSON
Validation API	Push data for validation purposes (notifications . . .)	AAS, IMT tools	JSON

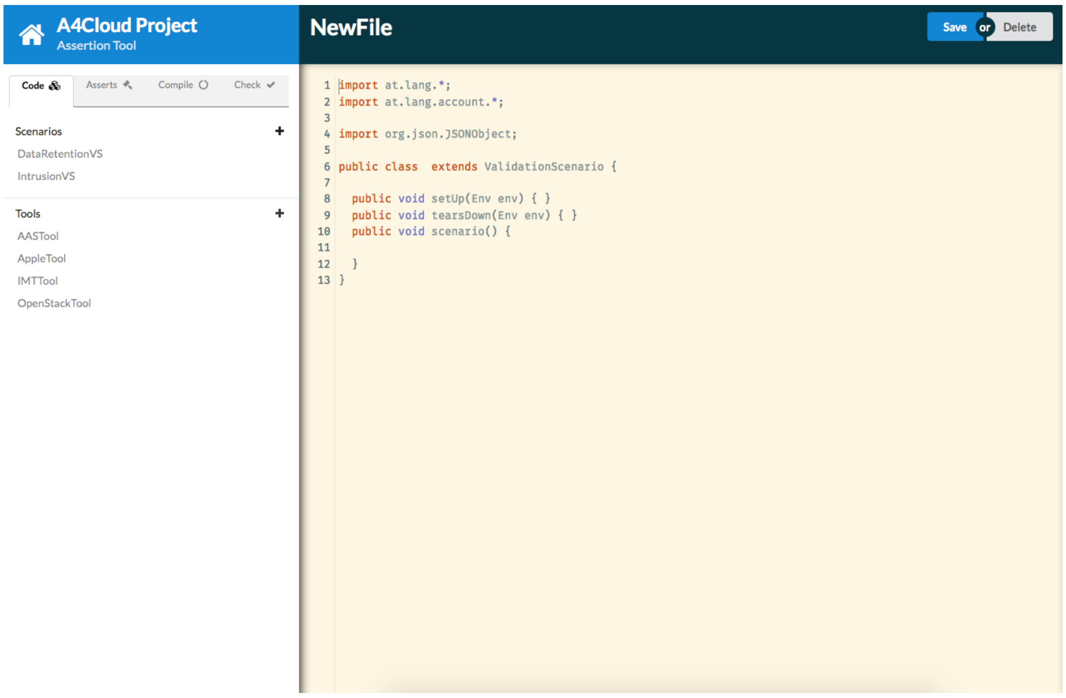
Table 3.1: REST interfaces offered by the Assertion Tool

AT's REST interface. The AT offers a set of APIs, collectively called the Assertion API. This API offers four subinterfaces: the Interaction, Tool, Scenario, and Push Server subinterfaces. Details of these subinterfaces can be found in Table 3.2.3.

GUI. The GUI implements the assertion API. A user can use the GUI to directly load and execute the configured validation scenarios following the possible execution modes, as well as, to get the result displayed through the GUI. A user can also define new tools and validation scenarios through the GUI.

Figure 3.5 shows a screenshot of the GUI. From the left-hand side a user can load already-defined tools, assertions, and validation scenarios (from "Code" tab). It can also execute the validation scenarios following either the running mode or the matching mode (from "Check" tab). After a validation scenario execution, the validation result are displayed: the number of assertions checked, the assertions that have succeeded, and those that have failed along with the reasons of failure. Using the buttons at the top a user can create new entities, save or delete them. The screenshot depicted in Figure 3.5 shows the GUI while displaying an empty structure of a validation scenario together with the related headers. Such a scenario's default structure can be loaded by the "+" symbol that is beside the scenario list. Similarly, a default structure for a tool definition can be loaded by the "+" symbol that is beside the tool list. Note that the assertion list (which is not appear in the screenshot presented in Figure 3.5) is under the "Assert" tab.

Figures 3.6 and 3.7 respectively show a screenshot of the GUI displaying the already defined Data Retention validation scenario and A-PPLE tool.



```
import at.lang.*;
import at.lang.account.*;

import org.json.JSONObject;

public class  extends ValidationScenario {

  public void setUp(Env env) { }
  public void tearsDown(Env env) { }
  public void scenario() {

  }
}
```

Figure 3.5: Screenshot of the GUI

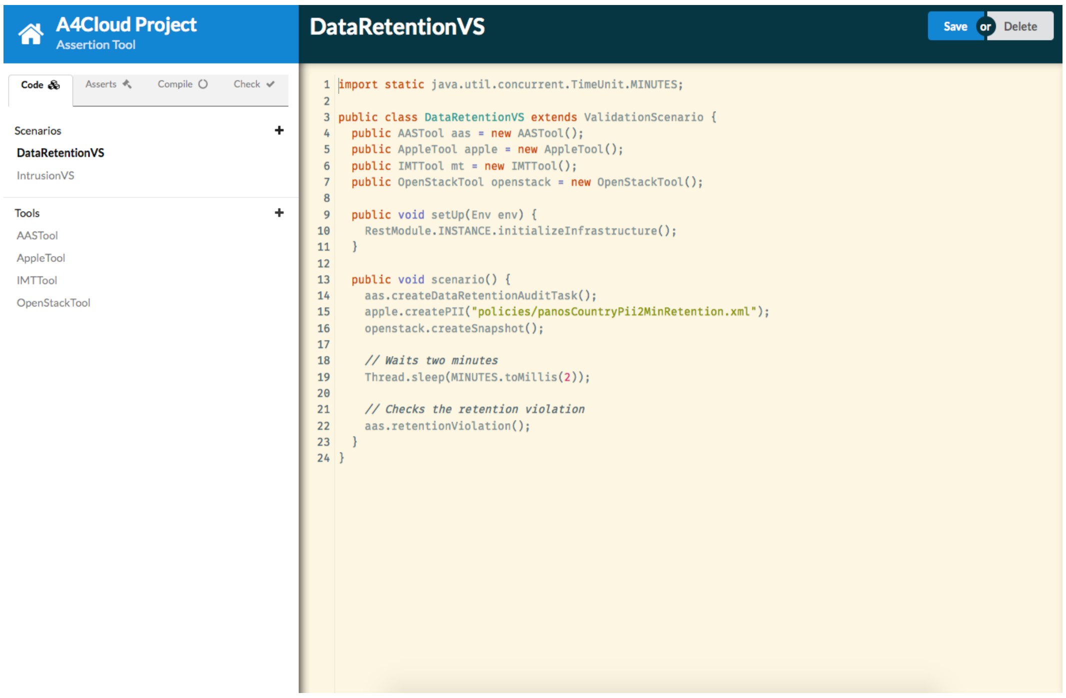
```
# Add a rule on HTTP request
Sysmon.add_http_rule("Rule1", control_type=Monitor.MonControlType.REAL_TIME
    ,
"G( ![req:GET]( USER('bob') => ~ReqIn(r\"taskManager/profile/\", req) )" )
```

3.3 AccMon: Accountability Monitor

3.3.1 Overview

The goal of AccMon is to provide means to monitor accountability policies in the context of a real system. AccMon allows to specify policies that are applicable to network traffic, web application code and external components via plugins. These policies are based on a distributed temporal logic with data. The framework allows centralized or distributed monitoring and both on-line and off-line controls.

In the following is an example of monitoring a rule ”user bob should not access to the URL ’taskManager/profile/’ ” (details are provided in next sections): The rule means: Always, forall GET requests if the logged user is bob then the url requested by bob should not starts with taskManager/profile/.



```java
import static java.util.concurrent.TimeUnit.MINUTES;

public class DataRetentionVS extends ValidationScenario {
  public AASTool aas = new AASTool();
  public AppleTool apple = new AppleTool();
  public IMTTool mt = new IMTTool();
  public OpenStackTool openstack = new OpenStackTool();

  public void setUp(Env env) {
    RestModule.INSTANCE.initializeInfrastructure();
  }

  public void scenario() {
    aas.createDataRetentionAuditTask();
    apple.createPII("policies/panosCountryPii2MinRetention.xml");
    openstack.createSnapshot();

    // Waits two minutes
    Thread.sleep(MINUTES.toMillis(2));

    // Checks the retention violation
    aas.retentionViolation();
  }
}
```

Figure 3.6: Screenshot shows the data retention scenario

3.3.2 Architecture

As the Figure 3.8 shows, AccMon acts as a middleware in the framework, it intercepts and logs client's HTTP requests, server's requests processing and responses. On the web application side the developer can configure the framework to intercept function/method calls and databases access. AccMon can acts as a daemon and can be interconnected with external tools, the property to monitor on the tool is defined in AccMon and the tool sends log events to AccMon via HTTP calls. The framework can be also connected with external hardware such as `Arduino`[1] electronic boards.

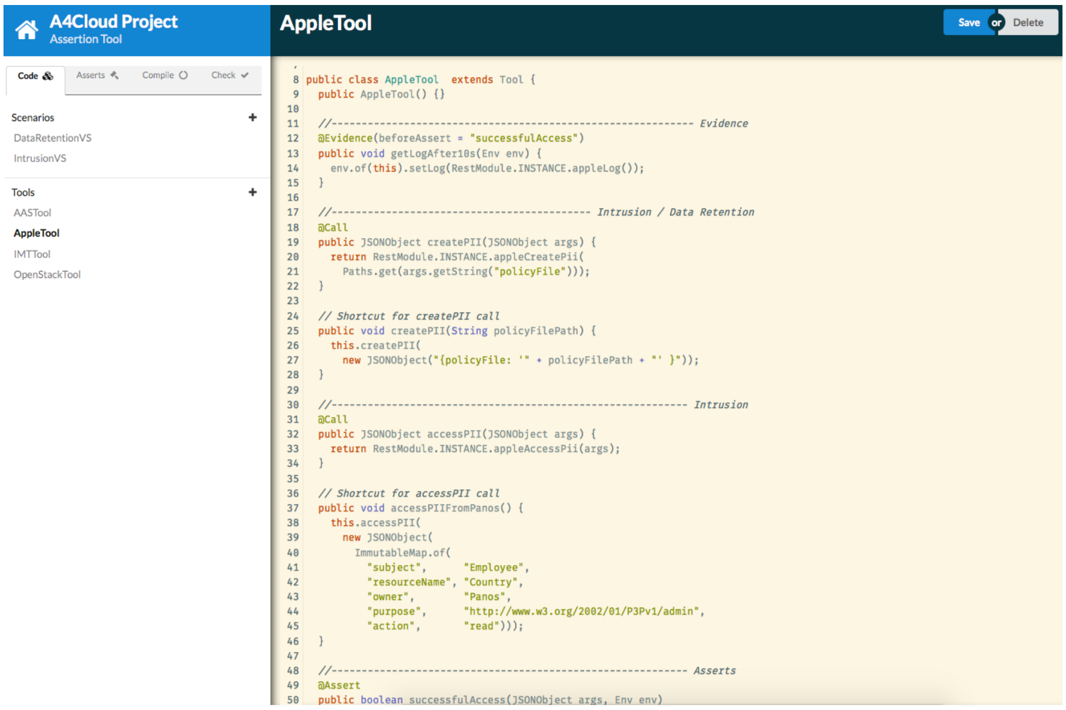
The Figure 3.9 shows AccMon internal architecture which is composed with:

1. *Monitors:* a monitor is defined by an id, a Fodtl formula[2], a control type (posteriori or realtime), and optionally a remediation formula which represent the formula to monitor in case of violation of the rule.

    - $add_{<}http|view|response > \_rule$: the target group for the monitor, it can be when a request is received, when the response is computed, or when a response is returned

[1] Arduino is an open-source prototyping platform based on easy -to-use hardware and software. for more details see https://www.arduino.cc/

[2] For First-Order Distributed Temporal Logic, this is the formal language supported by AccMon.



```java
public class AppleTool  extends Tool {
  public AppleTool() {}

  //------------------------------------------------------------ Evidence
  @Evidence(beforeAssert = "successfulAccess")
  public void getLogAfter10s(Env env) {
    env.of(this).setLog(RestModule.INSTANCE.appleLog());
  }

  //------------------------------------------ Intrusion / Data Retention
  @Call
  public JSONObject createPII(JSONObject args) {
    return RestModule.INSTANCE.appleCreatePii(
      Paths.get(args.getString("policyFile")));
  }

  // Shortcut for createPII call
  public void createPII(String policyFilePath) {
    this.createPII(
      new JSONObject("{policyFile: '" + policyFilePath + "' }"));
  }

  //----------------------------------------------------------- Intrusion
  @Call
  public JSONObject accessPII(JSONObject args) {
    return RestModule.INSTANCE.appleAccessPii(args);
  }

  // Shortcut for accessPII call
  public void accessPIIFromPanos() {
    this.accessPII(
      new JSONObject(
        ImmutableMap.of(
          "subject",      "Employee",
          "resourceName", "Country",
          "owner",        "Panos",
          "purpose",      "http://www.w3.org/2002/01/P3Pv1/admin",
          "action",       "read")));
  }

  //------------------------------------------------------------ Asserts
  @Assert
  public boolean successfulAccess(JSONObject args, Env env)
```

Figure 3.7: Screenshot shows A-PPLE tool

```
# Add a rule (monitor) in the system
Sysmon.add_<http|view|response>_rule(<name>, <Foddtl_formula>, description=
    "",
        control_type=Monitor.MonControlType. <REAL_TIME|POSTERIORI>,
  violation_formula=<V_Fodtl_formula> )
```

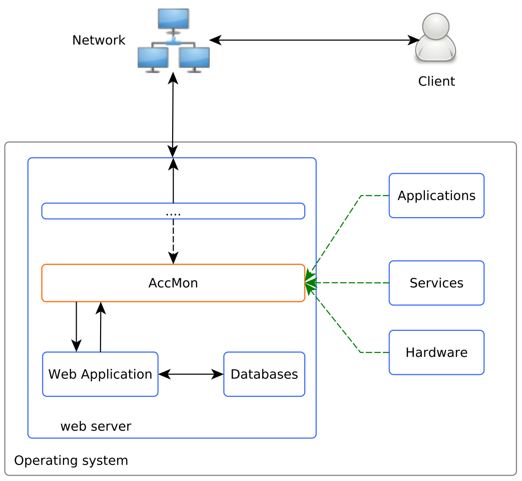


Figure 3.8: AccMon global architecture

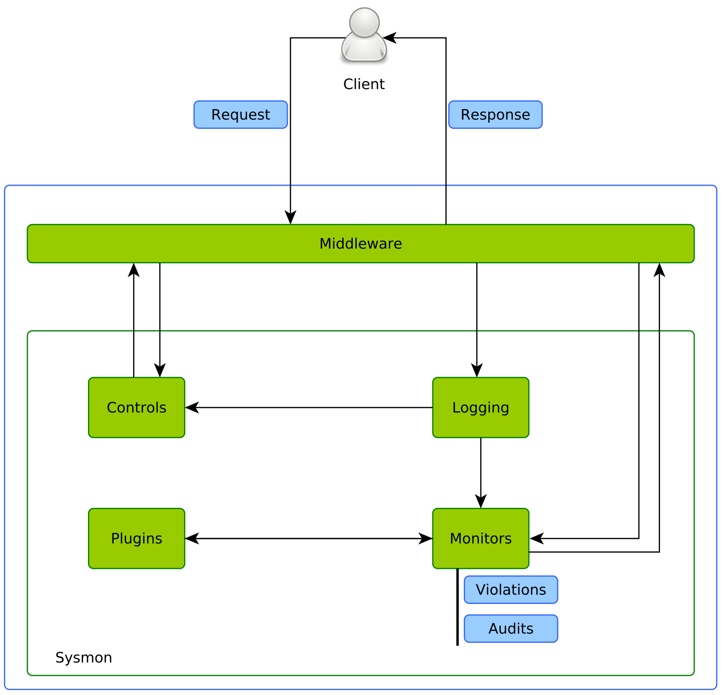


Figure 3.9: AccMon internal architecture

to the client.

- $< Fodtl\_formula >$: a first order distributed temporal logic formula which corre-

```
# Logging attribute evaluation function
# In this example we log the request scheme
def fx(request, view, args, kwargs, response):
       return P("SCHEME", args=[Constant(request.scheme)])

# Creating a logging attribute
lg = LogAttribute(<name>, description="...", enabled=<True|False>, eval_fx=
     fx)

# Adding the logging attribute to the system
Sysmon.add_log_attribute(lg, target=Monitor.MonType.<HTTP|VIEW|RESPONSE>)
```

```
class XSS(Control):
       def prepare(self, request, view, args, kwargs):
              data = getattr(request, request.method)
       for key in data:
              mutable = data._mutable
           data._mutable = True
           data[key] = sanitize(data.get(key))
           data._mutable = mutable
```

spond to the property to monitor.

- $< REAL\_TIME|POSTERIORI >$: the behavior of the monitor, in real time mode the monitoring is a blocking operation and when a violation occurs it blocks the current request. In the posteriori mode the monitoring operation is delayed and when a violation occurs it will be just reported.
- $< V\_Fodtl\_formula >$: the formula to monitor when this rule is violated.

2. *Logging:* a log attribute is defined by a name, a description and an evaluation function that takes five arguments (request, view, args, kwargs, response) and which will be called automatically by the framework if the logging attribute is enabled.

3. *Controls:* defines security checks that can be extended by the developer. Below an example of an input sanitizer in order to prevent some XSS attacks:

4. *Plugins:* connect AccMon with external components which allows to define monitors that monitor formulas for external softwares/ hardware in AccMon. The framework expose an interface in order to receive events from external components.

3.3.3 Implementation

The implementation of AccMon framework is based on Django which is an open-source web application framework written in Python and it is based on the model-view-controller (MVC) pattern.

```
# The evaluation function of the log attribute
# Here we return a predicate that looks like: USER('user_name')
def username(request, view, args, kwargs, response):
       return P("USER", args=[Constant(request.user)])

# Create the log attribute
username_log = LogAttribute("UserName", enabled=True, eval_fx=username)

# Register the attribute in the monitoring system and make it available
# on HTTP request level.
Sysmon.add_log_attribute(username_log, target=Monitor.MonType.HTTP)
```

```
class UserEq(IPredicate):
   # Compare the username of a user for a given id with a given username.
   # Usage in a formula: UserEq(usser_id, user_name)
   def eval(self, valuation=None):
       args2 = super().eval(valuation=valuation)
       u1 = User.objects.filter(id=args2[0].name).first()
       return u1.usernmae == args2[1].name
```

```
# Add a rule on HTTP request
Sysmon.add_http_rule("UserProfile", control_type=Monitor.MonControlType.
    REAL_TIME,
"G( ![id:UIDL uname:USER req:GET]( ReqIn(r\"taskManager/profile/\", req) =>
 UserEq(id, uname)) )" )
```

In the following we presents the steps to use our framework. First we need to define what we want to log, AccMon comes with predefined logging attributes that can be enabled and disabled at runtime. The developer can also define custom attribute to log. Here a basic example:

Next the developer can define custom interpreted predicates and functions that will be used in the monitoring formulas (this feature is provided by FodtlMon [3]). Here an example that checks if the corresponding user (Django user) of a given id has a certain username.

Finally the developer defines monitoring rules using Fodtl formulas. Here some examples:

- Add a rule on HTTP request:
- System interfacing example: In this example each time the user uses the change directory 'cd' command of the operating system, the system sends the event with the path to AccMon.
- Hardware interfacing example (Arduino): In this example we built a laser detector using an Arduino board, a laser diode and a light sensor. Basically the laser diode points into the light sensor which sends to the arduino board a value of the light intensity. [4] The code on

[3]FodtlMon is the fodtl monitoring engine.

[4]The AccMon arduino plugin read the events from the serial port and append them to arduino monitors traces.

```
# Check if the path is not in /root/*
remote.Remote.add_rule("cdroot", "G( ![path:cd]( ~Regex(path, r\"/root/*\")
    ) )")
```

```
# The light intensity should be always greater than 10
arduino.Arduino.add_rule("light", "G( ![x:LIGHT]( Gt(x, '10') ))")
```

```
DataRetentionExpire => (Forall x:snapshot snapshot_deleted(x) AND
  At(backupStorage, Future(Forall x:snapshot snapshot_deleted(x) )) )
```

the arduino board (written in C language) simply reads values from the light sensor and print them in the serial port, in the form of a predicate `LIGHT(value)`.

3.3.4 AccMon collaboration with the Assertion Tool

AccMon provides unique features and can also be used in conjunction with the Assertion Tool. The main usage of AccMon is for monitoring a distributed system and a second serves to check execution traces.

AccMon is able to monitor distributed systems from a property written in a Fodtl, but this property can be a security, accountability, or correctness property. The main way to use AccMon is to use it as a standalone tool for centralized or distributed systems monitoring. AccMon is based on first-order temporal logic with predicates. To interact with AccMon the monitored agents needs to be instrumented to generate events. An event is an instantiation of a predicate with some concrete values and it denotes the agent processing an action with the required parameters. Once the scenario to evaluate has been (manually) translated into the logic accepted by AccMon, the AccMon engine is able to generate monitors for each of the agents involved in the system. The language accepted by AccMon is based on first-order temporal logic, like AccLab, however there are several differences since AccMon can explicitly handle distribution.

Once agents are equipped with the adequate interfaces then AccMon is able to produce a set of distributed agents. For instance let us consider the scenario:
A service provider `csp` using a backup storage service `backupStorage`. The `csp` want to monitor that all snapshots are deleted after the retention period expires. Having AccMon deployed on the both side, `csp` will monitor the following formula: The formula represent the following property: when data retention period expires, all snapshots in `csp` should be deleted and at `backupStorage` service in the future all snapshots should be deleted.

Generally we will have more distributed agents and using a central controller as actually did by the Assertion Tool is not scalable. Thus AccMon provide a scalable solution here complementary to the Assertion Tool. However, the main drawback is the formula writing, which is currently not user-friendly, while it can be improved in the same way it has been done with the AccLab tool. The Assertion Tool provides an more classic input solution which is more suitable for Java programmers.

It is possible to use AccMon in collaboration with the Assertion Tool to validate execution traces against a behavioral formula. In order to reuse the Assertion Tool's validation scenarios

Listing 3.10: Fodtl formulas for AccMon

```
# intrusion detection with threshold=5
Future (apple.accessPIIFromPanos() AND
    (Future (apple.accessPIIFromPanos() AND
          (Future (apple.accessPIIFromPanos() AND
               (Future (apple.accessPIIFromPanos() AND
                      (Future apple.accessPIIFromPanos()))))))))
    => ass.triggerViolation("panos","access","threshold")
```

they have first to be translated manually (but an automated translation seems possible) into a first-order temporal formula.

The Listing above 3.10 describes, for instance, a Fodtl formula for the intrusion detection scenario.

On one hand, the Assertion Tool needs to implement the testing scenario which is done in a manner rather familiar to Java programmers. On the other hand, AccMon do not need to program something but takes as input a logical formula which is an approach more dedicated to formal specifier or user of model-checking tools. There is no silver bullet here but two tools providing different inputs for distinct end users.

Once these formulas are defined, the FodtlMon module, an essential part of AccMon, is able to analyze a global execution trace and to check if it violates the formula or not. There is a point to clarify here, we consider that we have dispose of a global time which is possible under some runtime hypotheses. Otherwise, we need to use logic global time for distributed system defined, for instance, using vector clocks [Mat89, Fid88].

The FodtlMon module takes a formula and a finite global trace, evaluates the events one by one and generates three possible outputs:

- `True`: the trace never violates the formula
- `False`: the trace has violated the formula
- `?`: there is not violation until now but the future is not determined

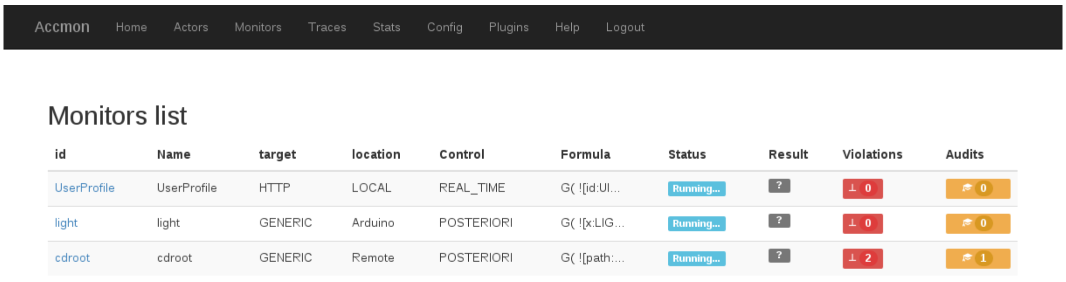
In fact in the last case it produces a future formula expressing what could happen if we add new events to the current trace. The reader should note that generally AccMon will provide a finer analysis than the Assertion Tool and this impacts greatly the performance of the checking mainly in the critical case of online testing. The important fact here is that the performance of AccMon is independent of the length of the trace to check and it only depends on the complexity of the formula. This is different with the Assertion Tool which will analyze again the complete trace each time new events are arriving.

Let us consider a property like PROP="always writing Kim's data implies that a notification is send to Kim". Consider a trace starting with a writing event followed by one million of `anyother` event. The Assertion Tool sees only the writing event in the current trace thus it will conclude that the property is false. But if the next following event is a notification then the tool will read again the full trace to conclude that the property is satisfied until now. The situation is different with AccMon since the property will be rewritten after the writing event in something like "waiting for a notification" and PROP. The `anyother` events do not rewrite the formula and when the notification event arrives the property is rewritten as PROP meaning that this is the future property to check. This improvement is critical in online testing since the trace is growing and of course the time to analyze it.

To conclude AccMon allows monitoring of distributed systems as soon as the agents are equipped with adequate interfaces. It is also complementary to the Assertion Tool and can be used with it to validate concrete scenarios of execution and then help to test the A4Cloud tools. Nevertheless AccMon relies on complex input formulas but it provides a better support for online testing.

3.3.5 Screenshots

Figure 3.10 shows the list of monitors that are running in AccMon.



Monitors list

id	Name	target	location	Control	Formula	Status	Result	Violations	Audits
UserProfile	UserProfile	HTTP	LOCAL	REAL_TIME	G(![id:UI...	Running...	?	⊥ 0	0
light	light	GENERIC	Arduino	POSTERIORI	G(![x:LIG...	Running...	?	⊥ 0	0
cdroot	cdroot	GENERIC	Remote	POSTERIORI	G(![path:...	Running...	?	⊥ 2	1

Figure 3.10: AccMon monitors overview

Figure 3.11 shows the events collected by AccMon on the HTTP requests. Figure 3.12 shows the Assertion Tool plugin for AccMon. The plugin allows to configure which attributes to log from APPLE and AAS logs.

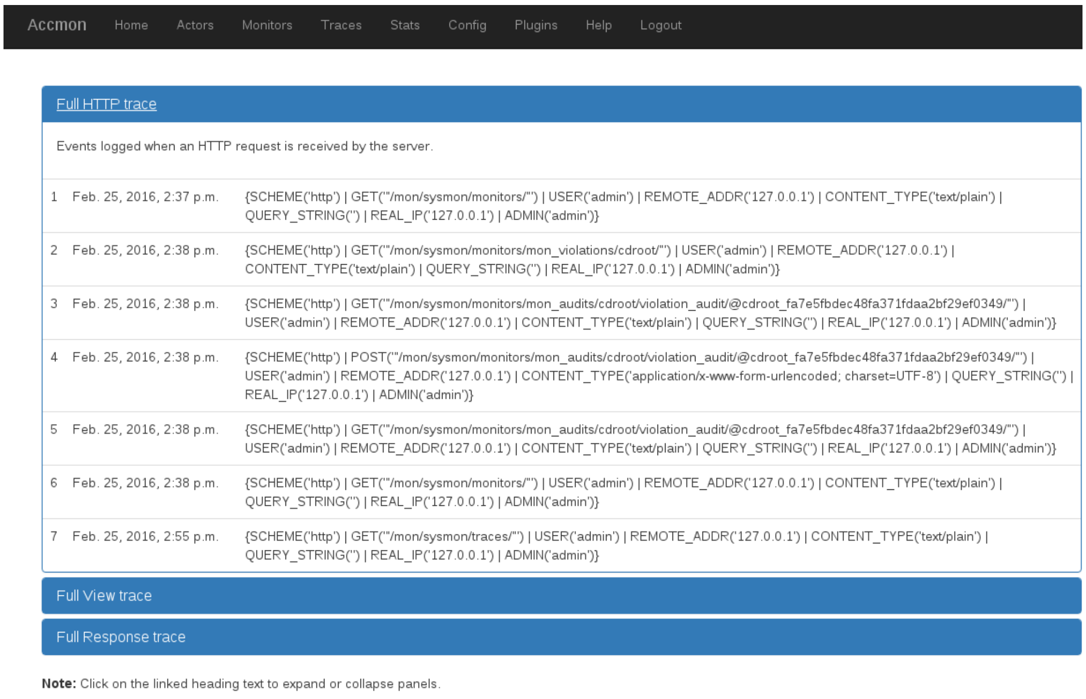
3.4 A4Cloud toolkit

In WP D-2, the A4Cloud partners conducted two internal surveys on A4Cloud tools and proposed a high level functional architecture of the A4Cloud tools [GKP$^+$15a]. In this architecture (see Figure 3.13), the 12 A4Cloud tools are grouped according to their main functionality into five areas.

In the following, we present the A4Cloud tools that are part of the validation toolkit, that is, that we have subjected to validation.

3.4.1 Tool set subjected for validation

The A4Cloud toolkit considered in D-6 for validation constitutes the “main input” for the assertion framework. Some of the tools, such as the contract and risk management tools are intended to be used manually as advisory tools, while others can be used in an automated fashion as part of Cloud ecosystems and infrastructures. Among these tools the accountability-enforcing tools, such as the Accountability Primelife Policy Engine (A-PPLE), the Audit Agent System (AAS), and the Incident Management Tool (IMT), play a particular role because of the variety, volume



Events logged when an HTTP request is received by the server.

```
1  Feb. 25, 2016, 2:37 p.m.   {SCHEME('http') | GET('"/mon/sysmon/monitors/"') | USER('admin') | REMOTE_ADDR('127.0.0.1') | CONTENT_TYPE('text/plain') |
                              QUERY_STRING('') | REAL_IP('127.0.0.1') | ADMIN('admin')}
2  Feb. 25, 2016, 2:38 p.m.   {SCHEME('http') | GET('"/mon/sysmon/monitors/mon_violations/cdroot/"') | USER('admin') | REMOTE_ADDR('127.0.0.1') |
                              CONTENT_TYPE('text/plain') | QUERY_STRING('') | REAL_IP('127.0.0.1') | ADMIN('admin')}
3  Feb. 25, 2016, 2:38 p.m.   {SCHEME('http') | GET('"/mon/sysmon/monitors/mon_audits/cdroot/violation_audit/@cdroot_fa7e5fbdec48fa371fdaa2bf29ef0349/"') |
                              USER('admin') | REMOTE_ADDR('127.0.0.1') | CONTENT_TYPE('text/plain') | QUERY_STRING('') | REAL_IP('127.0.0.1') | ADMIN('admin')}
4  Feb. 25, 2016, 2:38 p.m.   {SCHEME('http') | POST('"/mon/sysmon/monitors/mon_audits/cdroot/violation_audit/@cdroot_fa7e5fbdec48fa371fdaa2bf29ef0349/"') |
                              USER('admin') | REMOTE_ADDR('127.0.0.1') | CONTENT_TYPE('application/x-www-form-urlencoded; charset=UTF-8') | QUERY_STRING('') |
                              REAL_IP('127.0.0.1') | ADMIN('admin')}
5  Feb. 25, 2016, 2:38 p.m.   {SCHEME('http') | GET('"/mon/sysmon/monitors/mon_audits/cdroot/violation_audit/@cdroot_fa7e5fbdec48fa371fdaa2bf29ef0349/"') |
                              USER('admin') | REMOTE_ADDR('127.0.0.1') | CONTENT_TYPE('text/plain') | QUERY_STRING('') | REAL_IP('127.0.0.1') | ADMIN('admin')}
6  Feb. 25, 2016, 2:38 p.m.   {SCHEME('http') | GET('"/mon/sysmon/monitors/"') | USER('admin') | REMOTE_ADDR('127.0.0.1') | CONTENT_TYPE('text/plain') |
                              QUERY_STRING('') | REAL_IP('127.0.0.1') | ADMIN('admin')}
7  Feb. 25, 2016, 2:55 p.m.   {SCHEME('http') | GET('"/mon/sysmon/traces/"') | USER('admin') | REMOTE_ADDR('127.0.0.1') | CONTENT_TYPE('text/plain') |
                              QUERY_STRING('') | REAL_IP('127.0.0.1') | ADMIN('admin')}
```

Note: Click on the linked heading text to expand or collapse panels.

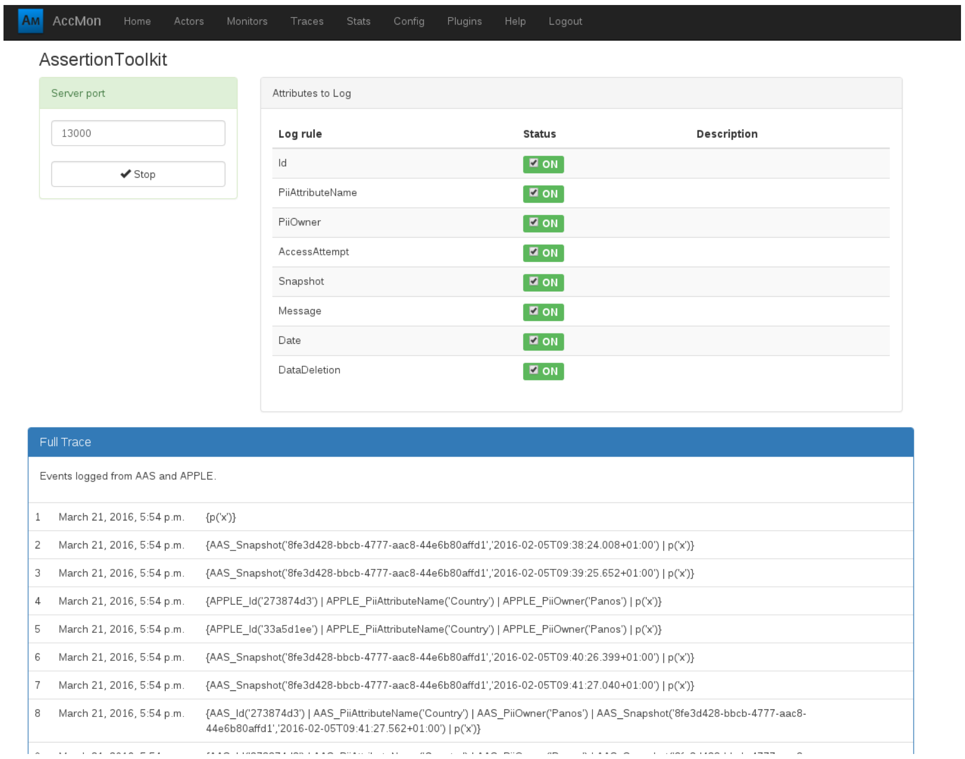
Figure 3.11: AccMon traces

and complexity of interactions between Cloud actors and their data. These interactions require that accountability tasks are automated as far as possible.

The A4Cloud tools achieve a high degree of automation, all the while being able to enforce diverse accountability properties at very different levels of Cloud architectures. They encompass high-level properties, such as a data subject knows about all locations at which her data is stored or intrusion attempts in access-controled data sets of Cloud users. They also have to satisfy low-level properties, such as the identification of sensitive data in virtual machine snapshots at the infrastructure level.

Due to the diversity of the tools, the interactions between actors and the data involved, the correct enforcement of accountability properties by the A4Cloud tools is difficult to be validated. After performing a study on the tools, we found that the A4Cloud tools that might be particularly interesting for the validation process are: the Accountable-PPL Engine (A-PPLE), the Audit Agent System (AAS), the Data Transfer Monitoring Tool (DTMT), the Incident Management Tool (IMT), and the Transparency Log (TL). These tools constitute the A4Cloud toolkit that is considered in D-6 for validation.

In the remainder of this section, we provide an overview of the interactions between these tools (Section 3.4.2), the corresponding data flows (Section 3.4.3), and a brief description for each tool (Section 3.4.4).



AssertionToolkit

Server port

13000

Attributes to Log

Log rule	Status	Description
Id	✓ ON	
PiiAttributeName	✓ ON	
PiiOwner	✓ ON	
AccessAttempt	✓ ON	
Snapshot	✓ ON	
Message	✓ ON	
Date	✓ ON	
DataDeletion	✓ ON	

Full Trace

Events logged from AAS and APPLE.

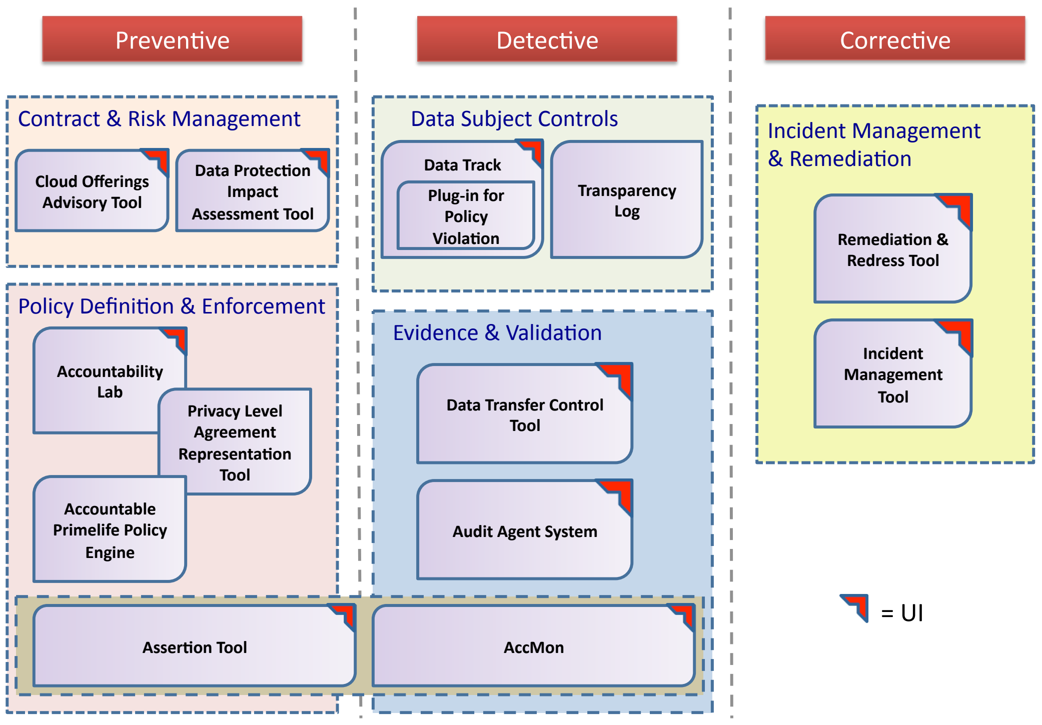
```
1   March 21, 2016, 5:54 p.m.   {p('x')}
2   March 21, 2016, 5:54 p.m.   {AAS_Snapshot('8fe3d428-bbcb-4777-aac8-44e6b80affd1','2016-02-05T09:38:24.008+01:00') | p('x')}
3   March 21, 2016, 5:54 p.m.   {AAS_Snapshot('8fe3d428-bbcb-4777-aac8-44e6b80affd1','2016-02-05T09:39:25.652+01:00') | p('x')}
4   March 21, 2016, 5:54 p.m.   {APPLE_Id('273874d3') | APPLE_PiiAttributeName('Country') | APPLE_PiiOwner('Panos') | p('x')}
5   March 21, 2016, 5:54 p.m.   {APPLE_Id('33a5d1ee') | APPLE_PiiAttributeName('Country') | APPLE_PiiOwner('Panos') | p('x')}
6   March 21, 2016, 5:54 p.m.   {AAS_Snapshot('8fe3d428-bbcb-4777-aac8-44e6b80affd1','2016-02-05T09:40:26.399+01:00') | p('x')}
7   March 21, 2016, 5:54 p.m.   {AAS_Snapshot('8fe3d428-bbcb-4777-aac8-44e6b80affd1','2016-02-05T09:41:27.040+01:00') | p('x')}
8   March 21, 2016, 5:54 p.m.   {AAS_Id('273874d3') | AAS_PiiAttributeName('Country') | AAS_PiiOwner('Panos') | AAS_Snapshot('8fe3d428-bbcb-4777-aac8-
                                44e6b80affd1','2016-02-05T09:41:27.562+01:00') | p('x')}
```

Figure 3.12: AccMon – Assertion Tool plugin

3.4.2 Overview of the A4Cloud toolkit interactions

In this section, we provide an overview of the interactions of the A4Cloud toolkit. The diagram of Figure 3.14 shows one instance for each tool except for TL that appears in two instances: the first is used by A-PPLE to store its logs, the second is used by DTMT to store its logs and communicate with AAS. The diagram shows the different messages that are communicated between the tools. Communication between the tools is unidirectional or bidirectional as indicated by the links. Labels on the links provide an abstract description of the data that flows between the tools. The different data flows that are highlighted in the interaction diagram are:

- A-PPLE sends the A-PPL policy to AAS. The policy has been been generated by Acclab and PLART,.
- AAS sends incidents and violation notifications to IMT.
- AAS requests and collects events/evidence from A-PPLE.
- A-PPLE uses its TL instance to store and retrieve its logs.
- AAS can retrieve logs from the TL of A-PPLE
- DTMT sends notification to the IMT.



Preventive
Detective
Corrective
Contract & Risk Management
Cloud Offerings Advisory Tool
Data Protection Impact Assessment Tool
Data Subject Controls
Data Track
Plug-in for Policy Violation
Transparency Log
Incident Management & Remediation
Remediation & Redress Tool
Incident Management Tool
Policy Definition & Enforcement
Accountability Lab
Privacy Level Agreement Representation Tool
Accountable Primelife Policy Engine
Evidence & Validation
Data Transfer Control Tool
Audit Agent System
Assertion Tool
AccMon
= UI

Figure 3.13: High level view of the A4Cloud tools [GKP+15a]

- DTMT uses its TL instance to store its logs.
- AAS is allowed to access logs stored in TL of DTMT.
- IMT sends end user notification to A-PPLE.

The interaction diagram provides a global view of the interactions that occur between the tools. The validation focuses on the collaborations between tools in order to satisfy non-local accountability properties.

3.4.3 Data flows and data format

In addition to the tool interactions, we consider the data formats used for these interactions that have to be taken into account during validation.

3.4.3.1 AAS and *Evidence Store*.

Evidence is stored in an *Evidence Store* by the corresponding evidence collection agent. The gathered information is transformed into evidence record(s) as defined in FoE [TRAR+15].

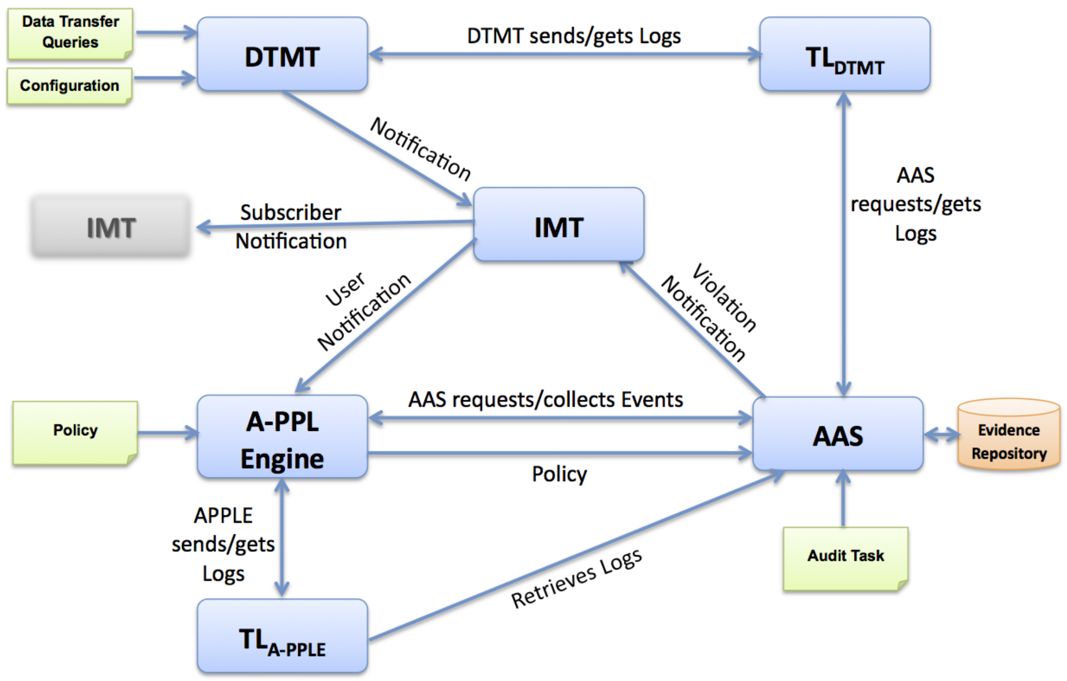


Figure 3.14: Toolkit interaction diagram.

Listing A.1 in the Appendix A shows a snapshot violation occurred at machine named *'Main-Container'*, collected by an agent named *DataRetentionPolicyEvaluationAgent*. There are supporting elements (which can be either an 'Blob' or an String).

3.4.3.2 AAS to IMT or AT (notify incident / policy violation)

AAS notifies the IMT about an occurred incident, and policy violations. Therefore all suspicious incidents are transformed into IMT notification messages. This message contains the generated evidence record[5] which describes the occurred incident. The policy violation report is also defined in FoE (Framework of Evidence) [TRAR+15].

Listing A.3 in Appendix A shows an example of an IMT notification sent by AAS regarding a possible Data Retention policy violation: The `custom_fields` contain the actual evidence - e.g., the PII store and delete messages as well as proof of an existing snapshot which includes the data already deleted by A-PPLE.

3.4.3.3 DTMT to A-PPLE (policy violation notification)

DTMT notifies the A-PPL Engine about a data transfer policy violation which has the three attributes as shown in the example below:

```
string logMessage: "John's virtual machine moved to Australia"
```

[5]defined in FoE (Framework of Evidence) [TRAR+15]

```
string resource: "virtual machine"
string owner: "John"
```

3.4.3.4 DTMT to TL (data transfer logs)

DTMT outsources all its logs to TL in JSON format. The log itself is simple text; an example of a log entry describing a new server creation is provided below:

```
"Create a new \"Server\" with id \"01asb2129231572321"\
name \"France Server\" in hypervisor \"frnc125\"
```

3.4.3.5 A-PPLE to TL

A-PPLE sends internal logs and notifications to the data subjects through the TL in JSON format. Below is an example of an internal log:

```
{
  "type":"Personal Data Deleted",
  "value":"Event associated with personal data  running belonging to
      WMIjiSqY1yIbs deleted"
}
```

Below is an example of a notification:

```
{
  "type":"Policy violation",
  "value":"This is a policy violation notification"
}
```

3.4.3.6 IMT to A-PPLE (end user notification)

IMT provides A-PPLE with the necessary information to notify end users, if this information has been given to IMT.

The payload sent if notifying users one by one:

```
{
  "owner": "johndoe",
  "resource": "creditcard",
  "message": "Your credit card was compromised, please cancel it."
}
```

The payload sent if A-PPLE should notify all users having a resource is the following:

```
{
  "resource": "creditcard",
  "message": "Your credit card was compromised, please cancel it."
}
```

3.4.3.7 IMT to IMT (subscriber notification)

Listing A.2 in Appendix A shows a notification sent by IMT about incident types to which the cloud customer subscribes. The same data format as the one used by AAS to IMT and AT is used.

3.4.4 High-Level description of the A4Cloud toolkit

In the following, we provide description of the tools that are subject to validation. These description have been updated compared to the their counterparts in the previous deliverable and milestones of D-6. This update includes, in particular, additions specifically geared towards validation, like validation interfaces and validation-specific information relating to the data structures used by the tools.

Incident Management Tool

The Incident Management Tool (IMT) is aimed at helping Cloud Service Providers (CSP) in handling anomalies and detected violations in cloud environments; it does not have an end-user interface[6]. Instances of IMT are capable of sending and receiving incident reports from each other through subscriptions, thus allowing each link in the cloud delivery chain to be informed about relevant incidents. When the IMT receives an incident report, an operator can create a local version of this incident report (which belongs to another entity) and notify end users through the A-PPL Engine and possibly other instances of IMT subscribing to this new incident. As a basis for our proposed specification, we have investigated functional aspects of how incident information should be represented and shared.

At some point, the CSP experiencing an incident needs to notify its cloud customers – that is other providers in the cloud supply chain buying services from the CSP. By mutual agreement it decides which notifications to be sent; by the provider listing which types of incidents the cloud customer is allowed to access, and the cloud customer subscribing to the incident types he needs. In order to allow the cloud provider to fulfil any legal obligations to notify the cloud customer, he could, when obligated by law, send such notifications whether the cloud customer subscribes to them or not.

Figure 3.15 shows an example of the relationship between service providers and services used, and thus provides an example of the amount of unneeded or unwanted incident information a subscriber could receive if the defined triggers are not taken into account. For example, in the case of cloud B, it only uses a fraction of the services offered by cloud C and D. If cloud B was notified about all incidents at C and D, this would include a large amount of unneeded information relating to services not in use by cloud B. Additionally if cloud D has thousands of customers, the overhead of negotiating and defining which incident information to share would become noticeable. Therefore, it is expected to be better to use the hybrid approach, where the subscriber defines what he wants to be notified about and the provider additionally pushes all information required by law to the subscriber by using mandatory notifications that could be defined for each subscriber.

IMT provides two interfaces from which incident information can be added: the graphical user interface – supporting human incident handlers – and the RESTful API – supporting machine to machine interaction. Through the graphical user interface, the incident handler can examine and

[6]End-user interaction is handled by the Remediation and Redress Tool (RRT).

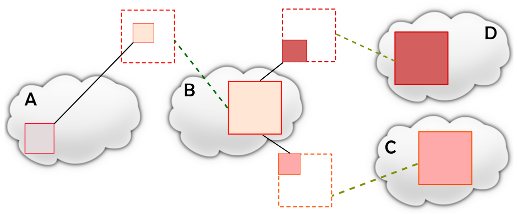


Figure 3.15: Data flow in supply chain. Each cloud represents a service provider. Each coloured square represents the services used by the subscriber. The coloured square inside each stippled square, represents the data or parts of the services actually used by the subscriber.

manage incident information as well as notify both subscribing cloud customers and end users. Through the API, detective tools such as DTMT and AAS can provide incident information to IMT. The API is also used to exchange incident information between different instances of IMT. Incidents provided through the API are encoded using a simple, extensible JSON format.

IMT also provides several API-methods to support tool validation:

- Notify subscribers:
  GET `/api/1.0/validation/incidents/{id}/notify`
  Substitute {id} for the ID of the incident for which you want to notify subscribers
- Notify end users:
  POST `/api/1.0/validation/incidents/{id}/notify-end-users`
  Must include ”message”: ”MESSAGE HERE”
  Substitute {id} for the ID of the incident for which you want to notify end users
- Get all incidents:
  GET `/api/1.0/validation/incidents`
- Get a specific incident:
  GET `/api/1.0/validation/incidents/{id}`

Additionally, IMT can be configured to interact with the Assertion Tool passive mode by setting the configuration variable `AT_ACTIVE = True`. This makes IMT push any relevant information directly to the Assertion Tool upon information being received or sent.

3.4.4.1 Data Transfer Monitoring Tool

The Data Transfer Monitoring Tool (DTMT) enables cloud service infrastructure to demonstrate compliance with respect to the location where the personal data processing occurs. The tool automates the collection of evidence, which demonstrates that obligations concerning personal data transfers are being carried out properly. To this end, it observes and analyze the operations performed on personal data which involves transfer at the infrastructure level, for example when performing load-balancing, or creating backups and then storing or deploying them in different hosts. These operations are accumulated in memory by a rule-based engine as the facts about the infrastructure layer. Combining with rules defined from agreements in user data between cloud customer and cloud provider, DTMT can identify whether all transfers were compliant with the authorizations from the Data Protection Authorities obtained beforehand by the data controller. In this way, policy violations can be detected and notified to data controller.

Part of the necessary information to monitor data transfers can be obtained from machine-readable policies specifying accountability obligations, such as A-PPL policies, but also the constraints about which parties, and geographical locations of physical hosts or datacenters are allowed for a given data set can be manually configured in the tool. Upon identification of a potential violation, further data can be collected to provide supporting evidence of an incident. Data controllers and processors can rely on tools such as AAS and IMT to perform further investigation about incidents. After the incident handling has been performed, the stakeholders can be notified using functionality of IMT combined with the A-PPL Engine. Notice that the tools can also be set up to handle notifications automatically, that is, without human intervention which is not advisable in most use cases. Therefore, notifications can be triggered by the auditor, but also automatically by of DTMT using the A-PPL Engine's interface.

The DTMT tool inputs and outputs include:

- Inputs
  - An A-PPL policy identifying the data controller and its resources in the IAAS;
  - IAAS configuration : identifying server IP addresses, identifiers and their physical location;
  - Keys and service endpoints for the transparency log;
  - Runtime information from the IAAS: all the network traffic to and from the Openstack components is intercepted and analyzed - basically restful API calls.
- Outputs
  - Log entries to TL
  - Potential violation notifications to A-PPLE

3.4.4.2 Accountable-PPL Engine

The A-PPL Engine (A-PPLE) is an extension of the already existing PPL Engine [TNR11], and is defined as the tool for enforcing an accountability policy written in the new A-PPL language (see deliverable D34.2 [dOSP$^+$15]). Thanks to this new language, the A-PPL Engine enforces new accountability related obligations. The A-PPL Engine mainly enforces data handling obligations and some logging and notification rules related to the data handling rules.

The A-PPL Engine inherits different components from the existing PPL Engine, namely, the Policy Decision Point (PDP), the Policy Enforcement Point (PEP) and the Policy Administration Point (PAP) which were part of the underlying HERAS[7] access control engine. These components have been modified/extended in order to be able to enforce new obligations defined in A-PPL. The enforcement of the A-PPL policy is coordinated by the PEP which associates the rules to be enforced to the corresponding data and further informs the Obligation Handler. The Obligation handler applies all the enforcement actions except the logging ones which are forwarded to the Logging Handler. The reason why there exists a separate Logging Handler is that this component is also continuously logging all actions taken over the data. The A-PPL Engine

[7]http://www.herasaf.org/

also defines a RESTful API to provide an entry point to the A-PPL Engine. More details about A-PPLE can be found in deliverable D43.2 [dOSoTP+15a].

The outputs of the A-PPL Engine are:

- logs (defined in JSON format) generated by the Logging Handler;
- notifications generated following an A-PPL policy obligation (for example upon a policy violation);
- personal data in case the access to retrieve the PII is granted.

3.4.4.3 Audit Agent System

The Audit Agent System (AAS) enables the automated audit of multi-tenant and multi-layer cloud applications and cloud infrastructures for compliance with custom-defined policies, using software agents. Agents can be deployed at different cloud architectural layers (i.e., network, host, hypervisor, IaaS, PaaS, SaaS) with the purpose of evidence collection and processing as well as generating audit reports.

The tool configures software agents and deploys them in agent runtime environments located at different cloud architectural layers. The type of agent that's being deployed and therefore the data collected depends on the audit task. The requirements defined in the policies or manually by the auditor are evaluated against the collected data (evidence). The evaluation results are presented in a web-based dashboard and may also be used in the automatic generation of audit report documents.

The outputs of the AAS are:

- Evidence Records (defined in the Framework of Evidence [TRAR+15]) stored inside the AASs local evidence store.
- Notification to IMT about new incidents.
- Logs for each agent (regardless of whether it is a Client or a Core AAS).
- Presentation of audit task results via the AAS web dashboard.

AAS provides the following APIs for tool validation support:

- AAS provides a rest call returning the recipient key for the user ”aas” ath the TL instance of APPLE. This is required in order to check the compliance of incidents created by AAS containing messages logged by APPLE. The recipient key can be requested using the following REST call:
  GET `/rest/validation/getAppleTlAasRecipientKey`
- A complete log of all actions performed for an audit task is offered by the:

  `de.hfu .A4Cloud.validationLog.ValidationLogAgent`

  Since AAS is a distributed system, this agent receives several information from any agent registered with its execution environment (JADE container) and persists it inside /opt/-Core/log/validation.txt All information received have the following format:

`<timestamp> <action> <additional information>`

These distributed logs can be requested using the following REST call:
GET `/rest/validation/getValidationLogs`

- All incidents notified to IMT are also sent to the AT for further assertions

3.4.4.4 Transparency Log

The Transparency Log (TL) is a secure and privacy-friendly one-way messaging system, enabling *senders* to send messages to potentially offline *recipients*. TL is asynchronous, tamper-evident, encrypts messages using state-of-the-art algorithms, prevents some information leaks, publicly verifiable, safe to outsource, enables both sender and recipient to prove several properties of sent messages, and supports distributed settings. TL's primary role in A4Cloud is to facilitate the communication of data from data controllers to data subjects. TL can be seen as a secure and privacy friendly replacement of email where the sender is fixed (the data controller) and recipients (data subjects) have to register at the sender (similar to providing their email address). In A4Cloud, other tools will provide notifications generated at the data controller that should be parsed by other tools used by a data subject. TL plays the role of a privacy friendly transport of these notifications. TL is also used as a secure log by A-PPLE and DTMT, and as an evidence store by AAS.

TL consists of two components: the sender and the recipient. Both the sender and recipient provide RESTful API over HTTPS. The sender is used for sending messages to recipients. Recipients register at the sender API by providing their public key. Private keys for recipients are stored in the recipient component, that can be used to reconstruct all messages sent to the recipient. The sender component is intended to run at the entity sending messages, such as the service provider. The recipient component should run *locally* at the recipient, such as data subjects.

Assuming a registered recipient, the TL tool input and output are:

- Input: A sequence of bytes for a recipient, using the TL sender.
- Output: The sequence of bytes sent to the recipient, using the TL recipient.

TL uses JSON and base64 encoding of data in different parts of its API, see the API documentation. The output can also be retrieved as raw bytes. Depending on use-case, the bytes stored in TL represent different data. This representation is use-case specific, and not the responsibility of TL to specify.

3.5 Summary

In this chapter, we first provided a description of the architecture of the assertion framework, and detailed the description of its main component, the assertion tool. We then provided an overview the A4Cloud toolkit that is considered for validation on D-6 WP and presented the APIs for tools validation.

Chapter 4

Validation Scenarios and Demo

As part of WP D-7 a realistic cloud service supply chain has been developed and deployed in order to demonstrate the accountability framework and the respective tools developed by the A4Cloud project. A scenario involving commercial and institutional actors in the domain of the wearables has been developed in order to perform concrete accountability tasks. This scenario constitutes a realistic and topical scenario, in which the involved business actors have to take the appropriate actions to ensure that the collection and processing of the customers' personal data are handled responsibly, based on the established regulations and declared security organizational policies.

We have harnessed this cloud environment in order to validate the A4Cloud policy-enforcement tool chain. We have defined five validation scenarios that enable different accountability properties of the tools to be validated in the context of the cloud service supply chain for wearables. These scenarios have then be used to automatically validate the accountability properties of the tools in two concrete deployments of the supply chain provided by partners ATC and HFU.

In this chapter, we first provide an overview of the D-7 wearable scenario in Section 4.1. In Section 4.2, we then present the five validation scenarios that we have developed as part of D-6 and how they are have been used for the accountability tool validation using the Assertion Tool. Finally, we provide a summary in Section 4.3.

4.1 Wearable scenario

The demonstration scenario in A4Cloud is set up from the perspective of the Wearable Co, which is the cloud customer and data controller offering the wearables service to their customers. This service is provided by Kardio-Mon with the support of Map-on-Web, and the DataSpacer. In this scenario, the Wearable Co regulates the type of data that should be collected from the Wearable Co Customers, the purpose for doing so and the accountability policies, under which this data will be processed and stored in the cloud. Such policies are compiled by Kardio-Mon and agreed with the Wearable Co.

Overview of the scenarios and the involved actors. Figure 4.1 shows an overview of the wearable scenario deployment.

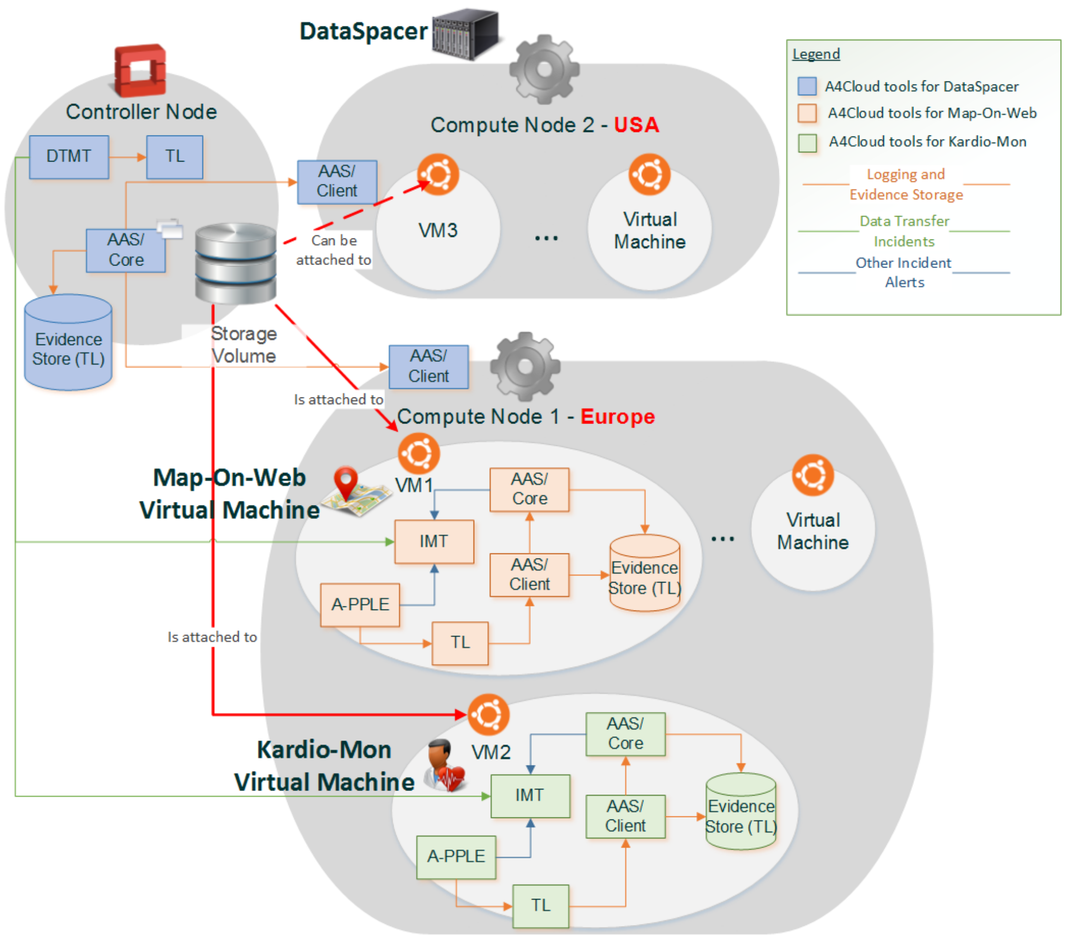


Figure 4.1: The deployment of the A4Cloud tools for the wearable use case scenario.

The DataSpacer operates as an independent IaaS cloud provider, offering cloud storage and computation services out of a number of datacentres located in different geographical locations around Europe and US. Those data centres are governed by different regulatory frameworks based on their location. DataSpacer expects to collaborate with cloud service providers hosting any kind of personal data. Thus, DataSpacer has to implement by default security and privacy mechanisms, while the relevant accountability tools, namely DTMT, AAS and IMT, have to be deployed to enable DataSpacer providing verification on their data handling processes upon ad-hoc requests from the collaborating cloud service providers or the governing cloud supervisor authorities and auditors.

Map-on-Web operates as a SaaS provider for delivering various data management solutions over large data streams, including data aggregation and smart visualisations. This data could be either public or personal data, thus Map-on-Web should consider for the appropriate accountability measures to process this data in a responsible way. These measures include the deployment of A-PPLE, AAS and IMT in their space. Additionally, Map-on-Web needs cloud storage facilities and, to this end, DataSpacer is selected.

Another actor involved in the wearable scenario is Kardio-Mon. This is a SaaS cloud provider, which delivers an in-house development for a Wearable Service, aiming to support potential cloud customers to deploy their own wearable platform services. Such services include the collection and processing of personal data from individuals, as well as advanced visualisations, which are directly fed from Map-on-Web as a third party provider of Kardio-Mon. On top of that, Kardio-Mon needs storage facilities which are provided by DataSpacer. Thus, the development and operation of the customised Wearable Service (for a specific client of Kardio-Mon) should take into account the deployment of the appropriate accountability tools. The tools include A-PPLE, AAS and IMT and they should be put in operation, as soon as an enforceable set of accountability policies has been agreed between Kardio-Mon and the cloud customer. It should be noted that these policies have been specified to reflect the capabilities of Kardio-Mon, Map-on-Web, and DataSpacer to implement specific security and privacy measures.

Scenario implementation and demo. For the implementation of the scenario from an accountability perspective, each actor involved in the provision of the wearable service use case should address the functional elements of the accountability lifecycle, which has been introduced in D42.4. In technical terms, each actor should be able to demonstrate how they follow the respective functional elements of the lifecycle, adopt the accountability practices and implement the respective accountability mechanisms. The implementation of the relevant mechanisms is supported by the A4Cloud tools, which are set up in the environment of each actor to facilitate specific accountability support services.

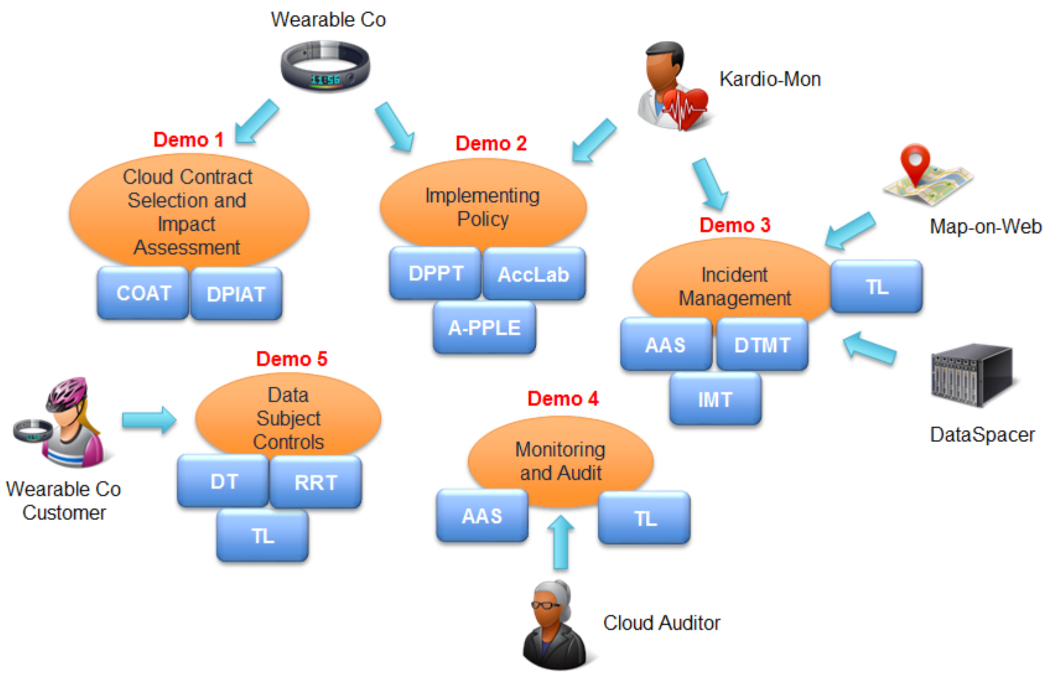


Figure 4.2: The scenarios for the accomplishment of the wearable service

For the design, development and execution of the wearable scenario, we emphasize on key activities that should be performed by each actor, as presented in Figure 4.2. As presented there, the main demonstration scenario is instantiated into five smaller scenarios, each one identifying a main actor to run the respective scenario. These scenarios involve all A4Cloud tools.

For this deliverable, we focus on the scenarios that involve the interaction of the A4Cloud tools for policy enforcement. Thus, in the scenario Demo 2: ”Policy Implementation”, we demonstrate the use of the policy definition, validation and enforcement tools from the perspective of the cloud provider. In this scenario, Kardio-Mon uses DPPT to create accountability policies. This process involves the interaction with the Wearable Co, who exploits AccLab to perform a matching between the abstract policy statements (capabilities) being advertised by Kardio-Mon with the provisions of the accountability policies. These tools can run ad-hoc on one’s desktop machine.

Kardio-Mon hosts A-PPLE, which is used to enforce the accountability policies for the management of personal data. Thus, the policies agreed between Kardio-Mon and the Wearable Co are submitted to A-PPLE through the DPPT. In turn, these policies are exploited by Kardio-Mon to control the collection and processing of the personal data of the Wearable Co Customers.

During the operation of the Wearable service, the interaction of the Wearable service being maintained by Kardio-Mon with the Map-on-Web services and the storage service facilities of DataSpacer are monitored by a set of tools. More specifically, the A-PPLE instance of Kardio-Mon generates logs on how this provider operates the respective data handling processes in terms of the data access and retention rules of the policies. Kardio-Mon is also deploying AAS to collect logs from all the operations happening across the cloud protocol stack of this provider and assess accountability in terms of data retention and integrity. These two tools are used from Kardio-Mon to enforce the data handling processes against the wearable service and the Map-on-Web service. Furthermore, AAS is used by Kardio-Mon to collect loges and create evidence records on the practices occurred at runtime. The collection of logs and the storage of evidence records is supported by TL. AAS is further used in scenario Demo 4: ”Monitoring and Audit”, in which we present the perspective of Cloud Auditor to perform an audit task on Kardio-Mon.

In order to implement accountability on the storage layer, DataSpacer has deployed two tools, namely DTMT to generate logs on the data handling processes with respect to data transfer rules agreed in the accountability policies and AAS to collect logs and create evidence in the environment of this IaaS cloud provider. Again, TL is integrated with AAS and DTMT to facilitate logs collection and storage of the DTMT evidence records. AAS is also used in scenario Demo 4: ”Monitoring and Audit”, to enable a Cloud Auditor perform an audit task on DataSpacer.

In the scenario Demo 3: ”Incident Management”, we present the perspective of the cloud providers in order to tackle with the detection of abnormal operations at runtime. At this point, Kardio-Mon and DataSpacer can automatically detect security breaches or incidents on potential policy violations through the deployed AAS and DTMT (only for DataSpacer) tools. For the management of the incidents, both Kardio-Mon and DataSpacer deploy IMT. This tool is used to communicate the incidents detected from AAS and DTMT. For example, in case that a data transfer incident is detected at DTMT, the IMT instance of DataSpacer is triggered. The IMT operator of DataSpacer assesses the incident and decides on whether to communicate this to the IMT instance of Kardio-Mon. In this case, the IMT operator of Kardio-Mon has to assess the received incident and decide on how to handle this. In case that this incident must be notified to the Wearable Co and their customers, IMT delegates the incident reporting process to A-PPLE.

On Demo 5: ”Data Subject Controls” scenario, we present the capabilities offered to the

Wearable Co Customers as cloud subjects to control the disclosure of their personal data in Kardio-Mon and the respective third party cloud providers (Map-on-Web and DataSpacer). This is done through DT, which integrates TL to secure the communication of the cloud subject device with the A-PPLE instance of Kardio-Mon. DT embeds RRT as a tool to provide suggestions on possible remediation to the Wearable Co Customers.

Validation scenarios derived from the wearable scenario. In D-6, we consider five validation scenarios which are directly linked to the wearable scenario.

- Intrusion Detection. Kardio-Mon hosts A-PPLE, which is used to enforce the accountability policies for the management of personal data. The Intrusion Detection validation scenario concerns a data protection problem during which an attacker tries to get unauthorized access to data.
- Data Retention. Potential data retention violations may be recognized in the service, based on the existence of PII in virtual machine snapshots at Data-Spacer. The Data Retention validation scenario aimed at validating the compliance with the data retention requirements stated in the related policy.
- Data Transfer across data centers. The accountability policy may specify that data collected and processed by Kardio-Mon, for instance, should be only maintained in the servers residing in the EU territory. For some reasons, the related storage volume may be detached from the EU servers and attached to another server cluster in US (see red links in Figure 4.1). This is a data transfer action that comprises an exception to the policy and should be raised. This scenario aimed at validating the compliance with the data transfer requirements.
- Data transfer across hosts. DataSpacer implements security and privacy mechanisms. To this end, the relevant accountability tools, namely DTMT, AAS and IMT, have to be deployed to enable DataSpacer providing verification on their data handling processes upon ad-hoc requests from the collaborating cloud service providers or the governing cloud supervisor authorities and auditors. The aim of this validation scenario is to validate that the incidents detected by DTMT are correctly handled and communicated to IMT.
- End User Notification. The Wearable Co would not necessarily receive the required incident information from Kardio-Mon or DataSpacer in order to notify their end users. To this end, IMT receives detected incidents from DTMT and AAS, and utilizes A-PPLE to notify end users about incidents relevant for them. This validation scenario allows us to validate that the end user notifications are correctly provided to A-PPLE.

4.2 Validation scenarios

We have defined five validation scenarios that allow different accountability properties of the tools to be validated. In the following we present each of these scenarios in two different steps:

- General description of the scenario and how it concerns different A4Cloud tools.
- Detailed description of how the scenario is used for validation of specific accountability properties using the Assertion Tool.

4.2.1 Intrusion detection validation scenario

General description

The Intrusion Detection validation scenario concerns a data protection problem during which an attacker tries to get unauthorized access to data.

The main goal of A-PPLE tool is to enforce the accountability policies for the management of personal data. Kardio-Mon hosts A-PPLE to control the collection and processing of the personal data of the Wearable Co Customers, by configuring A-PPLE with the necessary accountability policies. Whenever an intrusion that violates the policies occurs, A-PPLE should detect and log information related to this intrusion. Other accountability enforcement tools, such as AAS, also contribute to the accountability enforcement, for example, AAS creates an *EvidenceRecord* related for the intrusion, and also notifies IMT about it.

This scenario aims at validating, in case of an intrusion detection, that A-PPLE logged all related events, and that a notification is communicated to IMT. It covers several accountability properties, in particular:

P1 A transparency property: "Accesses from third-parties to data are made explicit."

P2 A responsiveness property: "Intrusion detection attempts are notified in a timely manner."

P3 A responsibility property: "The responsible entity for intrusion detection is made explicit and information about the source of the intrusion is made available."

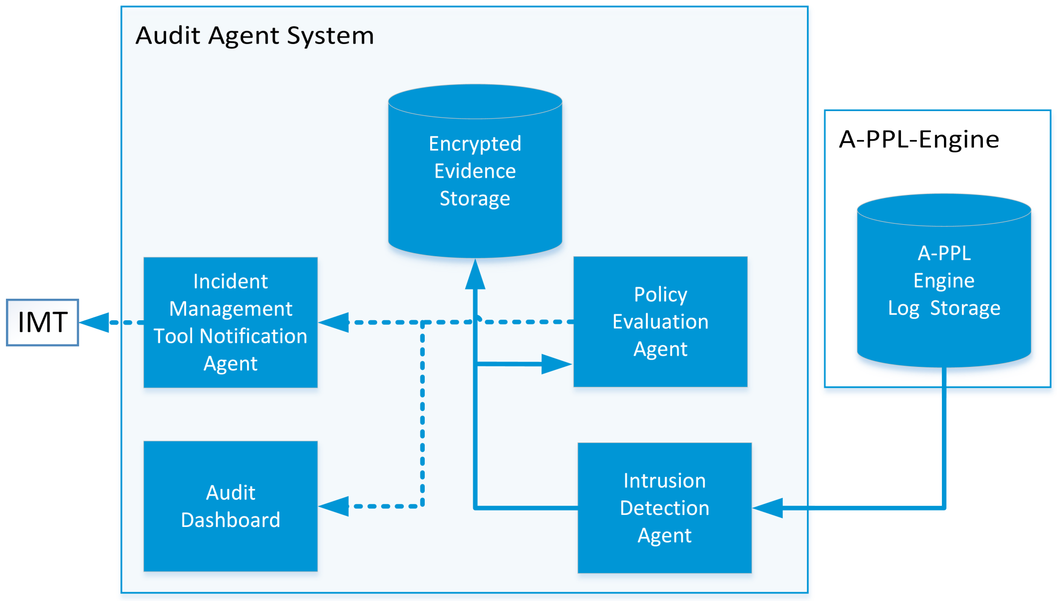


Figure 4.3: Intrusion detection audit task

Additionally, given that the timely notification of incidents is an important aspect, it could be useful to quantitatively evaluate the delay between the time of the incident and the time of notification. The Catalog of Accountability Metrics does not contain a metric for this aspect. However, we can define a new metric, called Average Timing of Notification (Metric 40), following the approach defined by WP C-5. This metric describes quantitatively how timely the notifications are delivered to interested stakeholders (e.g., end-users, administrators, etc)

- Input: This metric is computed using the following parameters:
  - N – Total number of notifications for a given period of time
  - A_i – Time of the i-th notification during the given period of time, expressed in UNIX time
  - B_i – Time of the i-th incident during the given period of time, expressed in UNIX time
- Formulation and output: Output = $\frac{\sum_{i=1}^{N}(A_i - B_i)}{N}$

This metric can be used in runtime-based validations, such as our validation methodology.

This validation scenario involves the following four A4Cloud tools: A-PPLE, AAS, IMT, and TL. The interactions between these tools are shown in Figure 4.3. In the scenario, we consider intrusions consisting in private data from a subject being accessed by an unauthorized person a number of times greater than the allowed threshold. This threshold is specified by the PII policy which is installed on A-PPLE. A-PPLE logs all the PII accessing events in the TL instance. An intrusion attempt can then be determined by AAS by analyzing the access logs written by A-PPLE on TL. This can be achieved by running the Intrusion Attempt Detection Audit Task, which is designed to communicate with the TL instance. When AAS finds an intrusion attempt (the no. of PII accesses exceeded the configured threshold), it should transform all events related to these accesses into an *EvidenceRecord*, which is then stored inside the *Evidence Store* Finally, AAS transforms the created *EvidenceRecord* to a notification (in JSON notation), and sends it to IMT.

Validation scenario and tool validation

This validation scenario consists of the following steps:

1. Start the Intrusion Detection Audit Task using the AAS *setAuditTask* REST call. This triggers AAS to checks periodically whether the number of PII accesses is greater than a certain threshold.
2. Configure the PII policy on A-PPLE. This policy specifies the threshold for accessing the PII.
3. Simulate an intrusion by accessing the same PII number of times greater than the allowed threshold. After simulating the intrusion, AAS should detect it, and then notify IMT about it.

Performing this validation scenario, we can use the Assertion Tool to assert the following tool-specific accountability-related properties:

T1 Check whether A-PPLE logs all the PII accesses

T2 Check whether AAS created an *EvidenceRecord*

T3 Verify that the *EvidenceRecord* created by AAS contains all the events logged by A-PPLE.

T4 Check whether IMT received a notification from AAS

T5 Verify that IMT received the notification from AAS on time.

T6 Verify that the notification received by IMT matches the *EvidenceRecord* stored by AAS in the Evidence Store.

Note that these tool-specific assertions allow the validation of the accountability properties P1–P3 introduced in the beginning: P1 follows from T1, T3 and T6, P2 from T5, and P3 from the contents of the evidence created in T2 and the contents of the notification T6.

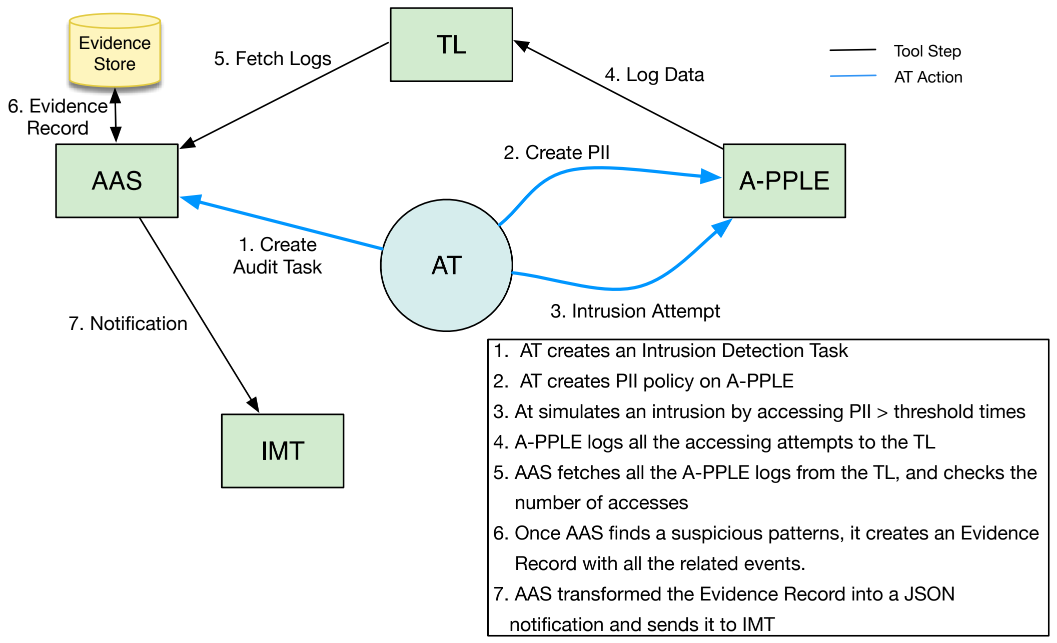

Evidence Store
5. Fetch Logs
TL
4. Log Data
Tool Step
AT Action
6. Evidence Record
AAS
2. Create PII
A-PPLE
1. Create Audit Task
AT
7. Notification
3. Intrusion Attempt
IMT
1. AT creates an Intrusion Detection Task
2. AT creates PII policy on A-PPLE
3. At simulates an intrusion by accessing PII > threshold times
4. A-PPLE logs all the accessing attempts to the TL
5. AAS fetches all the A-PPLE logs from the TL, and checks the number of accesses
6. Once AAS finds a suspicious patterns, it creates an Evidence Record with all the related events.
7. AAS transformed the Evidence Record into a JSON notification and sends it to IMT

Figure 4.4: AT driving the intrusion detection validation scenario

Figure 4.4 shows how AT can interact (blue links) with the a4Cloud tools (A-PPLE, AAS, IMT) to execute the Intrusion Detection validation case (Section 4.2.1). AT starts the validation case by triggering AAS for intrusion detection (Step 1), then creates a PII at A-PPLE (Step 2), and finally simulates an intrusion by accessing the PII a number of times which is greater than the configured threshold (Step 3). Figure 4.4 also shows the internal interactions between the A4Cloud tools: A-PPLE logs to TL all PII accesses (Step 4), AAS pulls the A-PPLE logs from TL (Step 5), AAS creates an *EvidenceRecord* with all the related events once it detects a suspicious pattern (Step 6), and AAS notifies IMT about the detected intrusion (Step 7).

During the execution of the validation case, AT also gets some data from A4Cloud tools, in particular the A-PPLE logs, AAS *EvidenceRecord*, and IMT notification. AT then uses the received information to verify the assertions to validate the intrusion detection case. Figure 4.5 shows the information AT gets from the A4cloud tools, and also the assertions that AT checks (red bullets): A-PPLE logs all the PII accesses, AAS created an *EvidenceRecord* that contains

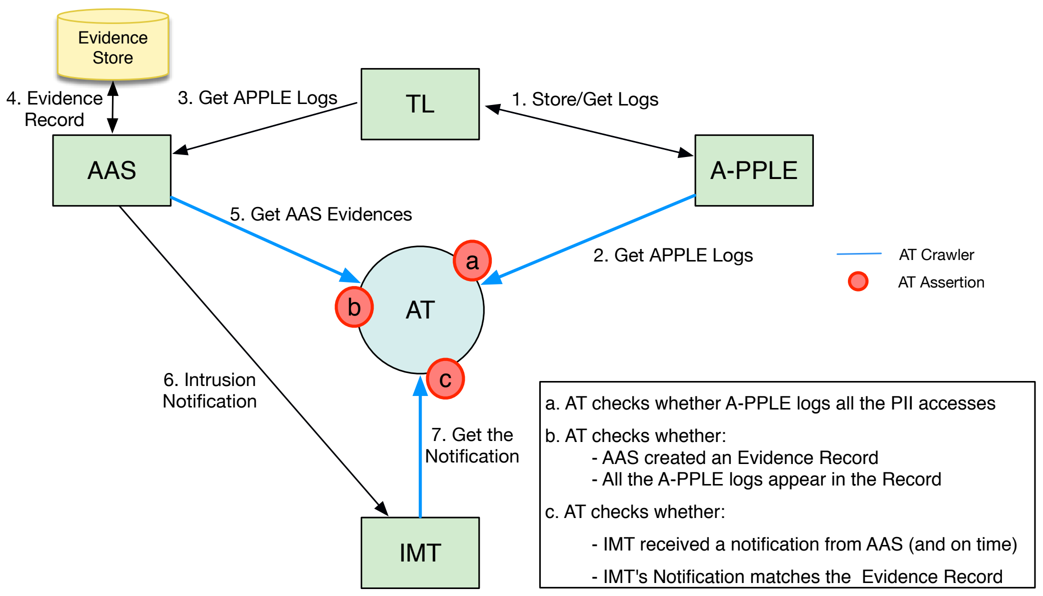

Evidence Store
4. Evidence Record
3. Get APPLE Logs
TL
1. Store/Get Logs
AAS
A-PPLE
5. Get AAS Evidences
2. Get APPLE Logs
AT
a
b
c
AT Crawler
AT Assertion
6. Intrusion Notification
7. Get the Notification
IMT
a. AT checks whether A-PPLE logs all the PII accesses
b. AT checks whether:
- AAS created an Evidence Record
- All the A-PPLE logs appear in the Record
c. AT checks whether:
- IMT received a notification from AAS (and on time)
- IMT's Notification matches the Evidence Record

Figure 4.5: AT checking assertions related to intrusion detection validation scenario

Listing 4.1: Java code for intrusion detection validation scenario

```java
public class IntrusionDetectionVS {
  //  I. Declare your tools
  AASTool aas = new AASTool();
  AppleTool apple = new AppleTool();
  IMTTool imt = new IMTTool();

  //  II. Setup Phase
  public void setUp()
  { RestModule.initializeInfrastructure(); }

  //  III. Validation Scenario
  public void scenario() {
    // a. Initialize the system & add the policy
    int threshold = 5;
    aas.createIntrusionDetectionAuditTask(threshold);
    apple.createPII("policies/panosCountryPii.xml");

    // b. Attempt to access PII more than ‘threshold‘ times
    for (int i = 0; i <= threshold;  ++i)
    { apple.accessPIIFromPanos(); }

    // c. Assert!
    ACCOUNT.intrusionDetection("Panos", "Country", threshold);
  }
}
```

all the events logged by A-PPLE, and IMT received a notification, on time, which is consistent with the AAS *EvidenceRecord*.

Listing 4.1 presents the executable Java representation of the Intrusion Detection validation scenario. It consists of three parts, respectively initializing the tools used as part of the scenario. Note that the TL instance is not made explicit here, because it is used as an implicit communication channel between the A-PPLE and AAS tools. In the second part some setup code for the AT is defined. The third part represents the validation scenario itself. This scenario, due to its conciseness and use of abstractions (methods, assertion), is essentially self-explanatory. Note that, as described in the previous chapter, no code is included for the gathering of information from logs and notification, because the AT handles this aspect implicitly and automatically using the default strategy for information gathering that requests the necessary information before each assertion is checked.

The assertion `successfulAccess()` is a complex one, which verifies that A-PPLE logs all the events related to some intrusion attempts, and that all these events are present in the related *EvidenceRecord* that results in a correct notification to be sent from AAS to IMT. Such an assertion can be explicitly defined as a conjunction of three tool specific assertions:

- `apple.intrusionDetection()`: checks whether A-PPLE logs all the related events.
- `aas.intrusionDetection()`: checks whether all the events logged by A-PPLE are present in the AAS *EvidenceRecord*.
- `imt.intrusionDetection()`: checks whether AAS sends the right notification to IMT and that IMT propagates it correctly.

Remember that the AT calls these predefined tool-specific assertions automatically when the complex assertion of the validation scenario is interpreted.

Using this validation scenario we have successfully validated the accountability properties P1–P3 mentioned over the cloud supply chain deployed as part of the A4Cloud project.

4.2.2 Data retention validation scenario

General description

An auditor that investigates the compliance of Kardio-Mon and Data-Spacer with data retention policies can use the AAS tool to automate evidence collection and evaluation in a continuous manner. Potential data retention violations are recognized in the service, based on the existence of PII in virtual machine snapshots at Data-Spacer. Snapshots can violate data retention policies, if they hold PII for which the maximum retention time was exceeded. The Data Retention validation scenario aimed at validating the compliance with the data retention requirements stated in the related policy. In this sense, the Data Retention validation scenario concerns a data protection problem consisting in copies of data existing beyond the lifetime stated in a corresponding policy. This scenario covers several accountability properties, in particular:

P1 A transparency property: “Data copies are made explicit.”

P2 A responsiveness property: “Data retention issues are notified in a timely manner.”

P3 A responsibility property: “The responsible entity for a copy is available.”

With regards to accountability metrics associated to this validation scenario, we can remark that the metric “Record of Data Collection, Creation, and Update” (Metric 7) evaluates a critical aspect of this scenario. In order to track the compliance of data retention periods, it is necessary that the dates associated to each piece of data is recorded:

This metric describes a percentage of the extent to which date is recorded when collecting, creating and updating private records. Date of data collection, creation and update is relevant for complying with data retention schedules.

- Input: This metric is computed using the following parameters:
  - N – Number of collected PII records for which the date is indicated
  - T – Total number of collected PII records
- Formulation and output: Output = $(N/T) \cdot 100$

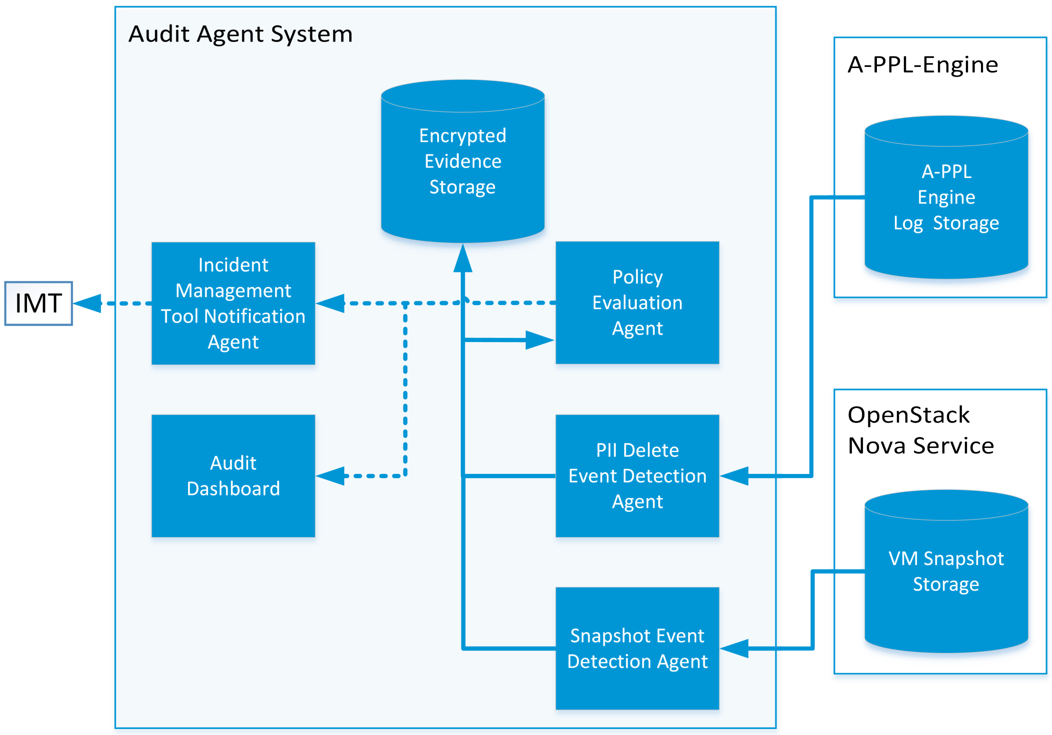


Figure 4.6: Data retention audit task

The scenario involves the following A4Cloud tools: A-PPLE, AAS, IMT, and TL. The main interactions involved in the data retention scenario are shown in Figure 4.6.

Validation scenario and tool validation

This validation scenario consists of the following steps:

1. Start the PII Data Retention Audit Task using the *setAuditTask* REST call. This triggers the PiiApplMessageCheckAgent to periodically pull all create and delete events from the A-PPLE's TL instance, and the PiiSnapshotCheckAgent pulls periodically snapshot existence information for a defined virtual machine. Both store their insights inside the AAS *Evidence Store*. The PiiDataRetentionPolicyEvaluationAgent decides if there has been an policy violation, depending on the create/delete messages as well as the VM snapshot meta data and it's timestamp.
2. Configure PII policy on A-PPLE. This policy specifies the data retention time. A-PPLE will force this policy and delete this PII from its database. A-PPLE logs to TL all the create/delete events.
3. Simulate a violation by creating a snapshot containing the related PII before the retention time. When AAS recognizes a suspicious pattern (existence of a snapshot that contains the PII after the retention time), it transforms all the related events into an *EvidenceRecord*, which is then persisted inside the *Evidence Store*. AAS also creates an IMT JSON notification form the *EvidenceRecord*, and sends it to IMT.

Performing this validation scenario, we can use the Assertion Tool to assert the following tool-specific accountability-related properties:

T1 Check whether A-PPLE logs all creation/deletion of the PII.

T2 Check that AAS keeps track of data copies, snapshots in particular.

T3 Check whether AAS created a violation related *EvidenceRecord* including an offending snapshot, that is, a data copy live after the retention period.

T4 Verify that the *EvidenceRecord* created by AAS contains all the events logged by A-PPLE.

T5 Check whether IMT received a notification from AAS

T6 Verify that IMT received the notification from AAS on time.

T7 Verify that the notification received by IMT matches the *EvidenceRecord* stored by AAS in the Evidence Store.

Once again, these tool-specific validations entail the validation of the accountability properties introduced in the beginning: P1 follows from T2, T3 and T7, P2 follows from T6, and P3 from T4 and T7.

AT can interact with the A4Cloud tools (A-PPLE, AAS, IMT) and an OpenStack node to execute the Data Retention validation scenario. The required interactions are shown in Figure 4.7 (blue links), together with the a4Cloud tools internal interactions (black links). To execute the Data Retention validation scenario, AT essentially (see Section 4.2.1) starts the PII Data Retention Audit Task (Step 1), creates PII on A-PPLE (Step 2) and simulates a violation by creating a snapshot containing the related PII before the retention time. A4cloud tools and OpenStack also interact internally: A-PPLE logs to TL (Step 4), AAS gets data/logs from TL's A-PPLE instance and OpenStack VMs (Steps 5-6), AAS creates an *EvidenceRecord* (Step 7) and notifies IMT (Step 8).

Figure 4.8 shows the information AT receives from the A4cloud tools, and also the assertions that AT checks (red bullets): A-PPLE logs all the PII creation/deletion, AAS created an

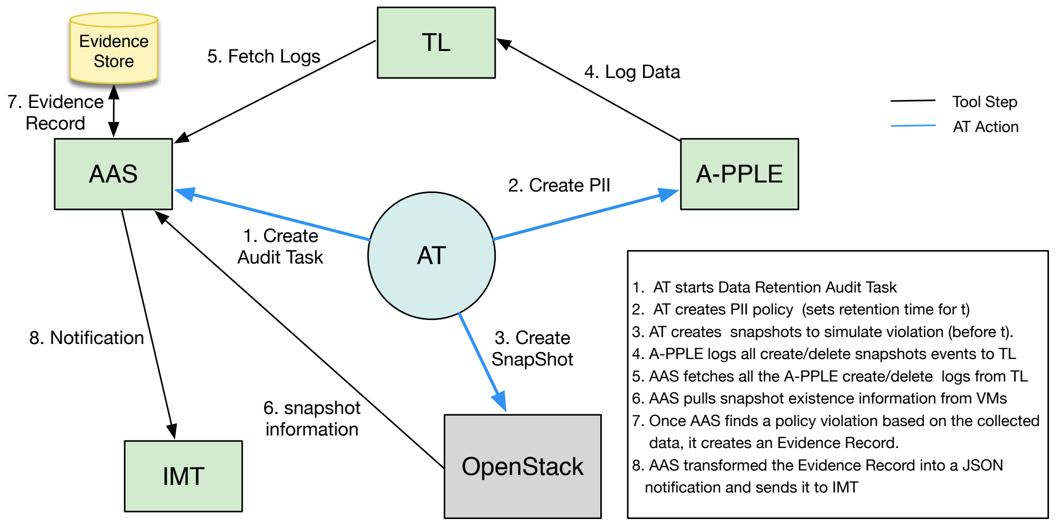

Evidence Store
5. Fetch Logs
TL
4. Log Data
7. Evidence Record
Tool Step
AT Action
AAS
2. Create PII
A-PPLE
1. Create Audit Task
AT
8. Notification
3. Create SnapShot
6. snapshot information
IMT
OpenStack
1. AT starts Data Retention Audit Task
2. AT creates PII policy (sets retention time for t)
3. AT creates snapshots to simulate violation (before t).
4. A-PPLE logs all create/delete snapshots events to TL
5. AAS fetches all the A-PPLE create/delete logs from TL
6. AAS pulls snapshot existence information from VMs
7. Once AAS finds a policy violation based on the collected data, it creates an Evidence Record.
8. AAS transformed the Evidence Record into a JSON notification and sends it to IMT

Figure 4.7: AT driving data retention validation scenario

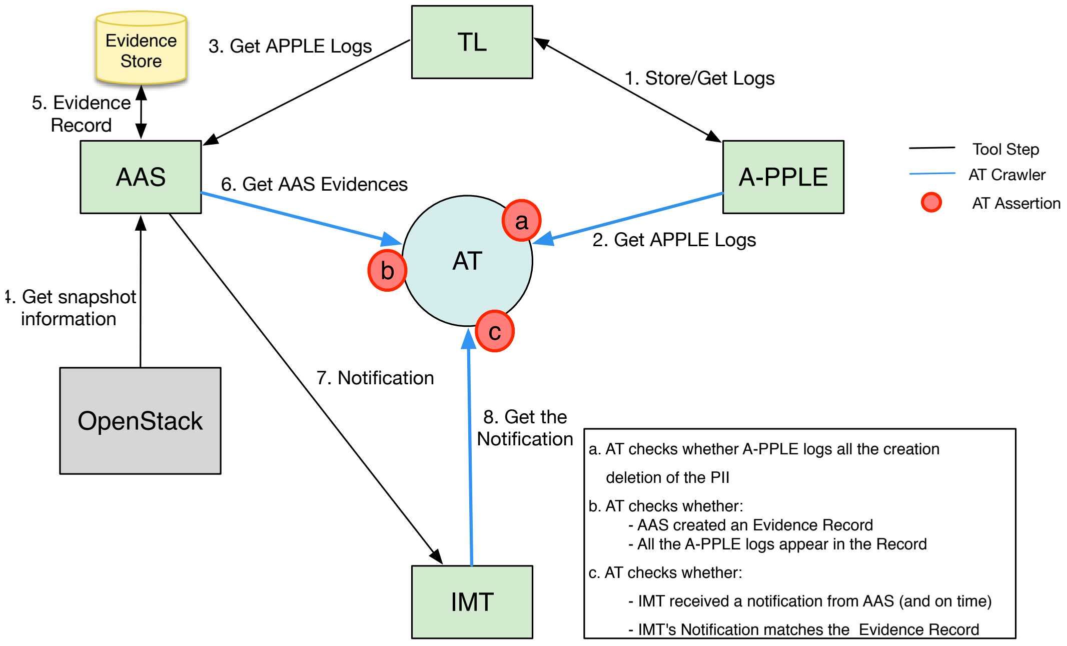

Evidence Store
3. Get APPLE Logs
TL
1. Store/Get Logs
5. Evidence Record
Tool Step
AT Crawler
AT Assertion
AAS
6. Get AAS Evidences
A-PPLE
a
b
AT
2. Get APPLE Logs
c
4. Get snapshot information
OpenStack
7. Notification
8. Get the Notification
IMT
a. AT checks whether A-PPLE logs all the creation deletion of the PII
b. AT checks whether:
- AAS created an Evidence Record
- All the A-PPLE logs appear in the Record
c. AT checks whether:
- IMT received a notification from AAS (and on time)
- IMT's Notification matches the Evidence Record

Figure 4.8: AT checking assertions related to data retention

Listing 4.2: Java representation of the Data Retention validation scenario

```java
public class DataRetentionVS {

  //  I. Declare your tools
  AASTool aas = new AASTool();
  AppleTool apple = new AppleTool();
  IMTTool imt = new IMTTool();
  OpenStackTool openstack = new OpenStackTool();

  //  II. Setup Phase
  public void setUp()
  {
    RestModule.initializeInfrastructure();
  }

  //  III. Validation Case
  public void scenario() {

    // a. Create Data Retention Audit Task
    aas.createDataRetentionAuditTask();

    // b. Configure related policy
    apple.createPII("policies/panosCountryPii2Min.xml");

    // c. Create OpenStack SnapShot
    openstack.createSnapshot();

    // d. Assert!
    ACCOUNT.retentionViolation();
  }

  public void finalize() {
    //  IV. Cleaning Case - Delete SnaphShot
    RestModule.opensDeleteSnapshot(snapshotName));
  }
}
```

EvidenceRecord that contains all the events logged by A-PPLE, and IMT received a notification, on time, which is consistent with the AAS *EvidenceRecord*.

Listing 4.2 shows the Java representation of the Data Retention validation scenario. The tool declaration phases, initialization of the AT and the validation scenario are defined in a way analogous to the intrusion detection scenario. The assertion `retentionViolation` in particular is automatically broken down into assertions for the involved tools. The scenario shows the use of a method `finalize()` for cleaning up tasks.

Using this scenario we have successfully validated the properties P1–P3 using the Assertion Tool.

4.2.3 Data transfer across data centers validation scenario

General description

In order to generate alerts of potential violations, a logging and evidence storage is maintained by the DataSpacer. The accountability policy may specify that data collected and processed by Kardio-Mon, for instance, should be only maintained in the servers residing in the EU territory. Considering the fact of a human error or an administration process, the related storage volume may be detached from the EU servers and attached to another server cluster in US. This is a data transfer action that comprises an exception to the policy and should be raised. In this respect, DTMT should log this transfer action on its TL instance in DataSpacer and inform the A-PPLE instance of Kardio-Mon. This scenario validation scenario involves the A4Cloud tools: AAS, DTMT, IMT and TL. It covers several accountability properties, in particular:

P1 A transparency property: “Cross-border data transfers are made explicit.”

P2 A responsiveness property: “Data transfer violations are notified in a timely manner.”

P3 A responsibility property: “The responsible entity for data transfer violations is made explicit and information about the source is made available.”

Given that timely notifications are important part of this validation scenario, the metric for Average Timing of Notification (Metric 40), see the Intrusion Detection validation scenario, can be reused here.

This validation scenario consists of the following steps:

1. Start the DTMT check Audit Task (DataTrackAuditTask) using the *setAuditTask* REST call. DTMT will check agent to periodically pull for new messages inside the DTMT’s TL instance.
2. Manually, the OpenStack admin moves a VM to a forbidden location. DTMT recognizes this event and logs the operation with a flag into its TL instance. DTMT also notifies IMT about the violation.
3. When AAS recognizes a location violation, it transforms the violation into an *EvidenceRecord* which is then stored inside the *Evidence Store*.

Using this validation scenario we can assert the following:

T1 Check whether AAS created an *EvidenceRecord* related to the location violation

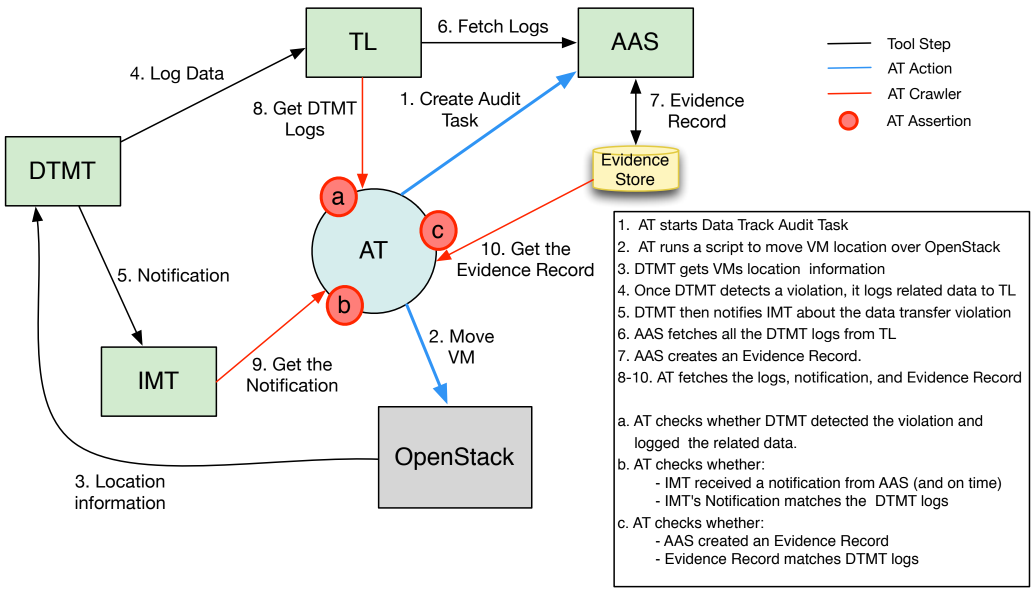
T2 Verify that the *EvidenceRecord* created by AAS corresponds to the log event written by DTMT, and that the timestamps are in close proximity (e.g., violation detected time, violation generated time, violation reported time)

T3 Check whether IMT received a notification from DTMT

T4 Verify that IMT received the notification from DTMT on time.

T5 Verify that the notification received by IMT matches the log event written by DTMT.

These assertions entail the accountability properties P1–P3 given above: P1 follows from T1–T3, P2 from T4 and P3 from T1, T3 and T5.



1. AT starts Data Track Audit Task
2. AT runs a script to move VM location over OpenStack
3. DTMT gets VMs location information
4. Once DTMT detects a violation, it logs related data to TL
5. DTMT then notifies IMT about the data transfer violation
6. AAS fetches all the DTMT logs from TL
7. AAS creates an Evidence Record.
8-10. AT fetches the logs, notification, and Evidence Record

a. AT checks whether DTMT detected the violation and logged the related data.
b. AT checks whether:
- IMT received a notification from AAS (and on time)
- IMT's Notification matches the DTMT logs
c. AT checks whether:
- AAS created an Evidence Record
- Evidence Record matches DTMT logs

Figure 4.9: AT driving data transfer across data centers validation scenario

Figure 4.9 shows the interactions between AT and the other tools to execute this validation scenario. The blue links reflect the actions AT takes to drive the other tools, while the black ones reflect internal interactions between the A4Cloud tools (other than AT). The Figure also shows the information AT receives from the A4cloud tools (red links), and the assertions that AT checks (red bullets).

Listing 4.3 shows the Java representation of the Data Transfer across data centers validation scenario. The tool declaration phases, initialization of the AT and the validation scenario are defined in a way analogous to the previous scenarios. However, the method `moveStorageVolume` may be implemented using a suitable script, or could receive some confirming data from OpenStack after the required action is performed manually through the OpenStack's dashboard.

Listing 4.3: Java representation of the data transfer across data centers scenario

```java
public class DataTransferDataCenterVS {
  //  I. Declare your tools
  AASTool aas = new AASTool();
  DTMTTool dtmt = new DTMTTool();
  IMTTool imt = new IMTTool();
  OpenStackTool openstack = new OpenStackTool();

  //  II. Setup Phase
  public void setUp()
  { RestModule.initializeInfrastructure(); }

  //  III. Validation Scenario
  public void scenario() {
    // a. Initialize the system
    aas.createDataTrackAuditTask();

    // b. Moves a VM to a forbidden location
    openstack.moveStorageVolume();

    // c. Assert!
    ACCOUNT.dataTransferCenters();
  }
}
```

4.2.4 Data transfer across hosts

For this validation scenario, DTMT has to be configured with a policy forbidding data transfer between hosts; DTMT should also be connected to an A-PPLE instance holding personal data. The migration of virtual disks and/or virtual machines to other hosts, a policy violation, is then performed by an OpenStack data transfer operation, either manually on the OpenStack dashboard or by AT running a certain script. Following the violation, DTMT will automatically trigger a policy violation event in the configured A-PPLE instance. A-PPLE will then transfer the incident information to IMT using a specialized client API.

This validation scenario can be seen as a special case of the previous one, and be used to validate analogous accountability properties:

P1 A transparency property: “Cross-machine data transfers are made explicit.”

P2 A responsiveness property: “Data transfer violations are notified in a timely manner.”

P3 A responsibility property: “The responsible entity for data transfer violations is made explicit and information about the source is made available.”

Similar to the previous validation scenario, the metric for Average Timing of Notification (Metric 40) can be reused here.

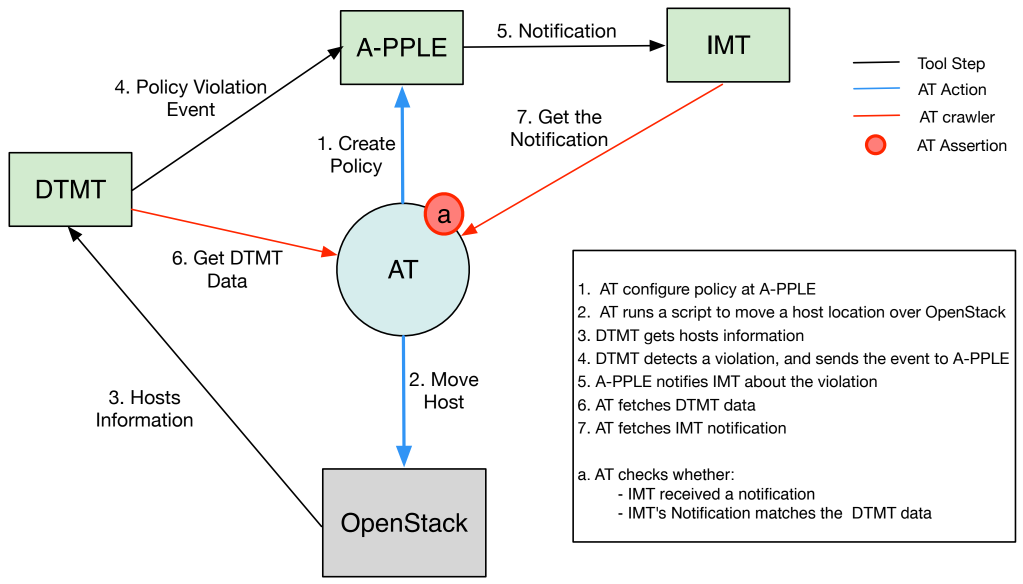
Using this validation scenario, we can assert the following:

T1 Check whether IMT received a notification from A-PPLE

T2 Verify that IMT received the notification from A-PPLE on time.

T3 Verify that the notification received by IMT matches the information in DTMT

These assertions entail the accountability properties P1–P3 given above: P1 follows from T1, T3; P2 from T2 and T3, and P3 from T1 and T3.



1. AT configure policy at A-PPLE
2. AT runs a script to move a host location over OpenStack
3. DTMT gets hosts information
4. DTMT detects a violation, and sends the event to A-PPLE
5. A-PPLE notifies IMT about the violation
6. AT fetches DTMT data
7. AT fetches IMT notification

a. AT checks whether:
- IMT received a notification
- IMT's Notification matches the DTMT data

Figure 4.10: AT driving data transfer across hosts validation scenario

Figure 4.10 shows the tools interactions involved in this scenario (black links), information AT receives from the A4cloud tools (red links), and the assertions that AT checks (red bullets).

Listing 4.4 shows the Java representation of the Data transfer across hosts validation scenario. Similar to the previous scenario `moveDiskToHost` can be implemented by AT running a script that performs the host movement location on the OpenStack configuration or just receiving information confirming that step.

4.2.5 End user notification

In the wearable scenario, there are two companies that can handle incidents: DataSpacer, at the IaaS level, and Kardio-Mon, at the SaaS level. However, the Wearable Co would not necessarily receive the needed information from Kardio-Mon, nor Kardio-Mon from DataSpacer, etc. Furthermore, complicated Cloud provider chains with multiple participants increase the need for more automated sharing of incident information – potentially allowing for some response actions to be automated. IMT receives the detected incidents from DTMT and AAS, and uses A-PPLE to notify end users about incidents that are relevant for them. When a notification of end users occurs, IMT sends a notification to A-PPLE, A-PPLE provides this information to TL that informs the end user about the incident. Depending on the placement of IMT instances on different

Listing 4.4: Java representation of the transfer across hosts scenario

```java
public class DataTransferHostVS {
  //  I. Declare your tools
  AppleTool apple = new AppleTool();
  DTMTTool dtmt = new DTMTTool();
  IMTTool imt = new IMTTool();
  OpenStackTool openstack = new OpenStackTool();

  //  II. Setup Phase
  public void setUp()
  { RestModule.initializeInfrastructure(); }

  //  III. Validation Scenario
  public void scenario() {
    // a. Configure policy
    apple.createPII("policies/locationPolicy.xml");

    // b. Trigger policy violation
    openstack.moveDiskToHost();

    // c. Assert!
    ACCOUNT.dataTransferHosts();
  }
}
```

machines and their interconnection with the other A4Cloud tools, notifications can also be sent between different IMT instances.

The goal of this validation scenario is to validate that the end user notifications are correctly provided and therefore validate the following accountability properties:

P1 A transparency property: “Affected stakeholders are informed about failures, disclosures or breaches.”

P2 A responsiveness property: “Stakeholders are notified in a timely manner.”

P3 A responsibility property: “The responsible entity for notifications is made explicit and their source is made available.”

Since this validation scenario is mainly centered around the notification of end-users, it could be interesting to assess the quality of notifications. The following metrics Coverage of incident notifications (Metric 31) and Type of incident notification (Metric 32) of the Catalog of Accountability Metrics can be used for this purpose.

Metric 31 provides the percentage of privacy incidents and breaches for which affected stakeholders were notified, for a given period of time.

- Input: This metric is computed using the following parameters:
  - N – Number of privacy incidents for which notification exists

  - T – Total number of privacy incidents over a given period

- Formulation and output: Output = $(N/T) \cdot 100$

Metric 32 describes the quality of the notification procedures after a privacy incident or breach.

- Formulation and output: Four levels are defined:
  - Level 0 – No notification of privacy incidents is done, or it is done inconsistently.
  - Level 1 – General notification, usually as a public notice. Affected users may not be aware of the incident
  - Level 2 – Individual notification to each affected user.
  - Level 3 – Automated and self-service procedures for data subject access are in place, including the case of denied access.

Note that the data for evaluating Metric 32 may not be collected automatically, but rather manually by an evaluator (e.g., auditor). Additionally, as in other validation scenarios, the metric for Average Timing of Notification (Metric 40) can be used here as well.

This validation scenario requires to create a new incident (or use an already existing one) either using the GUI of IMT or by a REST call to IMT's API. The sender of the incident should be configured as a provider in IMT prior to the sending, otherwise the incident is not accepted. Then, IMT, initiated by an incident handler, sends user notifications to A-PPLE, possibly indirectly via other IMT instances. In turn A-PPLE will execute the corresponding actions associated in all obligations associated to the user and to the policy violation. That is, A-PPLE notifies subscribers and end user (e.g., about the affected resources, users and a humanly understandable message). A-PPLE also logs all related events to the user TL sender instance and to the AAS instance.

Using this validation scenario, we can assert the following:

T1 Check whether subscribers (to the involved IMTs) received the notification

T2 Check whether subscribers (to the involved IMTs) received the notification in a timely fashion

T3 Check whether users received the notification

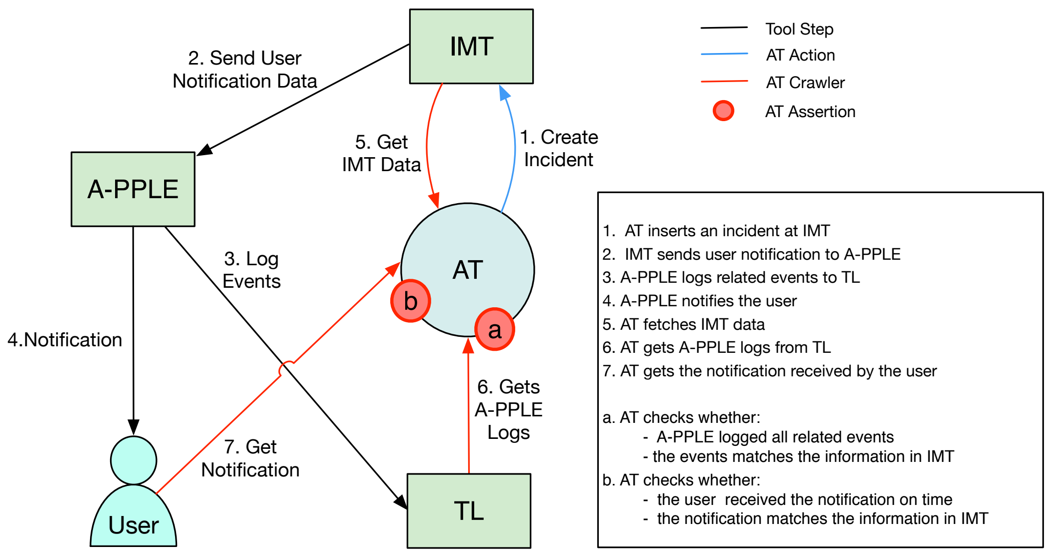
T4 Verify that A-PPLE logged all related events to the user TL sender instance.

T5 Verify that the events from A-PPLE matches the information in IMT.

T6 Verify that the notification received by the users matches the information in IMT.

These tool-specific validations allow the accountability properties P1–P3 from the beginning to be validated: P1 follows from T1, T3-T6, P2 follows from T2, and P3 follows from T4–T6.

Similar to the previous scenarios, Figure 4.11 shows the tools interactions, information gathering, and assertions involved in this scenario. Listing 4.5 shows the Java representation of the End user notification validation scenario.



1. AT inserts an incident at IMT
2. IMT sends user notification to A-PPLE
3. A-PPLE logs related events to TL
4. A-PPLE notifies the user
5. AT fetches IMT data
6. AT gets A-PPLE logs from TL
7. AT gets the notification received by the user

a. AT checks whether:
  - A-PPLE logged all related events
  - the events matches the information in IMT

b. AT checks whether:
  - the user received the notification on time
  - the notification matches the information in IMT

Figure 4.11: AT driving end user notification validation scenario – one IMT instance

Listing 4.5: Java representation of the end user notification validation scenario

```java
public class EndUserNotificationVS {
  //  I. Declare your tools
  IMTTool imt = new IMTTool();
  AppleTool apple = new AppleTool();

  //  II. Setup Phase
  public void setUp()
  { RestModule.initializeInfrastructure();   }

  //  III. Validation Scenario
  public void scenario() {
    // a. Create an incident
    imt.createIncident("incidents/incident.json");

    // b. Assert!
    ACCOUNT.incidentNotification();   }
}
```

4.3 Summary

In this chapter, we have provided an overview of cloud service supply chain and the wearable scenario that has been implemented and deployed in order to demonstrate the different functionalities of the A4Cloud accountability framework. We have then presented the validation scenarios considered for validation in D-6, and which are directly derived from the wearable scenario. We have then reported on the accountability validation of the A4Cloud policy-enforcement tools by executing the validation scenarios using the Assertion Tool.

Chapter 5

Conclusion

In this deliverable we introduced our methodology for validating the A4Cloud toolkit and a framework supporting this methodology. Our methodology is based on validation scenarios that involve particular interactions between the A4Cloud tools. The toolkit's functionalities are validated with respect to a set of accountability properties that have to be respected by compositions of the the different A4Cloud tools.

In Chapter 2, we first provided a review of existing academic approaches and tools for verifying accountability in the Cloud. This review showed that none of these approaches provided the validation functionality that our tools support and that the generality of our tools in terms of the covered accountability properties is unmatched. After a brief discussion how metrics are used as part of the validation methodology, we motivated and defined a set of accountability assertions that allows the declarative definition of accountability properties. The accountability properties can be combined to construct higher-level properties.

In Chapter 3, we defined an assertion framework implementing our validation methodology and offering the means to automate the execution of validation scenario. We introduced the architecture of the two validation tools. First, the Assertion Tool (AT), its core components orchestrating the functionalities of our framework, its features and implementation, as well as the different APIs provided by the framework. We have shown, in particular, how the Assertion tool can be extended to accommodate new tools, in particular, accountability-related tools external to the A4Cloud project. Second, we also presented the monitoring tool AccMon, which provide the means to monitor accountability policies in the context of a real system, and we discussed the potential integration between the AT and AccMon in order to take the advantages of the different functionalities provided by the two tools. Finally, we presented a brief overview of the A4Cloud tools that we have subjected to validation, their interactions, and the format of the data flows, notably those

In Chapter 4, we presented the wearable scenario and the setup of a Cloud service provider chain. Then we defined a set of validation scenarios, which are related to the global wearable scenario, and show how AT can drive A4Cloud tools in order to execute these validation scenarios and validate the related assertions.

The results of the validation workpackage pave the way to multiple extensions as well on the methodological level as well as the tool level. These include the extension of the set of low-level accountability predicates, for example in order to integrate new external accountability tools in the framework. In the long run, the development of a more direct representation of high-level accountability properties constitutes a worthwhile endeavor. On the tool level, a tighter integration of the Assertion and AccMon tools is useful. A first step to this end, is the reuse of the accountability predicate library by AccMon. In the long run, the runtime exchange of information between both tools is an important goal.

Appendix A

Examples of Evidence Records and Notifications

In this appendix, we provide examples for the data formats of some of the interactions between A4Cloud tools, which are discussed in Section 3.4.3.

Listing A.1 shows a snapshot of a violation occurred at machine named *'Main-Container'*, and collected by an agent named *DataRetentionPolicyEvaluationAgent*.

Listing A.1: A snapshot of a violation occurred at machine "Main-Container"

```
<evidenceContainer> <record id="1">
 <action>SnapshotExists(1)_6b0a9a97-0ca7-4fa1-9d68-0714fe3b03c2
 (Kardio-Mon-PII-Store)@Main-Container(10.0.0.6)</action>
 <actor>PiiSnapshotCheckAgent_1042_Main-Container@AAS_Core_Container
 </actor>
 <policyID>1042</policyID>
 <supportingElements elementID="0">
  <signature>uAE5vrQsJJIj2anVKxtj4fmoKmeeovxBdYF5879eql8=</signature>
  <element xsi:type="xs:string" xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">NovaImage{ ... }
       </element> </supportingElements>
 <supportingElements elementID="1">
  <signature>M29rUHtjhAjzWdmn/9kw+gUxbxTysHkMDVCdPJC7akU=</signature>
  <element xsi:type="xs:string" xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">Record UUID:
      d32315e9-9328-4ee1-9d5e-9f223d06261d
  </element> </supportingElements>
 <evidenceMetaData>
   <collectingInstance>PiiSnapshotCheckAgent_1042_Main-Container
   </collectingInstance> </evidenceMetaData>
</record> </evidenceContainer>
```

Listing A.2: A notification sent by IMT to cloud customer

```
{ "id":"119ae094-7c33-4070-92c7-716437b37a31",
 "type":{ "id":"2bee3378-a034-4bef-a080-d23b10579972",
  "name":"Reduced availability",
  "endpoint":"https://test.com/api/1.0/receive",
  "incidents":[  { "id":"36ece42d-aa4f-41cf-a12d-27c9295f5a64",
     "name":"Serious DDoS",
      "type":{ "id":"e5f022a6-8372-464c-9c2a-71dd031c6d08",
       "name":"DDoS",
       "description":"A large number of connections with the goal of
           making the page unavailable", "consequence":0.8 },
       "triggers":[ { "id":"a6dadd88-ebb5-454b-9b12-9d94f6dd3e4d",
       "type":{ "id":"72597fca-8032-4793-b5f2-cbd0abd87f4c",
       "name":"Ping time",
       "description":"The time it takes a package to round trip
           between the service in question and the measuring instance.
            Threshold is measured in milliseconds", "comparators":[
           ">", "<" ] }, "method":"NONE", "threshold":14, "comparator"
           : ">"
             }]}] },
  "generation_time":"2015-05-06 14:01:46Z",
  "sent_time":"2015-05-06 14:01:55Z",
  "sender":"434c693d-dff6-446c-923e-7a1c738d3d50",
  "hmac":"HMAC",
  "incidents":[ {  ...  }]
}
```

Listing A.2 shows a notification sent by IMT to cloud customer subscribes about incident types. The data format used is similar to the one used for the notification by AAS to IMT.

Listing A.3 shows an example of notification from AAS to IMT about a Data Retention policy violation. This message contains the generated evidence record which describes the occurred incident. The `custom_fields` contain the actual evidence - e.g., the PII store and delete messages as well as proof of an existing snapshot which includes the data already deleted by A-PPLE.

Listing A.3: A notification from AAS to IMT

```
{ "id" : "4fae8932-bc84-46c5-b371-9a25af975e64",
"type" : { "id" : "73c2af7b-31b4-44a0-b2cd-3f2551715600",
  "name" : "PII snapshot exists notification",
  "endpoint" : "http://10.0.0.8:8800",
  "incidents" : [ { "id" : "697a57c0-827e-40dd-a4fa-68dae13b44b4",
    "name" : "PII Data Retention Violation",
    "type" : { "id" : "9e8b2d47-1c14-402b-9236-cb88d57f4180",
      "name" : "PII Data Retention Snapshot Violation",
      "description" : "PII should be completely deleted but still
          exists in snapshot", "consequence" : 0.5 },
    "triggers" : [ { "id" : "668c53b4-3a9c-4364-8a43-654e37f5b396",
      "threshold" : "1", "comparator" : "?", "method" : "None",
      "type" : { "id" : "a68455f0-3357-4df8-b27e-ff3a4af0792a",
        "name" : "PII snapshot exists",
        "description" : "PII should be completely deleted but still
            exists in snapshot", "comparators" : [ "?" ] } } ] } ] },
"generation_time" : "2016-02-22 10:27:00.813",
"sent_time" : "2016-02-22 10:27:00.821",
"sender" : "60c36dd3-2df6-4c60-b083-1b94878af90b", "hmac" : "hmac",
"incidents" : [ { "id" : "dd73e2f9-6e62-4676-aac8-07442f82ff06",
"parent" : { "id" : "da2d909d-6704-4344-83e6-828bc0d04059",
"provider" : "DataRetentionPolicyEvaluationAgent_1042_Main-Container",
     "endpoint" : "141.28.98.114" },
"type" : { "id" : "2186237b-5cb6-4618-8fee-381f0cd658f7",
"name" : "PII Data Retention Snapshot Violation",
"description" : "PII should be completely deleted but still exists in
    snapshot", "consequence" : 0.5 },
"liaison" : { "id" : "03725dbd-1ef7-42ad-8a74-724e31851518",
"name" : "AAS", "email" : "aas@hfu.de", "phone" : "555 5555555",
"address" : "Robert-Gerwig-Platz 1", "zip" : "79117", "city" : "
    Furtwangen"},
"language" : "en_US", "status" : "Unresolved", "impact" : 0.6,
"summary" : "A website can be accessed using unencrypted communication
     (http)",
"description" : "The existence of this PII snapshot violates the data
    retention policy (ID: 1042)",
"detection_time" : "2016-02-22 10:27:00.625",
"occurrence_time" : "2016-02-22 10:27:00.625", "attachments" : [ ],
"custom_fields" : [ { "id" : "28811f6b-da1b-4054-b2aa-f7eef0539173",
 "value" : "<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <evidenceContainer> ... </evidenceContainer>",
        "type" : { ... } },
     ...  } ] }
```

Bibliography

[BB13a] Reza Babaee and Seyed Morteza Babamir. Runtime verification of service-oriented systems: a well-rounded survey. *International Journal of Web and Grid Services*, 9(3):213–267, 2013. PMID: 55699.

[BB13b] Reza Babaee and Seyed Morteza Babamir. Runtime verification of service-oriented systems: A well-rounded survey. *Int. J. Web Grid Serv.*, 9(3):213–267, August 2013.

[BGRS15] Walid Benghabrit, Hervé Grall, Jean-Claude Royer, and Mohamed SELLAMI. Abstract Accountability Language: Translation, Compliance and Application. In *ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE*, ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE, New Delhi, India, December 2015.

[BKV15] Andreas Bauer, Jan-Christoph Kster, and Gil Vegliach. The ins and outs of first-order runtime verification. *Formal Methods in System Design*, 46(3):286–316, 2015.

[BLS11] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, September 2011.

[BM14] Denis Butin and Daniel Métayer. *FM 2014: Formal Methods: 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, chapter Log Analysis for Data Protection Accountability, pages 163–178. Springer International Publishing, Cham, 2014.

[CEH$^+$05] R. Corin, S. Etalle, J. Hartog, G. Lenzini, and I. Staicu. *Formal Aspects in Security and Trust: IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST), World Computer Congress, August 22–27, 2004, Toulouse, France*, chapter A Logic for Auditing Accountability in Decentralized Systems, pages 187–201. Springer US, Boston, MA, 2005.

[DFK06] Anatoli Degtyarev, Michael Fisher, and Boris Konev. Monodic temporal resolution. *ACM Transactions on Computational Logic*, 7(1):108–150, January 2006.

[dOSoTP$^+$15a] Anderson Santana de Oliveira, Jakub Sendor, Võ Thành Phúc, Efthymios Vlachos, Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen, Walid Benghabrit, Jean-Claude Royer, Michela D'Errico, and Martin Gilje Jaatun. D:D-3.2:Prototype for accountability enforcement tools and services. Technical Report D:C-4.2, Accountability for Cloud and Future Internet Services - A4Cloud Project, 2015.

[dOSoTP$^+$15b] Anderson Santana de Oliveira, Jakub Sendor, Võ Thành Phúc, Efthymios Vlachos, Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen, Walid Benghabrit, Jean-Claude Royer, Michela D'Errico, Martin Gilje Jaatun, and Inger Anne Tøndel. D:D-3.2 Prototype for accountability enforcement tools and services. Technical Report D:D-2.3, Accountability for Cloud and Future Internet Services - A4Cloud Project, 2015.

[dOSP$^+$15] Anderson Santana de Oliveira, Jakub Sendor, Võ Thành Phíc, Stefan Berthold, Siani Pearson, Massimo Felici, Erdal Cayirci, Efthymios Vlachos, Monir Azraoui, Kaoutar Elkhiyaoui, Antorweep Chakravorty, Tomasz Wiktor Wlodarczyk, Melek Önen, Walid Benghabrit, Jean-Claude Royer, Michela D'Errico, Martin Gilje Jaatun, and Inger Anne Tondel. D:C-3.2: Prototype for accountability enforcement tools and services. Technical Report D:C-3.2, Accountability for Cloud and Future Internet Services - A4Cloud Project, 2015.

[EM11] Daniel Le Métayer Eduardo Mazza, Marie-Laure Potet. A formal framework for specifying and analyzing logs as electronic evidence. *Formal Methods: Foundations and Applications*, 6527:194–209, 2011.

[Fid88] C. J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. *Proceedings of the 11th Australian Computer Science Conference*, 10(1):5666, 1988.

[FP$^+$14] Massimo Felici, Siani Pearson, et al. Conceptual framework. Technical Report D:C-2.1, A4Cloud, September 2014.

[GBT11] Jerry Gao, Xiaoying Bai, and Wei-Tek Tsai. Cloud testing-issues, challenges, needs and practice. *Software Engineering: An International Journal*, 1(1):9–23, 2011.

[GI09] N. Gruschka and L. L. Iacono. Vulnerable cloud: Soap message security validation revisited. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 625–631, July 2009.

[GKP$^+$15a] Frederic Gittler, Theo Koulouris, Siani Pearson, Rehab Alnemr, Jean-Claude Royer, Michela D Errico, Anderson Santana De Oliveira, Thomas Rubsamen, Philipp Ruf, Christoph Reich, Mehdi Haddad, Tobias Pulls, Carmen Gago, Christian Froystad, and Lorenzo Dalla Corte. Annex to d:d-2.ab: The a4cloud toolkit. Technical report, The A4Cloud project, October 2015.

[GKP$^+$15b] Frederic Gittler, Theo Koulouris, Siani Pearson, Vasilis Tountopoulos, Mehdi Haddad, Melek Önen, Richard Mark Brown, Jesus Luna, Alain Pannetrat, Jean-Claude Royer, Mohamed Sellami, Monir Azraoui, Kaoutar Elkhiyaoui, Niamh Gleeson, Asma Vranaki, Anderson Santana De Oliveira, Karin Bernsmed, Carmen Gago, and David Núñez. D:D-2.3: Initial reference architecture. Technical Report D:D-2.3, Accountability for Cloud and Future Internet Services - A4Cloud Project, 2015.

[GMM06] Alexandre Genon, Thierry Massart, and Cédric Meuter. Monitoring distributed controllers: When an efficient ltl algorithm on sequences is needed to model-check traces. In *Proceedings of the 14th International Conference on Formal Methods*, FM'06, pages 557–572, Berlin, Heidelberg, 2006. Springer-Verlag.

[GP10] Alwyn Goodloe and Lee Pike. Monitoring distributed real-time systems: A survey and future directions, 2010.

[HARD10] Andreas Haeberlen, Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel. Accountable virtual machines. In Remzi H. Arpaci-Dusseau and Brad Chen, editors, *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, pages 119–134. USENIX Association, 2010.

[HyT] HyTrust. `http://www.hytrust.com/`.

[IEE90] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.

[KTV10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 526–535. ACM, 2010.

[LHKR08] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Authenticated index structures for outsourced databases. In Michael Gertz and Sushil Jajodia, editors, *Handbook of Database Security - Applications and Trends*, pages 115–136. Springer, 2008.

[Mat89] Friedemann Mattern. Virtual time and global states of distributed systems. In M. Cosnard et. al., editor, *Parallel and Distributed Algorithms: proceedings of the International Workshop on Parallel & Distributed Algorithms*, pages 215–226. Elsevier Science Publishers B. V., 1989.

[MLK14] Fatma Masmoudi, Monia Loulou, and Ahmed Hadj Kacem. Multi-tenant services monitoring for accountability in cloud computing. In *IEEE 6th International Conference on Cloud Computing Technology and Science, CloudCom 2014, Singapore, December 15-18, 2014*, pages 620–625. IEEE Computer Society, 2014.

[MNP$^+$11] Philippe Massonet, Syed Naqvi, Christophe Ponsard, Joseph Latanicki, Benny Rochwerger, and Massimo Villari. A monitoring and audit logging architecture for data location compliance in federated cloud infrastructures. In *IPDPS Workshops*, pages 1510–1517. IEEE, 2011.

[MSLK15] Fatma Masmoudi, Mohamed Sellami, Monia Loulou, and Ahmed Hadj Kacem. Analyzing multi-tenant cloud services' accountability. In Yinsheng Li, Xiang Fei, Kuo-Ming Chao, and Jen-Yao Chung, editors, *12th IEEE International Conference on e-Business Engineering, ICEBE 2015, Beijing, China, October 23-25, 2015*, pages 239–244. IEEE Computer Society, 2015.

[NFGA$^+$14] David Núñez, Carmen Fernandez-Gago, Isaac Agudo, Jesus Luna, and Alain Pannetrat. Validation of the accountability metrics. Technical Report D35.2, Accountability for Cloud and Future Internet Services - A4Cloud Project, 2014.

[NSG$^+$15] Athanasios Naskos, Emmanouela Stachtiari, Anastasios Gounaris, Panagiotis Katsaros, Dimitrios Tsoumakos, Ioannis Konstantinou, and Spyros Sioutas. Dependable horizontal scaling based on probabilistic model checking. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015*, pages 31–40. IEEE Computer Society, 2015.

[NSKG15] Athanasios Naskos, Emmanouela Stachtiari, Panagiotis Katsaros, and Anastasios Gounaris. Probabilistic model checking at runtime for the provisioning of cloud resources. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, volume 9333 of *Lecture Notes in Computer Science*, pages 275–280. Springer, 2015.

[PTC$^+$12] S. Pearson, V. Tountopoulos, D. Catteddu, M. Sudholt, R. Molva, C. Reich, S. Fischer-Hubner, C. Millard, V. Lotz, M.G. Jaatun, R. Leenes, Chunming Rong, and J. Lopez. Accountability for cloud and other future internet services. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 629–632, 2012.

[SS14] T. Scheffel and M. Schmitz. Three-valued asynchronous distributed runtime verification. In *Formal Methods and Models for Codesign (MEMOCODE), 2014 Twelfth ACM/IEEE International Conference on*, pages 52–61, Oct 2014.

[SSL12] Smitha Sundareswaran, Anna Cinzia Squicciarini, and Dan Lin. Ensuring distributed accountability for data sharing in the cloud. *IEEE Trans. Dependable Sec. Comput.*, 9(4):556–568, 2012.

[SVA$^+$06] Koushik Sen, Abhay Vardhan, Gul Agha, , and Grigore Roşu. Decentralized runtime analysis of multithreaded applications. In *NSF Next Generation Software Program Workshop (NSFNGS'06) (Satellite Workshop of IPDPS'06)*. IEEE Digital Library, 2006. (To Appear).

[SVAR04] Koushik Sen, A. Vardhan, Gul Agha, and G. Rosu. Efficient decentralized monitoring of safety in distributed systems. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, pages 418–427, May 2004.

[Tes] Emulab Network Emulation Testbed. www.emulab.net.

[TNR11] Slim Trabelsi, Gregory Neven, and Dave Raggett. Report on design and implementation. Technical Report D5.3.4, Primelife Project, 2011.

[TRAR$^+$15] Christoph Reich Thomas Ruebsamen, Monir Azraoui, Jean-Claude Royer, Jenni Reuben, Tobias Pulls, Karin Bernsmed, and Massimo Felici. D:C-8.2:Framework of Evidence. Technical Report D:C-8.2, Accountability for Cloud and Future Internet Services - A4Cloud Project, 2015.

[WZ10] Chen Wang and Ying Zhou. A collaborative monitoring mechanism for making a multitenant platform accountable. In Erich M. Nahum and Dongyan Xu, editors, *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010*. USENIX Association, 2010.

[YCW$^+$10] Jinhui Yao, Shiping Chen, Chen Wang, David Levy, and John Zic. Accountability as a service for the cloud. In *2010 IEEE International Conference on Services Computing, SCC 2010, Miami, Florida, USA, July 5-10, 2010*, pages 81–88. IEEE Computer Society, 2010.

[YCW$^+$12] Jinhui Yao, Shiping Chen, Chen Wang, David Levy, and John Zic. Accountability services for verifying compliance in the cloud. *IJCC*, 1(2/3):240–260, 2012.