

D:C-4.2 Policy representation and enforcement techniques

Deliverable Number: D:34.2 (D:C-4.2)

Work Package: WP:C-4

Version: Final

Deliverable Lead Organisation: EURECOM

Dissemination Level: PU

Contractual Date of Delivery (release): 30/09/2014

Date of Delivery: 30/09/2014

Status: Delivered

Editor

Melek Önen (EURECOM)

Contributors

Monir Azraoui (EURECOM), Walid Benghabrit (Armines), Karin Bernsmed (SINTEF), Kaoutar Elkhyaoui (EURECOM), Hervé Grall (Armines), Anderson Santana de Oliveira (SAP), Melek Önen (EURECOM), Jean-Claude Royer (Armines), Mohamed Sellami (Armines), Jakub Sendor (SAP)

Reviewers

Michela D'Errico (HP), Tobias Pulls (KAU), Christoph Reich (HFU)

Executive Summary

The A4Cloud project aims at increasing trust in cloud services and hence bringing solutions to provide greater transparency on the handling of personal and business confidential data. In order to render cloud stakeholders accountable for their actions, an accountability framework should primarily identify the relevant accountability obligations and provide means to express them into machine-readable policies.

The A4Cloud policy representation framework described in the previous deliverable D34.1 defines two different languages namely AAL (Abstract Accountability Language) and A-PPL (Accountability PPL) which map accountability obligations at different levels: while the machine-understandable A-PPL language extends the already existing PPL language with new features required by accountability obligations (focusing mostly on obligations related to data handling and data protection), AAL is human-readable and thus allows any cloud stakeholder including the data subject to express policy rules and/or user preferences.

This second deliverable aims at describing the A4Cloud policy framework entirely by first analyzing the correctness of AAL policies which are the result of the matching of user (data subject) preferences and the data controller's policy. AAL policies are further translated into A-PPL in a semi-automated way. Finally, the deliverable suggests some methodologies and solutions for the enforcement of different A-PPL policy rules. Most of data handling and some of the rules related to logging are enforced with the help of a new enforcement engine named as A-PPLE and implemented within work package D-3. We also propose some additional enforcement solutions which can either be integrated within A-PPLE or implemented independently. These solutions are classified into two categories following the classification of the accountability framework: while preventive solutions implement data confidentiality and some privacy enhancement technologies, the proposed detective solutions target the problem of secure logging.

With this aim, the deliverable is organized as follows:

- Some refinements of both languages are presented in chapter 1.
- In chapter 2 we first introduce the notion of accountable component design which assures the correctness and consistency of different AAL rules and further describe some means to translate the final AAL policy into A-PPL.
- In chapter 3 both languages are demonstrated with obligations deriving from Business Use Cases defined under work package B-3 and with some particular data transfer rules defined by work package B-5.
- Chapter 4 presents different enforcement methodologies and solutions to achieve policy compliance. In this chapter, the A-PPL Engine (A-PPLE) which mainly enforces data handling and corresponding logging rules is first introduced. Some additional enforcement mechanisms are further proposed.

Contents

1. A4Cloud Policy Languages	7
1.1. Refinement of AAL	7
1.2. Refinement of A-PPL	12
2. From Abstract Clauses to Concrete Policies	17
2.1. Accountable Component Design	17
2.2. Towards a rule-based system to transform AAL clauses into A-PPL policies . . .	26
3. Mapping Obligations in AAL and A-PPL	33
3.1. Application to an A4Cloud Use Case	34
3.2. Application to Data Transfer Laws	44
4. Policy Enforcement Methodologies	50
4.1. Policy Enforcement Approaches	50
4.2. Approach taken in A4Cloud: A-PPLE	52
4.3. Suggested mechanisms to enforce accountability obligations	56
4.4. Summary	62
Appendix A. List of Triggers and Actions in A-PPL	69
Appendix B. From AAL to A-PPL Examples	69
B.1. Rule 1: AAL expression with a PERMIT authorization	69
B.2. Rule 2: AAL expression with a DENY authorization	70
B.3. Rule 4: a conjunctive set of AAL expressions (AND)	71
B.4. Rule 5: a disjunctive set of AAL expressions (OR)	74
Appendix C. A-PPL Examples from the Use Cases	76
Appendix D. Model for Data Transfer Regulation	80
D.1. Data Transfer Rules	80
D.2. Example of A-PPL rules for data transfer	83
Appendix E. From AAL to A-PPL: the final A-PPL policy	85

Introduction

Accountability attributes and obligations

In work package C-2, the accountability attributes defined within the conceptual framework are reminded below:

- “**Observability** is a property of an object, process or system which describes how well the internal actions of the system can be described by observing the external outputs of the system.
- **Verifiability** is a property of an object, process or system that its behaviour can be verified against a requirement or set of requirements.
- **Attributability** is a property of an observation that discloses or can be assigned to actions of a particular actor (or system element).
- **Transparency** is a property of an accountable system that it is capable of giving account of, or providing visibility of, how it conforms to its governing rules and commitments.
- **Responsibility** is the state of being assigned to take action to ensure conformity to a particular set of policies or rules.
- **Liability** is the state of being legally obligated or responsible.
- **Remediability** is the state (of a system) of being able to correct faults or deficiencies in the implementation of a particular set of policies and rules and/or providing a remedy to a party, if any, harmed by the deficiency.”

Machine-readable policies are considered to interpret these accountability attributes via obligations. Regulatory obligations currently derive from the EU Data Protection Directive 95/46/EC and usually reflect normative ones. Such obligations mainly define the relationships between different actors. Contractual obligations (deriving from Service Level Agreements (SLAs), Use of Terms, etc.) usually apply regulatory obligations to specific situations and provide more details both on the actors and the relevant actions. For example, a cloud customer can require the storage of its data in a specific region. In deliverable D34.1 [ABC⁺13], we proposed an initial set of accountability policy requirements and classified them as either data handling or accountability. Accountability requirements are the ones that are specific to accountability and that were not addressed by existing policy languages such as PPL [ABDCDV⁺09]. Following these requirements, the interaction with the C-2 and B-3 work packages, we identify that an accountability policy should include three classes of obligations:

- **Data Handling Obligations:** Such obligations include privacy constraints, access and usage control rules: In addition to defining the actors authorized to access some data via a set of attributes (name, role, purpose, etc.), an accountability policy should express additional rules on the actions taken on the data; such actions include the sharing to third parties, usage for a particular purpose, data deletion or data anonymization. The

new accountability language should also express rules about data retention and data location. Such data handling rules should be as specific as possible in order to reflect the seven accountability attributes. The proposed A4Cloud policy language expresses requirements corresponding to the **responsibility** attribute by capturing all cloud roles defined within the C-2 work package.

- **Logging/Monitoring Obligations:** In order to verify the compliance with data handling obligations, accountable services may be audited. The accountability policy language should specify what information is targeted by an audit and which evidence should be collected. Logs are defined as one of the most important sources of evidence; hence, the policy language must specify which events have to be logged and what information related to the logged event have to be added in the log. Additionally, the policy language should enable the sending of notifications to data subjects and third parties on any event. Such obligations contribute to the implementation of **transparency**, **observability**, **verifiability** and **attributability** attributes.
- **Incident Management Obligations:** The policy language should express the set of actions/penalties that need to be taken to handle and/or recover failures. Such policy rules mostly capture requirements originating from the **remediability** and **liability** attributes.

Such a classification is also in accordance with the accountability framework defined in MSC2.3 [DFG⁺14]:

- **preventive mechanisms** will ensure the privacy and security controls on personal data based on the data handling rules expressed in a policy;
- **detective mechanisms** will follow the logging/monitoring rules in order to identify any unexpected event such as unauthorized access or unauthorized data transfer;
- **corrective mechanisms** will apply the incident management rules to take appropriate actions upon the occurrence of an undesired incident. Notice that not all corrective mechanisms can be executed in an automated way.

The A4Cloud Policy Framework

Deliverable D34.1 [ABC⁺13] describes the A4Cloud policy representation framework which defines two different policy languages:

- a high-level language defined as AAL (Abstract Accountability Language) which is simple and therefore human readable. Such a language can be used by any cloud actor including data subjects to define their preferences.
- a machine-understandable language defined as A-PPL (Accountable PPL) which will help service providers to deploy automatic enforcement of accountability policies when they process personal data. This new policy language is an extension of the existing PPL [ABDCDV⁺09] language.

In this new and final deliverable of work package C-4, we first present some refinements of these two languages and further suggest some means to translate AAL rules into A-PPL rules. Before such a translation, we also propose some methods to verify the correctness and consistency of AAL clauses including the matching between data subject preferences and the cloud provider's policy. We further demonstrate the mapping of obligations into these languages using the obligations defined for the A4Cloud business use cases in work package B-3. We additionally analyze personal data transfer rules and suggest to map some of them into AAL and A-PPL.

We finally suggest different enforcement solutions that would fulfill most of the obligations mapped into A-PPL: We first describe the A-PPL engine (A-PPLE) which mainly enforces data handling obligations and partially logging obligations; We provide some discussion on the design of this particular engine and further suggest some additional mechanisms (preventive and detective) that may also be considered as potential enforcement solutions.

1. A4Cloud Policy Languages

1.1. Refinement of AAL

In the previous deliverable [ABC⁺13], we presented the main concepts and a lightweight grammar of an accountability policy language called AAL (Abstract Accountability Language). AAL allows to define accountability clauses, with emphasis on data usage and privacy concerns. The language allows to express authorizations and also actions and temporal modalities. In this deliverable we tuned the language and we introduced some major evolutions (i.e declarations, authorizations, etc), based on our experiments, reviewers' feedback and fixed some issues revealed by the language semantics, but the main foundations remain the same. In the following section, we present these evolutions and future work perspectives on AAL. Note that all examples in this section refer to the health-care use case presented in deliverable [BFO⁺13]. A summary of the changes are:

- The MAY and other authorizations are replaced by specific keywords denoting permission (PERMIT) and prohibition (DENY). The objective is to make more clear the semantics of these constructions while keeping a pure temporal logic interpretation.
- The usage of quantifiers is extended allowing to cover more complex examples and particularly linking in a more general way the audit and the rectification steps.
- We make the declarations of types and services more explicit.
- We did some minor syntactic modifications to improve readability and writing.

1.1.1. AAL kernel

In the following, we explain the AAL constructs and illustrate them with examples. We present in Listing 1 the syntax of the AAL kernel.

Listing 1: AAL Syntax

```

1 AALprogram      ::= Declaration* Clause*
2 Declaration     ::= AgentDec | ServiceDec | DataDec
3 AgentDec        ::= AGENT Id TYPE('Type*') REQUIRED('service
   *') PROVIDED('service*')
4 ServiceDec      ::= SERVICE Id TYPE('Type*') [PURPOSE Id]
5 DataDec         ::= DATA Id TYPE('Type*') SUBJECT agent
6 Clause          ::= CLAUSE Id ':' [Quant*] Usage [Audit
   Rectification]
7 Usage           ::= ActionExp
8 Audit           ::= AUDITING [ActionExp THEN] agent.audit '['
   agent'] '()'
9 Rectification   ::= IF_VIOLATED_THEN ActionExp
10 ActionExp      ::= Action | NOT ActionExp | Modality ActionExp
   | Author | Condition
11               | ActionExp (AND|OR|ONLYWHEN) ActionExp | IF
   ActionExp THEN ActionExp
12 Exp            ::= Variable | Constant | Variable.Attribute

```

```

13 Condition      ::= [NOT] Exp | Exp ['==', | '!='] Exp |
    Condition (AND|OR) Condition
14 Author         ::= (PERMIT | DENY) Action
15 Action         ::= agent.service '['[agent]']', '('Exp')', [Time
    ] [Purpose]
16 Quant          ::= (FORALL | EXISTS) Var [WHERE Condition]
17 Variable       ::= Var ':' Type
18 Modality       ::= MUST | MUSTNOT | ALWAYS
19 Type, var, val, attr Id, agent, Constant, Purpose ::= literal

```

An AAL program is divided in two parts : declarations and clauses (*line 1*). Declarations allow you to specify different actors involved in the cloud system. Each agent has an id, a type, a set of services that it can use (i.e required services) and a set of services that it provides (i.e provided services). In the following example we declare a data subject called Kim, which can use sensors service and offers get service.

```

// Declaring an agent (line 3)
AGENT Kim TYPE(Subject) REQUIRED(sensors) PROVIDED(get)

```

A service is defined by an id, a set of types and a purpose specifying the context of its usage.

```

// Declaring a service (line 4)
SERVICE sensors TYPE() PURPOSE medical

```

In the same way, data is defined by an id, a set of types and a subject

```

// Declaring data (line 5)
DATA KimData TYPE(PLevel3) SUBJECT(Kim)

```

More details on the type system can be found in section 1.1.2.

We define an accountability clause as a triplet (*uc*, *aa*, *rc*), where *uc* is the usage control, *aa* and audit action and *rc* a rectification clause. The informal meaning of such clause is: “Do the best to ensure usage control (*uc*), and if a violation of the usage is observed by audit (*aa*) then the rectification (*rc*) is applied”. The clause can be parameterized by a set of quantified variables (line 6).

- Usage control: it is composed by a combination of actions, condition applied to variables (line 13) and authorizations (line 14). An action has the following pattern `agent1.serviceX[agent2](args)` (line 15) which means that the agent *agent1* uses the service *serviceX* of the agent *agent2* on the data *args*. The authorizations are managed with two keywords PERMIT for a permission and DENY for a prohibition, these authorizations are applied to an action (line 15).

```

// Usage clause for the obligation "Kim's right to access and modify
    his data"
PERMIT Kim.read[hospital](KimData) AND PERMIT Kim.write[hospital](
    KimData)

// Usage clause for the obligation "The hospital must log Kim's data
    access"
Kim.read[hospital](KimData) THEN MUST hospital.log[]()

// Usage clause for the obligation "The hospital may collect Kim's
    data only when Kim send his consent"

```

```
PERMIT hospital.collect[Kim](KimData) ONLYWHEN Kim.consent[hospital]
("OK")
```

- Audit: introduced by the AUDITING keyword, this part is characterized by an explicit audit action (line 8).

```
// Specifying that leslie audit the hospital
AUDITING leslie.audit[hospital]()
```

- Rectification: introduced by the IF_VIOLATED_THEN keyword, the usage part described in the rectification is executed when the usage part of the clause is violated.

```
// An example of a rectification
IF_VIOLATED_THEN
    MUST (leslie.sanction[hospital]() // Sanctioning the violator
        AND leslie.revokeAuthorisation[hospital]("...")) //
        Updating policy
```

In the following, we explain several points on the language constructs in order to clarify their use.

The usage of authorizations. The authorizations may be conflicting with modalities. Unlike the previous version of AAL where the interpretation of the MAY modality was confusing, now we distinguish between the permission of an action and the time when the action really occurs. For instance, if we want to write the obligation "The hospital must log Kim's data access", with the MAY operator we write it as follows:

```
MAY Kim.read[Kim](KimData) THEN MUST hospital.log("Data␣Access")
```

Following the same idea, one would write it with permission :

```
// The wrong way
IF PERMIT Kim.read[hospital](KimData) THEN MUST hospital.log("Data␣Access
")
```

Even if this expression is syntactically correct, the semantic is not consistent with the meaning of the obligation. Indeed, it means when a permission event (for the read action of the hospital on KimData) occurs, then the hospital must log a data access. Thus, this obligation should be written as follows :

```
// The correct way
PERMIT Kim.read[Kim](KimData)
IF Kim.read[Kim](KimData) THEN MUST hospital.log("Data␣Access")
```

Note that we intentionally allow the mixing of the authorization actions using the IF_THEN expression; for instance; consider the following obligations :

1. Hospitals must log all emitted permissions

```
IF PERMIT Kim.read[hospital](KimData) THEN
    MUST hospital.log("Permission␣emitted")
```

2. Hospitals may send Kim's data to a third party only for research purposes. If Kim's data are sent for another purpose, revoke the hospital's authorization.

```
// Revoking authorizations example
PERMIT hospital.send[thirdParty](KimData) PURPOSE research
AUDITING leslie.audit[hospital](KimData)
IF_VIOLATED_THEN DENY hospital.send[thirdParty](KimData)
```

Another minor point is the difference between the DENY and MUSTNOT operators. It is different to prohibit an action from not doing this action. This corresponds to two different uses: as an authorization or a non-action. The first difference is that DENY defines a prohibition (not a permission) while MUSTNOT effectively prohibits an action or a certain combination of actions. There is also a syntactic difference: the MUSTNOT operator can be applied on a complex action expression : MUSTNOT (action1 THEN action2), whereas DENY (action1 THEN action2) is syntactically and semantically incorrect.

The usage of Quantifiers. Here, we explain how to use the quantifiers for variables. We introduce the quantifiers in AAL language in order to have more expressiveness. Since we have a gap between natural language and logic, some points may not be trivial. Consider the following obligation : "Kim has the right to read all his data, and when Kim reads his data the hospital must log the action". Thinking in natural language, one would write :

```
// Allow Kim to read all his data
FORALL d:Data WHERE d.subject = Kim
PERMIT Kim.read[hospital](d)
IF Kim.read[hospital](d) THEN MUST hospital.log('DataAccess')
```

Since AAL is also a formal language, the previous usage expression is interpreted as follows : "if Kim reads ALL his data at the same time then the hospital must log the action". But the obligation specifies that the hospital has to log any read action. Therefore, the correct form is :

```
// Allow Kim to read all his data
FORALL d:Data WHERE d.subject = Kim
PERMIT Kim.read[hospital](d)
EXISTS d2:Data WHERE d2.subject = Kim
IF Kim.read[hospital](d2) THEN MUST hospital.log('DataAccess')
```

Furthermore, we need to have two quantifiers, the first one FORALL is applied on the PERMIT to allow Kim to access all his data, and the second one EXISTS is used in usage expression. The semantics of this usage expression with EXISTS operator is : "if Kim reads one of his data, the hospital must log the action".

1.1.2. AAL Type system

We introduce a small type system in AAL, for two reasons: firstly, a type system gives more power to the semantics and allows to perform more precise checks; the second reason is to improve the language usability. The AAL type system extension allows a user to define his own types.

Listing 2: AAL Type system

```
1 TypesDec      ::= TYPE Id [EXTENDS Type*] ATTRIBUTES(  
    AttributeDec*) ACTIONS(ActionDec*)  
2 AttributeDec  ::= Id:Type  
3 ActionDec     ::= Id  
4 Type, Id      ::= littoral  
5 Affectation   ::= var.id '=' val
```

Standard Types (Prototype) In the following, we present the predefined types of AAL written using the AAL type extension.

Listing 3: Standard library AAL types

```
// Language types  
TYPE List ACTIONS(size add remove)  
TYPE String  
TYPE Number  
  
// The main types (components)  
TYPE Actor ATTRIBUTES(id:Id) ACTIONS()  
TYPE Agent EXTENDS Actor ATTRIBUTES(id:Id)  
TYPE Data EXTENDS Actor ATTRIBUTES(subject:Id) ACTIONS(read write delete  
    send receive)  
TYPE Service  
  
// The extended types for accountability  
TYPE Subject EXTENDS Agent  
TYPE Processor EXTENDS Agent ACTIONS(audit)  
TYPE Controller EXTENDS Agent ACTIONS(audit)  
TYPE Auditor EXTENDS Agent  
  
...
```

1.1.3. Time extension

We did not investigate the semantics of this extension but we could rely on a translation into pure first-order temporal logic making the time flow explicit, as quoted in [Fis08].

Listing 4: AAL Time extension

```
1 ActionExp ::= ActionExp | Action Time  
2 Author    ::= (PERMIT | DENY) Action Time  
3 Action    ::= agent.service '[' agent ']' '(' Exp ')' [Time] [  
    Purpose]  
4 Time      ::= (AFTER | BEFORE) date  
5           | Time (AND | OR) Time  
6 date      ::= hh ':' mm ':' ss DD '/' MM '/' YYYY
```

1.1.4. AAL clause comparison

We can compare two AAL clauses but we make the assumption that the two audit parts remain the same. In other words they have the same characteristics (frequency, auditor, auditee, etc). The first reason is that we do not have, yet, a complete temporal logic interpretation of the audit action. The second reason is that it seems practically sufficient to work under this hypothesis. In order to check if a data subject preferences match with cloud service provider policy, we need to compare clauses in different AAL policies. This comparison is possible, using the formal semantic of AAL. Here we explain the principle in an abstract way. Let us consider two clauses : $U(uc_1, aa_1, rc_1)$ for a data subject preference, and $O(uc_2, aa_2, rc_2)$ for the service provider. The clause O satisfies the data subject preference U if : $(uc_2 \implies uc_1) \wedge (aa_1 = aa_2) \wedge rc_2 \implies rc_1$. This means that the usage(respectively rectification) part of the clause O has to be *stricter* than the usage (respectively rectification) part of the clause U , and the audit parts of the two clauses have to be equivalent.

In this example the data subject preference is not satisfied by the service provider policy :

```
Clause U :  
FORALL d:Data WHERE d.subject = Kim  
PERMIT hospital.collect[Kim](d) AND PERMIT hospital.send[thirdParty](d)  
    PURPOSE (research)  
AUDITING DPA.audit[hospital]()  
IF_VIOLATED_THEN DENY hospital.send[thirdParty](d)
```

Kim allows the hospital to send his data to third party for research purpose only.

```
Clause O :  
FORALL s:Subject  
FORALL d:Data WHERE d.subject = s  
PERMIT hospital.collect[s](d) AND PERMIT hospital.send[thirdParty](d)  
    PURPOSE (research statistics)  
AUDITING DPA.audit[hospital]()  
IF_VIOLATED_THEN DENY hospital.send[thirdParty](d) PURPOSE (statistics)
```

The hospital is allowed to send subject's data to third party for research and statistics purposes. Comparing two different auditing steps is an open question, we envision to study in future work.

1.2. Refinement of A-PPL

The formalization of efficient accountability practices into accountability policies is a fundamental aspect to safeguard provider-customer relationships. In particular, a concrete policy language enables the representation of human-readable policies into computer-readable and understandable format. In the first deliverable on the policy representation framework [ABC⁺13], we presented a first version of Accountable-PPL (A-PPL). This language extends PPL with features that make the representation of accountability policies possible. A-PPL is designed in such a way that the enforcement of these policies are achievable. In this section we first summarize the outcome of our initial work on A-PPL, and then we present the refinement of this language.

1.2.1. Initial specification of A-PPL

A low-level accountability policy language such as A-PPL supports some of the phases involved in the management of accountability policies, in particular the phase of enforcing the policies. One of the advantages of A-PPL is to be fairly simple since it has been designed upon three existing languages, among which two are standard languages. A-PPL extends the Primelife Policy Language (PPL) [TNR11], which is built upon the eXtensible Access Control Markup Language (XACML)[OAS] which is also based on the eXtensible Markup Language (XML) [BPSM⁺97]. The expressiveness of accountability policies in a machine-readable format is one motivation for the design of A-PPL. Facilitating the process of enforcement of these policies is another motivation. That is the reason why A-PPL provides enforceable accountability extensions to PPL.

A-PPL enables the specification of the following items:

Access Control Rules. As A-PPL extends XACML, it allows the definition of access control rules.

Data Handling and Accountability Obligations. Defined as a combination of Triggers (events filtered by a condition) and Actions, obligations in A-PPL state the specific actions to be performed by the data controller or processor after the occurrence on an event. In particular, A-PPL provides Triggers and Actions for specific accountability obligations such as notification, logging and audits.

Authorizations. An A-PPL policy can specify authorizations regarding the purpose of usage of collected data. It also can define authorizations for sharing collected data to third parties (referred as authorizations for downstream usage).

In [ABC⁺13], we proposed several extensions that are summarized below. They aim at capturing the identified accountability obligations in such a way that the enforcement of the policies that may include those extensions is feasible and facilitated.

Roles We defined an attribute to explicit the roles of different entities involved in the policy rules. This is needed in the case where attribution of responsibilities is required. We also proposed to define a role for Auditor.

Triggers We devised several Triggers for accountability-related events. We identified two Triggers that are in relation with evidence collection processes: `TriggerOnEvidenceRequestReceived` and `TriggerOnEvidenceReceived`. We also proposed two new Triggers that fire actions in case of policy or preference updates: `TriggerOnPolicyUpdate` and `TriggerOnPreferencesUpdate`. Finally, we created the Trigger `TriggerOnComplaint` that allows to govern actions on receipt of a complaint. Table 2 in Appendix A shows the list of available triggers in the current version of A-PPL.

Actions The contributions of A-PPL with respect to accountability actions lie on two main actions: notification and logging. The former, ruled by the new element `ActionNotify`, enables the sending of notifications to any recipient specified in this element. It also permits to specify the type of the notification. The latter, provided by the element `ActionLog`,

enables to log the event that triggers this action. This element introduces several parameters to make explicit what information is required to be logged, including timestamps, actions, purposes, resource id and resource location.

Evidence In [ABC⁺13], we proposed two new actions that are related with evidence: `ActionAudit` that initiates an audit protocol to verify compliance with policies and `ActionEvidenceCollection` that controls the collection of the requested evidence by the audit. We also envisioned to extend the XACML Request-Response format to represent an Evidence request.

Duration for Purposes We recommended to add an attribute in relation with the purposes of collection and usage of data. PPL defines a `Purpose` element. A-PPL enables the expression of a duration to that purpose element `<Purpose duration=2Y>`. This is particularly important in the case where a piece of data has been collected for different purposes that imply different retention periods.

1.2.2. Updates in the language

The initial steps on the implementation of the A-PPL engine (A-PPLE) and the refinement of the obligations coming from the B-3 work package led us to a reconsideration of our first proposal of A-PPL.

We propose a new trigger denoted `TriggerOnDataCollection`. This trigger suggests that the entity which collects personal data from data subjects who already gave their consent may be subject to obligations on the time of the collection. Essentially, the related obligations consist in informing the data subjects about the processing and the purpose of processing their personal data.

We introduce two new triggers that relate to access control. We propose `TriggerPersonalDataAccessPermitted` and `TriggerPersonalDataAccessDenied`. They fire actions based on the result of an access decision taken on a piece of data. In other words, if the evaluation of the access control on the targeted data is *Permit*, `TriggerPersonalDataAccessPermitted` may trigger an action specified in the policy. Symmetrically, `TriggerPersonalDataAccessDenied` triggers actions when an access on a piece of data is denied. Listing 5 shows an example of usage of these triggers.

Listing 5: Notification of the data subject of access permitted to his personal data

```
<!-- Notify the data subject when access to his personal data is
      permitted -->
<Obligation>
  <TriggerPersonalDataAccessPermitted>
    <MaxDelay>
      <!-- Notify within 2 minutes -->
      <Duration>0Y0M0DT0H2M0S</Duration>
    </MaxDelay>
  </TriggerPersonalDataAccessPermitted>
  <ActionNotify>
    <Media>e-mail</Media>
    <Address>data.subject@example.com</Address>
    <Recipient>Data Subject</Recipient>
```

```

    <Type>Access granted to personal data</Type>
  </ActionNotify>
</Obligation>

```

As a consequence to these two new triggers, we modify the existing `TriggerPersonalDataAccessedForPurpose` and call it `TriggerPersonalDataAccessed`. This trigger occurs when the data is actually released to the requester.

We proposed in [ABC⁺13] the element `ActionAudit` that enables the specification of audits in the policy language. At the time of writing this document, we do not envision to enforce a A-PPL policy that involves `ActionAudit` via the A-PPL engine. Instead, when triggered, `ActionAudit` requires to send a message to the Audit Agent System tool, developed in the A4Cloud project and responsible for performing audits.

We update the way A-PPL enables the specification of data location rules. We define a new attribute `region` to express the location of collected data. This attribute should be used as an attribute of the A-PPL `<Purpose>` element that is nested inside a `<AuthzUseForPurpose>` environment.

User awareness is one aspect of accountability. In particular data subjects should be informed about the processing of their personal data. They also have the right to give their consent about the processing. For that reason, we suggest to A-PPL with a new action related to this issue. We denote this new action `ActionRequestConsent`.

Figure 1 shows the structure of an A-PPL policy, that derives from the XACML structure and highlights the new extensions provided by A-PPL.

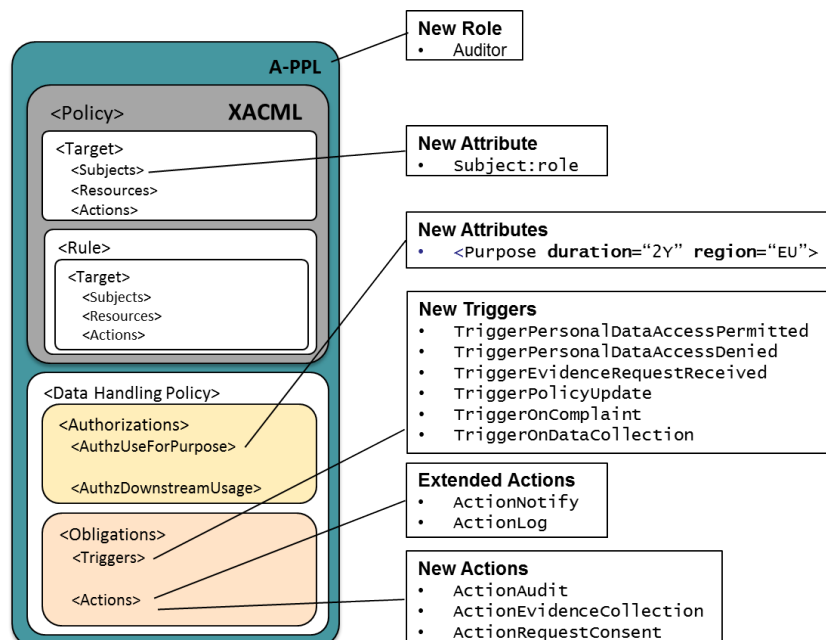


Figure 1: Structure of an A-PPL policy

Note that A-PPL is extensible by essence. Therefore, the list of triggers and actions can

be extended in the future in order to address further obligations. For example, policies may define the obligation to encrypt collected data to preserve its confidentiality. An A-PPL action such as `ActionEncrypt` may be created to be used in combination with the trigger `TriggerOnDataCollection` to specify such an obligation. This potential `ActionEncrypt` may include dedicated parameters such as the encryption algorithm, the key size, etc.

1.2.3. Related Work

Recent related work support our choice to extend PPL with accountability features. However, unlike ours, these works do not propose a complete accountability policy language such as A-PPL, neither an extension of the PPL engine. They rather propose some extensions to the PPL language to address the requirements that the authors of [BCLM13, HGKW13] have identified.

In parallel to our work, Butin et al. [BCLM13] leverage PPL to design logs for accountability. They develop two case studies (a private bank account and a hospital handling personal data) and they show the limitations of PPL with respect to logs. In particular, similarly to our PPL analysis, they identify the lack of expressiveness of PPL logging action. Indeed, the `ActionLog` element does not provide sufficient information in the logs. Besides, they discuss the fact that the PPL element `ActionNotifyDataSubject` does not allow to specify the content of the notification. Our accountability language A-PPL proposes a solution for these two above problems by extending `ActionLog` and `ActionNotifyDataSubject` for better expressiveness. As a matter of fact, we extend the notification element to recipients other than the data subject.

Similarly, Henze et al. [HGKW13] focus on the goal to make cloud computing aware of data handling requirements. They identify location and duration of storage as the two main challenges in cloud data handling scenarios. They propose to create *data annotations* that contain the data handling obligations (e.g. “delete after 30 days”). These annotations are transmitted to the cloud service before the annotated data in order to match the corresponding obligation against the cloud service’s data handling policies. If the annotations match the data handling policies, the cloud service signs the annotation and sends it back to the user, who now has a proof that the cloud will process the data as stated in the annotation. The authors in [HGKW13] suggest to leverage the PPL language to formalize the data annotation. Without giving more details, they propose to address the obligation of duration of storage by introducing maximum and minimum duration of storage attributes. They also define an extension to PPL with an element that restricts the location of stored data. A-PPL also addresses these two challenges and we give in the A4cloud project the details of the extensions that solve these issues. On the other hand, sending annotations can impact data privacy, since the annotations leak potentially privacy-sensitive information to the cloud service.

2. From Abstract Clauses to Concrete Policies

In chapter 1, we introduced AAL and A-PPL, the two languages we provide to express accountability obligations in A4Cloud. The power of AAL resides in its simplicity which makes it usable by non-experts in policy languages to express accountability obligations in an abstract fashion. On the other hand, A-PPL allows the expression of obligations with concrete policies and hence is enforceable using an adequate policy engine and enforcement methodology (see chapter 4).

Even if the mapping of accountability obligations into abstract clauses in AAL can easily be performed, transforming them into concrete accountability policies might not be an obvious task. In addition, inconsistencies in AAL clauses might prevent their correct transformation into A-PPL policies and hence their enforcement. In this chapter, we first present our approach for checking the consistency and correctness of an accountable system whose accountability obligations are represented in AAL in Section 2.1. We further define a set of rules to transform abstract AAL clauses into concrete A-PPL policies in Section 2.2.

2.1. Accountable Component Design

The goal of this section is to discuss and present the notion of an accountable component design. This notion is a particular case of what Butin et al. called “accountability by design” in [BCM14]. Indeed, accountability by design takes into account more precise and operational concerns, for instance secure logs, while an accountable design is only the abstract view of a design enabling accountability. An accountable component design helps in designing a correct system in several ways by putting constraints on the system specification. An accountable component design:

1. defines necessary conditions to enforce a system, (for instance, a logical inconsistency makes impossible a faithful implementation of a specification),
2. adds the guarantee that data subject preferences are satisfied by the components in the system in the abstract (logical) level,
3. early checks that a component behavior is compliant with the clauses it declares, and
4. provides a compositional approach allowing for an easy and safe replacement of accountable components.

2.1.1. Introduction

There are several variations which are important to consider before designing and implementing accountability in a system. We quote only some of them relevant to our purpose here:

- one data communication paths or several,
- one piece of data or data can be split in several pieces,
- explicit proof or static assurance by design construction, and

- policy evolution and modifications.

We first assume that there is no data splitting, only one data communication path and no policy modification; furthermore, our purpose is first to discuss and present a way to define static guarantee regarding accountability. Then, we will relax some of our hypotheses and take into account some extensions. Our presentation here will be informal and more details can be found in [BGR⁺14, BGRS14]. We illustrate the introduced concepts with a simple example.

We consider that our system of interest is provided with a design in terms of interacting components. These components define provided and required services allowing data communication. We name "an accountable component design" as a design where each service is annotated with clauses. We of course use the AAL language to annotate the services. A data is also annotated with its data subject preferences acting as a kind of implicit sticky clause.

We expect to define some properties for an accountable component design: for instance a clause should logically be consistent, a clause is not contradicting data subject preferences, etc. These properties either are necessary conditions for the design to be enforced (that is, accountability can be implemented) or provide some guarantees on the component behavior regarding the data subject preferences and the provider clauses, or allow a compositional replacement.

2.1.2. Illustrating Example

We consider a simple example coming from the health care use case, described in [BFO⁺13]. We consider a refined design with some of the involved agents. We do not allow data splitting.

- Kim can get access to his data and modify his data subject preferences.
- cloudX which is a subcontractor of cloudZ gets the data from Kim and stores it using cloudY subcontractor.
- The hospital should allow Kim to access his data and allow him to update his preferences. Furthermore we will assume that the hospital is able to process Kim's data for research purposes (research internal action).
- cloudZ allows Sandra to upload some data shared with Kim and it also enables data transfer from cloudX to hospital.
- Sandra will upload some information related to Kim's activity.

Regarding the definition of the type Data, our purpose is to capture the daily activity of Kim, that is Kim's identity, date and time, type of activity, and duration. But there is also a concern about the patient position. Indeed, collected data from Kim are sensitive and data uploaded by Sandra are sensitive too. This information can also be represented in the abstract design.

In Figure 2 there are labels which denote data subject preferences and provider clauses that are described below.

- U1 represents Kim's initial preferences.

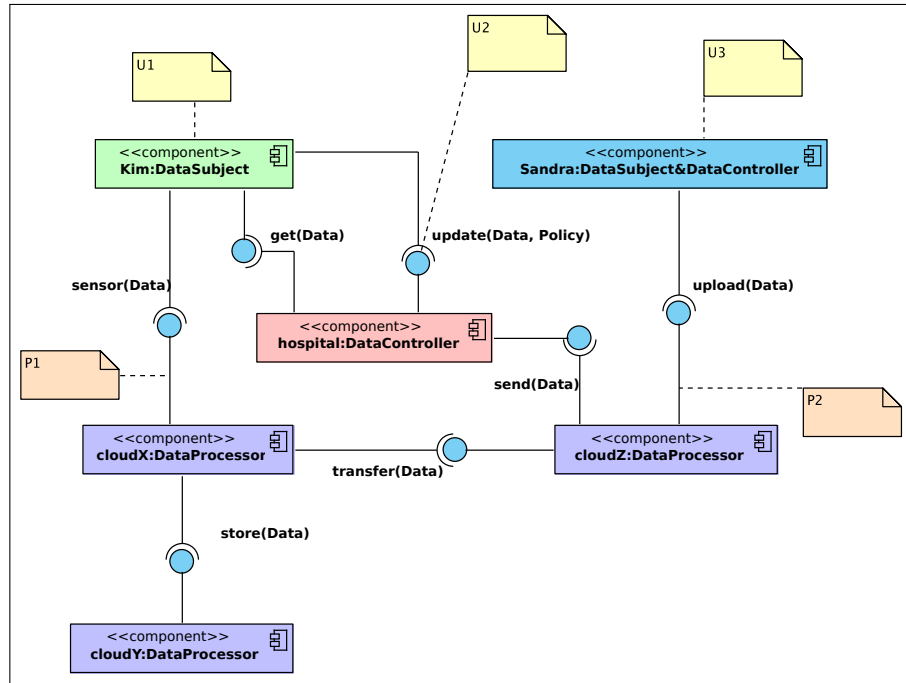


Figure 2: A Design Example

- U2 represents Kim's updated preferences.
- U3 represents Sandra's preferences both regarding her own data and the data in common with Kim.

A possible AAL formulation of these clauses is provided below. For U1 we could enumerate the permitted actions as follows.

```

EXISTS d:Data, p:Policy IF (d.subject=Kim) THEN
(PERMIT hospital.research(d) AND PERMIT hospital.get[Kim](d)
 AND PERMIT Kim.update[hospital](d,p) AND PERMIT cloudX.transfer[cloudZ](
 d)
 AND PERMIT cloudX.store[cloudY](d) AND PERMIT Sandra.upload[cloudZ](d)
 AND PERMIT cloudZ.send[hospital](d))
AUDITING auditor.audit[hospital]() everyday
IF_VIOLATED_THEN auditor.sanction[hospital](...)

```

As an example we provide here an audit and a rectification clause. For the sake of clarity, we only focus on usage control expressions.

The usage expression of U2 could be described as follows: restricting the data type and removing the right to do research. This restriction is simply done by substituting the original Data type with its DataWithLocation subtype.

```

// U2
FORALL d:DataActivity, EXISTS p:Policy IF (d.subject=Kim) THEN

```

```
(PERMIT hospital.get[Kim](d) AND PERMIT cloudZ.send[hospital](d)
AND PERMIT Kim.update[hospital](d,p) AND PERMIT cloudX.transfer[cloudZ](
d)
AND PERMIT cloudX.store[cloudY](d) AND PERMIT Sandra.upload[cloudZ](d)
AND DENY hospital.research(d))
```

We propose, in figure 3, a data structuring which makes the location concern explicit. This information could be useful in writing AAL clauses, for instance for naming attributes or for identifying a specific subtype.

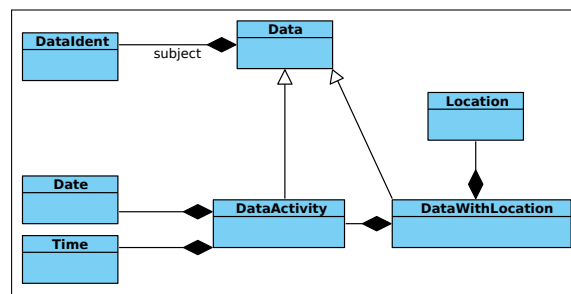


Figure 3: Data Structuring

U3 usage control expressions could be written as U2 AND <MORE>, where MORE corresponds to the data subject preferences for the data of Sandra.

Provided services can define clauses as illustrated below:

```
// P1 is the clause attached to the provided sensor service.
FORALL d:DataActivity IF (d.subject=Kim) THEN
(PERMIT cloudX.store[cloudY](d) AND PERMIT cloudX.transfer[cloudZ](d))

// P2: is the clause attached to the provided upload service.
FORALL d:DataActivity IF (d.subject=Kim) THEN PERMIT cloudZ.send[
hospital](d)
```

2.1.3. Well-formedness

In [BGRS14] we define the preliminary notion of well-formedness. This notion covers three different aspects:

1. Inconsistency in clauses.
2. Clause implying no action.
3. Static insurance of the data subject preferences and providers policies.

While the two first aspects are interesting, these are quite obvious properties (but may be not trivial to prove). We will focus on the third which is the most interesting one from an accountability point of view. The rationale behind this property is to ensure that the data subject preferences

are statically satisfied. One way to check this is to do a global proof on the design, but this is often complex and tedious. We rather propose a more modular way using the notion of well-formedness. The property expresses that a provider clause should imply the data subject preferences of the data carried on the service. For instance, considering the usage expressions of U1 and P1 we should have $P1.uc \Rightarrow U1.uc$.

However, the data subject preferences can also control the data communication in the agent network; thus, the provider clause is stricter than the data subject preferences if the communication is triggered. Hence, the real property should informally be stated as: the data communication and the provider usage control expressions ensure the user's usage control clauses, furthermore the data communication and the provider's rectification clauses ensure the user's rectification rules. As an example, consider that the provided clause of transfer is P2; we should have: $(\text{MUST Sandra.upload[ccloudZ]}(d) \text{ AND } P2.uc) \Rightarrow U1.uc$.

A data can follow a communication path and travel across several communication edges and agents. Thus, the well-formedness property should be checked on each agent of the path. However, a data can reach an agent with several communication paths. In this case we should consider a union of all possible paths. To check well-formedness on a network we can consider a covering of the edges with paths of maximal length but without cycle. The well-formedness property ensures that the data subject preferences are satisfied in any place the data is authorized to travel.

Note that we use the following principle: on a communication path the clauses should be in (non strict) increasing ordering for the \Rightarrow relation. Otherwise one clause could invalidate the data subject preferences of the data circulating on this path.

2.1.4. Well-connection

The previous property is interesting but one weakness is that it refers to explicit data subject preferences. A better way is to abstract from data subject preferences, that is to define a stronger property. A data subject can input his data in the network and if his preferences are satisfied by the input service clause then well-formedness is ensured. This new property, called *well-connection*, enables a compositional approach, that enables the definition of static rules for the replacement of accountable components.

We enrich the accountable component design with required clauses by attaching AAL clauses to each required service. For each connection the required clause should ensure the provided clause; this is defined as *connection compliance*. We get a similar property to well-formedness but based on the connection compliance of the communication edges in the network. Moreover, we should have the following relation between clauses of provided services and clauses of required services of an agent: any required clause should imply any provided clause. The rationale for this is that a data can go across an agent (that is entering by a provided service and leaving it by a required service) and the principle of policy evolution should be respected.

Furthermore, we consider that we have components with an abstract specification of their behaviour. For instance, it is classical to use process algebras or state machines to specify component behaviors. We consider a simple process algebra denoting what kind of actions a component can process. For instance, the behaviour of the hospital could be: "update Kim's preference", or "get Kim's data from cloudZ and then resend data to Kim" or "do research on

it". This could be described using an informal process algebra notation:

```
hospitalProcess = Kim.update[hospital](u) +
                  cloudZ.send[hospital](d) ; (hospital.get[Kim](d)
                  +
                  hospital.research(d))
                  ; hospitalProcess
```

Thus, now we have some means to check that, for each component, its provider clauses are compliant with its behavior. More precisely, figure 4 helps us to understand the different relations. In this figure, we have a simple communication path from the data subject to a processor. The data subject defines some preferences, denoted as U . The first condition is: $P_input \Rightarrow U$ which means that when entering in the network the data follows the declared clause of the controller. A second condition is related to the communication edge, we should have strictness of policies (or stronger policies) along the communication path. Thus we have $P_get \Rightarrow R_get$ and this relation is conditioned by the `MUST Controller.get[Processor](d)` as in the reachability formula of [BGRS14]. The third relation is also due to strictness along the communication path which goes across the Controller agent and can be formulated as `ForAll out (R.out \Rightarrow ForAll in P.in)`.

The last relation glues the agent behavior with the clause attached to the provided service. This clause describes what the component providing the service declares to do and what is expected in the future. This clause can not only contain information related to the current controller usage control or rectification rules but also more global information. The clause restricted to the controller agent should be compatible with its behaviour; in other words the agent should act according to its provided clauses. The formula expresses that the clause restricted to the agent should be satisfied by its behaviour. As an example of the last property, assume that the provided clause of `send` is:

```
FORALL d:DataActivity IF (d.subject=Kim) THEN
  (PERMIT hospital.send[Kim](d) AND PERMIT hospital.research(d))
```

The projection to the agent `hospital` is trivial here and does not change the clause. We should check that it is compatible with the `hospital`'s behavior, that is, if the agent is processing an action this action should be authorized.

For each agent we should check the behavioral compliance with the provided clauses. Furthermore, as with well-formedness, we should check the clause strictness on all maximal communication paths.

2.1.5. Accountability Types

Let us note in^* and out^* the provided and required services of a component respectively. We can consider both these sets as types and they together define the accountability type of a component. This is a usual approach in component specification and was used not only for typing systems, but also with various behavioral notations (see [Ame89, LW94, Nie95, LP00, dAH01, SC00]). We can now state rules to have a safe component substitution regarding the accountability clauses. We get a kind of contra-variant rule; this is a similar situation as in

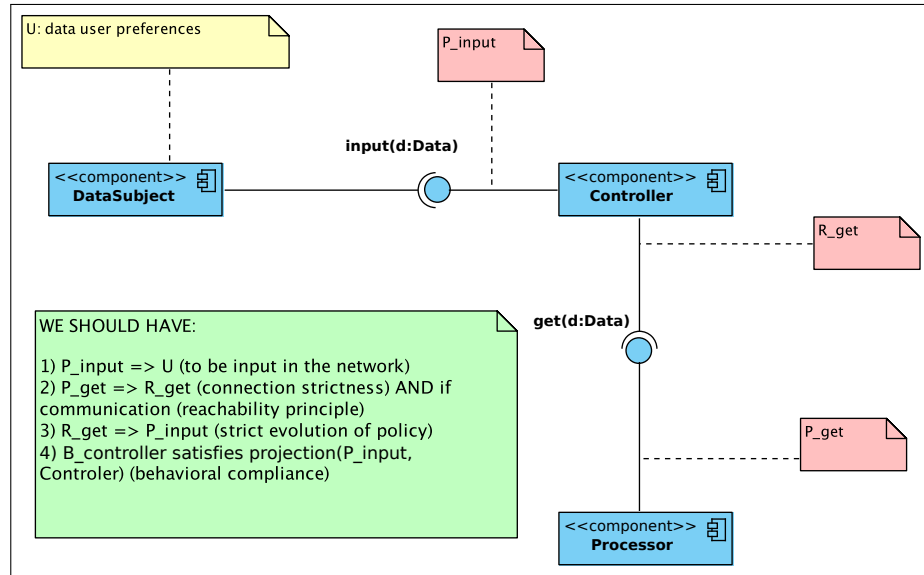


Figure 4: Typical Relations

classical component model [SC00]. To allow a safe substitution of a component, the type of the original component and the type of the new component should adhere to the following rules:

1. the provided clauses can be strengthened, and
2. the required clauses should be relaxed.

We informally justify the well-foundation of these rules in the sequel. The substitution of aComponent is graphically illustrated in Figure 5. The substitution context is captured by R_in^* and P_out^* and does not change in the two parts of the picture. Let consider data subject preferences U and only one provided and one required service for the sake of clarity. In the top part we should have, $P_out \Rightarrow R_out \Rightarrow P_in \Rightarrow R_in \Rightarrow U$ meaning that data subject preferences are satisfied. A similar relation should stand after the replacement of aComponent by the newComponent. Formally, the rules state that $P'_in \Rightarrow P_in$ and $R_out \Rightarrow R'_out$ for all required and provided services. Thus we have $P_out \Rightarrow R_out \Rightarrow R'_out \Rightarrow P'_in \Rightarrow P_in \Rightarrow R_in \Rightarrow U$, that is $P_out \Rightarrow R'_out \Rightarrow P'_in \Rightarrow R_in \Rightarrow U$, meaning that the newComponent satisfies the relation for this couple of services.

The substitution rule allows to have more provided services in the new component, the condition $P'_in \Rightarrow P_in$ stands only for the common services. For the required service the situation is dual: the new component should either have the same required services or less than the replaced component.

2.1.6. Possible Extensions

In this section, we discuss two extensions and present some open issues.

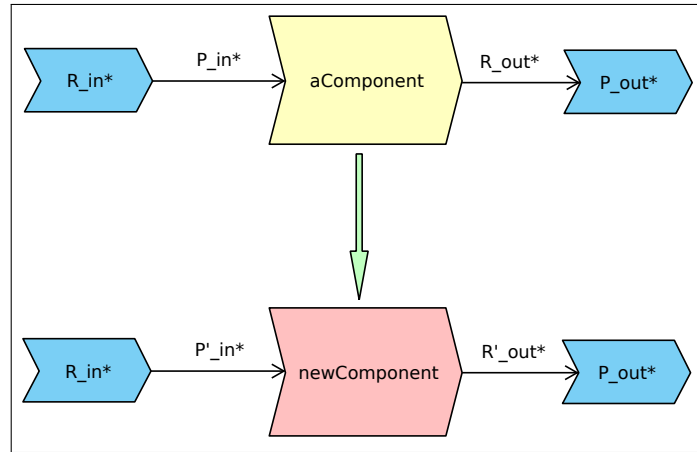


Figure 5: Accountable Component Substitution

Several Data We consider the case where we have a set of data cannot be split into several parts and rebuilt later on. In this case we consider each data with its corresponding data communication network and we apply the well-connected formula to each of them. This suggests that we should have an enumeration of data. Data subject preferences would then depend on the nature of the data (public, private, confidential, sensitive etc.). Thus, by providing the adequate subtypes for data and the associated preferences we can apply the well-connected formula to each data sub case.

Policy Modification Policy evolution denotes the allowed change for a data policy along a communication path. The usual rule is to have more strict policies along the communication path (see [PSSW09, CCD⁺05, BMB10]), and this is what we use in our model for well-formedness. Another, useful aspect is *policy modification*, that is a clause is a data and we can modify it during computation. This is what is done by the `update` service in our example. This allows a finer control for components in the system. For instance, a user can define weak preferences for his data, but the data controller can modify it to disable uncontrolled access from other components or completely remove the right to transfer the data. With this feature we need a notion of explicit sticky policy. Until now, since the data subject preferences do not change, we can consider that these preferences are implicitly attached to the data.

However, data subject preferences can now be modified (the action is named `update` here), and thus we should make the link between a data and its policy explicit. As with policy evolution, clauses and preferences should evolve monotonically; thus updating data subject preferences should be performed following a strict evolution rule. As an example, we should verify that $U2 \Rightarrow U1$.

Another meaning of policy modification is to allow the self-modification of a policy or a clause from the rectification clause. For instance, after a security breach the rectification clause automatically fixes the usage control to avoid the problem in the next runs. However, this feature raises two different problems: The first one is about the understanding and the readability of

the clause since we have a program which is self-modifying. This is a situation we want to avoid as it is forbidden in current programming languages. The second problem is that if we know in advance the kind of problem and the corresponding countermeasure, we are in the preventive/security case. ; in this case it is better to provide a more complex usage control rule preventing the known breach.

Open Issues One difficult case to address is when we allow splitting and merging data. There are various things we could expect to do in this domain:

- Once we have the behavior of the agents and the component diagram we can check that the model satisfies the data subject preferences. This is a global proof we can process using model-checking, in the same spirit of [BGRS14].
- Behavioral compatibility could also be checked in a similar way but we must precise the principle of projection and the adequate compatibility relation. Model-checking will still be the right approach to do that.
- We can use some prover to check the validity of AAL clauses but the problem is to find an adequate prover. Prover for linear temporal logic exists, for instance the Logics WorkBench¹, but effective provers for temporal logic with data and quantifiers are not common. This kind of tool is needed in order to check the well-connected property or to check for accountability subtyping.

Discussion and Limitations In this chapter we have provided the notion of accountable component design and the principle of well-connected accountable component design. Our idea is to capture well-known and intuitive concepts useful for a correct accountable component design. The benefits of the approach can be enumerates as follows.

1. It provides a complete and precise design for designers and for implementers.
2. The specifier can check the well-connected property with adequate tools like mCRL2 [CGK⁺13]. Some preliminary experiments have been performed [BGRS14] and we will extend it with the AccLab tool as part of our work in work package D3.
3. The well-connected property assures that the data subject preferences will be satisfied by any agent at any time.
4. It is a compositional approach, which facilitates proof, reuse and implementation.
5. It can provide some assistance and help in designing the system.

Writing clauses in a real setting seems challenging. All these rules can also assist in completing a partial design.

Another question is related to subtyping relationship for behavioral descriptions as described in Section 2.1.5. We did not investigate this issue because it does not seem so critical since we already have a subtyping relation on clauses.

¹<http://www.lwb.unibe.ch/>

2.2. Towards a rule-based system to transform AAL clauses into A-PPL policies

Despite the fact that AAL and A-PPL are respectively used to represent accountability clauses and policies, they are based on heterogeneous models. This is mainly due to the fact that we designed AAL to express accountability clauses, and thus the three parts (i.e. usage control, audit and rectification) of an accountability clause are explicit. On the other hand, we designed A-PPL as an extension of PPL which is also an extension of XACML dedicated to access control policies. Hence, this heterogeneity must be considered while transforming abstract AAL clauses into concrete A-PPL policies.

As seen in chapter 1, AAL has some specific features:

1. It allows permissions and prohibitions,
2. it allows actions (obligations) and complex conditions on data and agents,
3. it allows quantifiers, for data and agents,
4. it allows temporal expressions (temporal logic expressions)
5. it allows time expressions (duration, date, deadlines, ...)

First, there exists an important semantic difference between AAL and the target language (A-PPL). An AAL clause is assumed to be consistent², thus it is satisfiable, for instance we do not have a subexpression like `PERMIT a AND DENY b`. However, this is not the case with A-PPL.

In this section, we present the set of rules we define to transform abstract accountability clauses in AAL into concrete policies in A-PPL. Since the three parts of AAL clauses (i.e. usage, audit and rectification) are based on AAL expressions (i.e. `PERMIT | DENY User.read[processor](arg)`), we provide the rules associated to their mapping in Section 2.2.1 and rules for mapping special AAL constructs in Section 2.2.2.

2.2.1. AAL expressions

AAL expressions are of two types:

- Authorizations used to state actions (internal or exposed by another party) that an entity is allowed or not allowed (see Section 1.1 for details) to execute.
- Actions used to express an executable functionality performed by an entity in an AAL clause.

Transforming AAL expressions to A-PPL elements depends on their types, i.e. authorization or actions. The set of rules associated to both types are introduced in the sequel of this section.

²The consistency and correctness of AAL clauses are checked through the properties we defined in our accountable component design approach.

AAL authorizations We represent an AAL authorization expression exp as a tuple $\langle auth, agt, serv, res, args \rangle$ where $auth$ is the authorization (i.e. PERMIT or DENY), agt the agent performing the service, $serv$ the service, res the resource offering the service and $args$ eventual parameters passed to the service.

In A-PPL, such AAL authorization expression can be represented by an XACML `<rule>`. $exp.auth$ will determine the effect e of the rule (i.e. permit or deny), $exp.agt$ the access subject $subj$, $exp.serv$ the action act and $exp.res$ the resource res . In the `<rule>` element there is no direct support to represent eventual $exp.args$. At this stage, we only consider AAL expressions where $exp.args = \emptyset$.

Mapping an AAL authorization expression is done in different fashions based on its authorization:

- **rule1:** An AAL expression with a PERMIT modality corresponds to a `<rule>` element in an A-PPL policy with "permit" in the effect attribute.
- **rule2:** An AAL expression with a DENY modality corresponds to a `<rule>` element in an A-PPL policy with "deny" in the effect attribute.

In the following we provide a formal representation of these rules and their associated conditions. Rules are represented using the notation of Formula 1 which is read as follows: this *AAL representation* is mapped to this *A – PPL representation*, IF some *conditions* are verified. Here the *AAL* (resp. *A – PPL*) *representation* can be a complete or an excerpt (an AAL expression for instance) of an AAL clause (resp. A-PPL policy).

$$\begin{array}{l}
 \textit{AAL representation} \\
 \rightarrow \\
 \textit{A – PPL representation} \\
 \textit{IF} \\
 \textit{conditions}
 \end{array} \tag{1}$$

Considering an AAL exp represented by a tuple $\langle auth, agt, serv, res, args \rangle$, its mapping to an A-PPL rule $r = \langle e, subj, act, res \rangle$ is done using the following rules (Formula 2 and 3).

Rule 1: AAL expression with a PERMIT authorization

This transformation is conditioned by the facts that: (i) the *authorization* of the AAL expression is equal to PERMIT, (ii) the *agent*, *service* and *resource* elements are specified and (ii) the *arguments* element is null.

$$\begin{array}{l}
exp = \langle "PERMIT", agt, serv, res, \emptyset \rangle \\
\rightarrow \\
r = \langle "permit", exp.agt, exp.serv, exp.res \rangle \\
IF \\
exp.m = "PERMIT" \wedge exp.agt \neq \emptyset \wedge exp.serv \neq \emptyset \wedge exp.res \neq \emptyset \wedge exp.args = \emptyset
\end{array}
\tag{2}$$

Rule 2: AAL expression with a DENY authorization

This transformation is conditioned by the facts that: (i) the *modality* of the AAL expression is equal to DENY, (ii) the *agent*, *service* and *resource* elements are specified and (ii) the *arguments* element is null.

$$\begin{array}{l}
exp = \langle "DENY", agt, serv, res, \emptyset \rangle \\
\rightarrow \\
r = \langle "deny", exp.agt, exp.serv, exp.res \rangle \\
IF \\
exp.m = "DENY" \wedge exp.agt \neq \emptyset \wedge exp.serv \neq \emptyset \wedge exp.res \neq \emptyset \wedge exp.args = \emptyset
\end{array}
\tag{3}$$

We provide examples for the previous rules in Appendix B.

AAL actions We represent an AAL action expression *exp* using the same tuple representation we used for authorization expressions without specifying the authorization (i.e. *exp.auth* = \emptyset). In an AAL action *exp* = $\langle \emptyset, agt, serv, res, args \rangle$, *agt* represents the agent performing the service, *serv* the service, *res* the resource offering the service and *args* eventual parameters passed to the service.

As A-PPL extends XACML, it allows the definition of XACML *<obligations>*. Obligations are defined in the XACML 2.0 specification document [OAS] as:

XACML provides facilities to specify actions that MUST be performed in conjunction with policy evaluation in the <obligations> element ... There are no standard definitions for these actions in version 2.0 of XACML. Therefore, bilateral agreement between a PAP and the PEP that will enforce its policies is required for correct interpretation. PEPs that conform with v2.0 of XACML are required to deny access unless they understand and can discharge all of the <obligations> elements associated with the applicable policy. <obligations> elements are returned to the PEP for enforcement.

Hence, an AAL action expression can be mapped to A-PPL using the *<obligations>* element. In fact, and unlike authorization expression, we need to express an executable function-

ality that **must be performed** by an entity and this can be ensured using obligations. However, the following points have to be considered:

- Obligations in XACML are expressed in conjunction with a set of rules (i.e. authorization expression). The granted permission (i.e. permit or deny) will condition the fulfillment of the obligation. Thus, to express an AAL action in A-PPL, we represent it as an obligation that will be fulfilled on permit and we associate this obligation to a rule which is always permitted in a dedicated policy.
- Only a finite set of services (i.e. serv) can be considered. In fact, the PEP will be in charge of enforcing the action (i.e. the <obligations>). Here we have to assume that the entity offering the service (i.e. res) and the entity executing the service (i.e. agt) are known by the PEP or the the PEP itself and cannot be the data subject for instance.

The XACML <obligations> allows the definition of system obligations in a limited fashion [Pri11] and hence can be inadequate for realistic obligations (periodic obligations, conditioned obligations, etc.).

The obligation policy language offered by PPL [Pri11], which is the basis for A-PPL, offers more adequate means to express realistic obligations. PPL specifies obligations as Event-Condition-Action expressions. For simplicity, obligations in PPL are represented as:

Do **Action** when **Trigger** (from Start to End)

where trigger are events filtered by conditions.

PPL obligations are more adequate than XACML obligations for AAL actions representation. However, since they are initiated by triggers it's not possible to use them to represent simple AAL actions. Nevertheless, PPL obligations are adequate to represent AAL actions as part of a future construct (i.e. IF *exp* THEN MUST|MUSTNOT *act*).

2.2.2. Special AAL constructs

While using AAL expressions in the different parts of an AAL clause, i.e. usage, audit and rectification, several constructs can be applied to express, among others, temporal constraints, logical expression combinations and the purpose of an authorization. In the following we provide rules to translate such constructs.

Logic operators Combined AAL expressions can be formulated using the logic operators AND and OR. Here we assume that a combined AAL expression is purely conjunctive or disjunctive (i.e. AND and OR are never used together in a combined expression). We represent a combined AAL expression exp_c as a tuple $\langle E, op \rangle$ where E is a set of n atomic AAL expressions combined using the op operator. For example, a combined AAL expression $e1 \text{ AND } e2 \text{ AND } e3$ is represented as $\langle \{e1, e2, e3\}, AND \rangle$.

According to the used operator $exp_c.op$, the mapping into A-PPL is done as follows:

- **rule4:** a conjunctive set of n atomic AAL expressions corresponds to a <policy> in A-PPL defining $n + 1$ <rule>s:

- n <rule>s representation expressed as defined in Formula 2 and 3.
- an extra <rule> denying other actions apart the conjunction of the actions allowed in the previous <rule>s (i.e. with effect permit).
- **rule5:** a disjunctive set of n atomic AAL expressions corresponds to a <policy> in A-PPL defining $n + 1$ <rule>s:
 - n <rule>s representation expressed as defined in Formula 2 and 3.
 - an extra <rule> denying other actions apart the disjunction of the actions allowed in the previous <rule>s (i.e. with effect permit).

In both cases, the extra <rule> is expressed using the XACML <condition> element and all the <rule>s are sub-elements of the <policy> element. The so created A-PPL policy uses the deny overrides rule combining algorithm³.

In the following we define the two rules for mapping a combined AAL expression exp_c to an A-PPL policy p represented by $\langle comb, R \rangle$ where $comb$ is the used rule combining algorithm and R a set of rules r . We also extend our previously introduced representation of a rule r by adding a new element c representing a <condition> element: $r = \langle e, subj, act, res, c \rangle$.

Rule 4: a conjunctive set of AAL expressions (AND)

This transformation is conditioned by the facts that: (i) the provided set of expressions E contains at least two atomic expressions and (ii) the logic operator is equal to AND.

$$\begin{array}{l}
 exp_c = \langle E, "AND" \rangle \\
 \rightarrow \\
 p = \langle "deny - overrides", R^* \rangle \\
 IF \\
 |exp_c.E| \geq 2 \wedge exp_c.op = "AND"
 \end{array} \tag{4}$$

$$* R \text{ such as } \forall exp_i \in exp_c.E, R.add(rule_{1,2}(exp_i)) \wedge R.add(r_{AND})$$

In Formula 4, r_{AND} is the extra rule denying other actions apart the conjunction of the actions allowed in R . A representation of this rule in A-PPL is provided in Appendix B.3.

Rule 5: a disjunctive set of AAL expressions (OR)

This transformation is conditioned by the facts that: (i) the provided set of expressions E contains at least two atomic expressions and (ii) the logic operator is equal to OR.

³In XACML, rule-combining algorithms define a procedure for arriving at an authorization decision given the individual results of evaluation of a set of rules [OAS]. Deny-overrides is one of the standard combining algorithms defined in the XACML's specifications.

$$\begin{array}{l}
exp_c = \langle E, "OR" \rangle \\
\rightarrow \\
p = \langle "deny - overrides", R^* \rangle \\
IF \\
|exp_c.E| \geq 2 \wedge exp_c.op = "OR"
\end{array} \tag{5}$$

* R such as $\forall exp_i \in exp_c.E, R.add(rule_{1,2}(exp_i)) \wedge R.add(r_{OR})$

In Formula 5, r_{OR} is the extra rule denying other actions apart the disjunction of the actions allowed in R . A representation of this rule in A-PPL is provided in Appendix B.4.

Temporal modalities AAL supports the expression of basic temporal constraints such as actions to be ensured or allowed in the future or actions that have to be ensured in order to authorize some other actions. In the following we provide a high level overview on necessary rules to translate into A-PPL AAL expression using the following constructs:

- IF exp THEN MUST|MUSTNOT act
- exp ONLYWHEN exp

The general principles will be to translate these expressions using the notion of obligations, thus it is only possible with A-PPL rather than XACML. Since we have temporal expressions we should rely on program synthesis here, more precisely translating these expressions into a finite state machine controlling the behaviour. Note also that without explicit deadline it is not possible to check if a MUST action has been realized. Real obligations have generally dense time and temporal deadlines thus this checking remains possible.

Rule 6: future construct: IF exp THEN MUST|MUSTNOT act

We define this AAL future construct F as a tuple $\langle exp, auth, act \rangle$. It can be represented in A-PPL using the PPL obligation construct. We define an A-PPL obligation element o as a tuple $\langle action, trigger, start, end \rangle$ where $action$ is the identifier of the action (i.e. $F.act$), $trigger$ defining the event and conditions for which the obligation must be fulfilled (the adequate trigger will be selected based on $F.auth$ and $F.exp$) and $\langle start, end \rangle$ the validity period for the obligation (depends on the trigger). The list of triggers supported in A-PPL is available in Appendix A.

Rule 7: past construct: exp ONLYWHEN exp

This AAL construct can be represented in A-PPL using the PPL $\langle ProvisionalAction \rangle$ element. Provisional actions describe which actions have to be performed in order to grant access to a resource. In PPL, $\langle provisionalAction \rangle$ is part of the $\langle Rule \rangle$ element. Six provisional actions are defined: reveal attributes (4 variants), sign statement and spend (how many time you can use your credentials).

2.2.3. Discussion and Limitations

Since AAL clauses are made of AAL expressions, the rules we propose can be seen as the founding blocks for an automatic rule-based transformation of abstract accountability clauses into concrete A-PPL policies. However, the heterogeneities between the AAL and A-PPL models causes several transformation issues. For instance, AAL clauses might be expressed using quantifiers (i.e. `FORALL` | `EXISTS`) to express for instance that a user can be authorized to read all resources. Expressing such kind of quantifications is not supported by A-PPL. Also, AAL is based on a Linear Temporal Logic (LTL) allowing the expression of actions that will eventually hold in an unbounded future using the F operator for instance. Expressing such a rule is not supported in A-PPL and hence a clause expressing that a user will eventually read a resource in an unbounded future cannot be enforced.

Our rules allow an automatic transformation of simple AAL clauses into concrete A-PPL policies. However, we do not handle complex clauses involving for instance quantifiers and temporal operators.

3. Mapping Obligations in AAL and A-PPL

Obligations are an important aspect of service provisioning in accountable cloud ecosystems. In work package C-2, the term obligation is defined as

Obligation: An obligation is a requirement, agreement or promise for which an actor is morally or legally bound and that has certain consequences if it is breached. Obligations can be based on: contract, statute or morality [DFG⁺14].

Legal obligations are a subset of obligations and cover accountability obligations imposed by law and regulation. The scope of the A4Cloud project encompasses obligations related to personal data and business sensitive information. In this chapter we analyze obligations related to the processing of personal data, which arise from the EU Data Protection Directive (“the Directive”)⁴ and the proposed Data Protection Regulation currently undergoing the EU legislative process (“the proposed Regulation”)⁵.

Organisations that are subject to obligations need not only to meet their obligations, but they also need to make sure that their business partners and sub-contractors do not invalidate them. In particular, an accountable organization needs to make sure that the obligations to protect personal data are adhered to all along the service provisioning chain. An important aspect of governance in an accountable organisation is to define and deploy policies for their data processing practices and to make sure that they are followed by all the involved service providers. The policies should ideally travel with the data, and they should be used as input to monitoring of data processing practices, to generate evidences that policies are fulfilled, to correct policy violations that may occur and in general to demonstrate policy compliance.

Accountability obligations arising from the Directive (and also the proposed Regulation) are centered on the notion of control, i.e. the entity that is perceived to have control over the processing of personal data will (in most cases) be held responsible and accountable to the data subject for ensuring compliance by all providers in the service delivery chain. To derive accountability obligations from a data protection perspective, it is therefore necessary first to analyse the roles of the actors involved in terms of who are the data subjects, data controllers and data processors in relation to the processing of personal data.

In this chapter we demonstrate how legal obligations can be mapped into the A4Cloud policy languages that are defined in this deliverable. In Section 3.1 we provide examples of obligations and their corresponding policies, which have been derived for one of the A4Cloud business use cases (the use case itself is further documented in [BFO⁺13]). This section contains a summary of work that has previously been published in [BFO⁺13] and [BFO⁺14]. In Section 3.2 we then demonstrate how personal data transfer rules, which have been derived from the proposed Regulation, can be expressed in AAL and A-PPL. This section contains previously unpublished work.

⁴Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data.

⁵Proposal for a Regulation of the European Parliament and of the Council on the Protection of Individuals with Regard to the Processing of Personal Data and on the Free Movement of Such Data (General Data Protection Regulation)

3.1. Application to an A4Cloud Use Case

3.1.1. The Health Care Use Case

The health care use case concerns the flow of sensitive personal data generated by medical sensors in the cloud. The system, which is illustrated in Figure 6, is used to support diagnosis of patients by the collection and processing of data from wearable sensors. Here, we investigate the case where medical data from the sensors will be exchanged between patients, their families and friends, the hospital, as well as between the different cloud providers involved in the final service delivery.

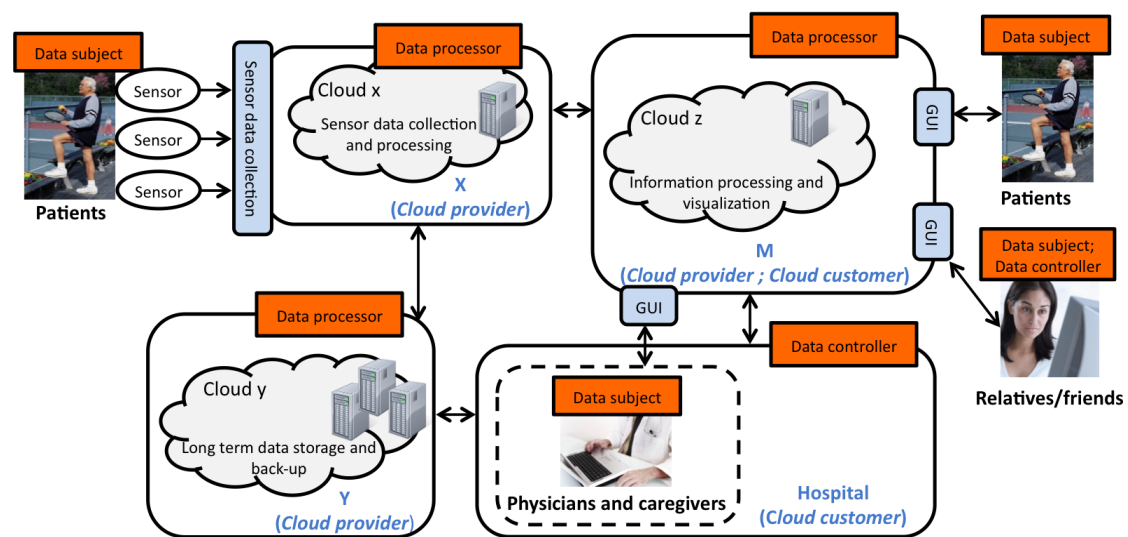


Figure 6: An overview over the main actors involved in the health care use case

In this use case the patients are the data subjects from whom (sensitive) personal data is collected. The hospital is ultimately responsible for the health care services and will hence be the data controller for the personal data that will be collected. The patients' relatives may also upload personal data about the patients and can therefore be seen as data controllers (as well as data subjects, when personal data about their usage of the system is collected from them). As can be seen in Figure 6, the use case will involve cloud services for sensor data collection and processing (denoted cloud provider "X"), cloud services for data storage (denoted cloud provider "Y") and cloud services for information sharing (denoted cloud provider "M"), which will be operated by a collaboration of different providers. Since the primary service provider M, with whom the users will interface, employs two sub-providers, a chain of service delivery will be created. In this particular case, the M platform provider will be the primary service provider and will act as a data processor with respect to the personal data collected from the patients. Also the sub-providers, X and Y, will act as data processors. The details of the use case are further described in DB3.1 [BFO⁺13].

3.1.2. Obligations for the Health Care Use Case

We summarize here the obligations that are detailed in DB3.1 [BFO⁺13]. These obligations are listed as either *legal*, *contractual* or *ethical* obligations.

- **Legal obligations:**

Obligation 1-4: Informing about processing, purposes, recipients and rights. The hospital is accountable to patients, relatives/friends and hospital staff for informing that their personal data is being collected, the purpose of the processing, the recipients of their personal data and the rights the data subjects have in relation to the processing of their personal data.

Obligation 5: Data collection purposes. The hospital is accountable to patients, relatives/friends and hospital staff for collecting data only for specific, explicit and legitimate purposes.

Obligation 6: The right to access, correct and delete personal data. The hospital is accountable to patients, relatives/friends and hospital staff for their right to access, correct and delete personal data.

Obligation 7: Data storage period. The hospital is accountable to patients, relatives/friends and hospital staff for keeping their personal data in a form which permits identification for no longer than necessary.

Obligation 8, 11-12: Security and privacy measures. The hospital is accountable to patients, relatives/friends and hospital staff for the security and privacy of the personal data they collect. The hospital is also accountable for choosing sub-providers that can provide sufficient safeguards to protect the personal data they process.

Obligation 9-10: Rules for data processing by providers. The hospital is accountable to patients, relatives/friends and hospital staff for how the sub-providers processors process their personal data.

Obligation 13-15: Consent to processing. The hospital is accountable to patients, relatives/friends and hospital staff for obtaining informed consent before collecting data, explicit consent (agree or disagree) before collecting sensitive personal data and before allowing joint data controllers to process them.

Obligation 16: Informing DPAs. The hospital is accountable to the data protection authorities to inform that they collect personal data.

- **Contractual obligations:**

Obligation 17: Informing about the use of sub-processors. The M platform is accountable to the hospital for informing about their usage of data processors X and Y to process personal

data.

Obligation 18: Security breach notification. X and Y are accountable to M for notifying about security breaches related to data subjects' personal data. Similarly, M is accountable to the hospital, which is in turn is accountable to the patients, relatives/friends and hospital staff for notifying security incidents.

Obligation 19-20: Evidence on data processing and data deletion. X and Y are accountable to M, which in turn is accountable to the hospital, for, upon request, providing evidence on their data processing practices and on the correct and timely deletion of personal data.

Obligation 21: Data location. Cloud X and Y are accountable to M and M is accountable to the hospital and the hospital to the data subjects for the location of the processing of personal data.

- **Ethical obligations:**

Obligation 22: Informing about personal data processing. This obligation means for instance that the hospital is accountable to cloud X and Y for informing them that they will use their services to process personal data.

Obligation 23: Personal data minimization. The hospital is accountable to the patients and the hospital staff for offering services that have been designed to minimize the amount of personal data it collects from them.

Obligation 24: Privacy-by-default. M is accountable to the hospital for offering services that have been designed in such a way that the strongest privacy settings are the default settings.

Obligation 25: Specifying user preferences. M is accountable to the hospital which is accountable to the patients for offering services that allow the users to specify privacy preferences.

Obligation 26: Monitoring of data practices. Cloud X and Cloud Y are accountable to M and M is accountable to the hospital to keep records of their respective monitoring of their data practices.

Obligation 27: Compliance with user preferences. Cloud X and Cloud X are accountable for providing evidences to M that personal data is processed in accordance to user preferences. The same accountability relationship exists between M and the hospital and between the hospital and the patients.

Obligation 28: Compliance with privacy policies. M should be demonstrate to the patients, their relatives and the hospital staff compliance with their policies in a timely fashion *reactively* and where possible *proactively*.

Obligation 29-30: Informing about policy violations and privacy preferences violations. Cloud X and Cloud Y should inform M about any policy violation that are related to personal data and any violation of users' privacy preferences. The same relationship exists between M and the hospital and between the hospital and the patients.

Obligation 31: Remediation in case of incidents. Cloud X and Cloud Y should provide remediation to M and its users in the case of damages caused to data subjects due to processing of personal data. M and the hospital have the same obligation.

The remainder of this section proposes attempts of translation of these obligations into AAL and A-PPL policies.

3.1.3. Obligations in AAL

We reuse the same method as in the deliverable D34.1.

- First, we assume an abstract design of the system described with components and services.
- Second, original obligations, as described in the previous section, are sometimes rewritten to map the design. The objective is to try to capture an operational intent.

Obligation 1-5: informing about processing, purposes, recipients and rights. This kind of obligation is too imprecise, but the mere principle is to send information to some agents under some conditions. For instance, the following AAL expression means that an agent will be informed about the processing of its data by the hospital.

```
FORALL d:Data, any:Agent ALWAYS
  IF (hospital.PROCESS(d:Data) AND (d.subject=any.ident))
    THEN MUST hospital.inform[any]("processing purpose")
```

Obligation 6: The right to access, correct and delete personal data. AAL is able to express full accountability clauses. However, the current example does not describe the audit and the rectification steps. We provide a simple possible example of a complete accountability clause. Provided that the controller provides the required services, the AAL clause looks like the following:

```
FORALL d:Data ALWAYS
  (PERMIT Kim.READ(d:Data) AND PERMIT Kim.WRITE(d:Data) AND PERMIT Kim.
    DELETE(d:Data))
  AUDITING Leslie.AUDIT(hospital.logs)
  IF_VIOLATED_THEN MUST leslie.SANCTION(hospital)
```

Obligation 7: Data storage period. With the introduction of time notion in AAL, we can express such obligation as the following :

```
FORALL d:Data
MUST hospital.delete(d) BEFORE d.expireDate
```

Obligation 8, 11-12: Security and privacy measures. At this , "abstract and fuzzy" level, we have no operational means thus AAL cannot be used. However, as soon as, precise information about the security measures, then AAL is relevant. For instance, if a data is required to be encrypted we can express it using the adequate action.

Obligation 13-15: Consent to processing This kind of obligation expresses a permission after a more or less complex consent protocol. We give below an example.

```
FORALL d:Data ALWAYS
(MUST cloudX.sensors(d:Data)) ONLYWHEN d.subject.informedConsent[
    hospital]()
AND (MUST d.subject.informedConsent[hospital]())
    ONLYWHEN hospital.getPolicy[d.subject]("processing_policy_and_
    purpose")
```

Obligation 16-18: Informing DPAs, about the use of processors, security breach notification This kind of obligations expresses information sent to one of several agents, this is quite obvious in AAL. For instance, part of Kim data preferences could be: "if there is a data breach I want to be notified of that":

```
FORALL d:Data ALWAYS IF (d.subject=Kim) THEN
(MUST hospital.notify[Kim](d, 'data breach')) ONLYWHEN (hospital.
    detectBreach(d))
```

Obligation 19-20: Evidence on data processing and data deletion. In this case the problem is to provide the correct evidences. Then it is not an issue in AAL to express sending this information.

Obligation 22: Informing about personal data processing. As many of the previous obligations it is sufficient to send notifications to some agents only.

3.1.4. Obligations in A-PPL

In this section, we review each of the obligations proposed by WP:B-3 and, if applicable, we define an A-PPL rule or policy mapping to these obligations.

We address first the legal obligations outlined in [BFO⁺14].

Obligation 1-4: Informing about processing, purposes, recipients and rights. To express this obligation with A-PPL language, the hospital prepares an A-PPL policy that states the

proposed data handling rules. Listing 11 from Appendix C is an example of a data handling policy proposed by the hospital about the processing of the collected personal data (lines 37-85).

When the data is about to be collected, the trigger `TriggerOnDataCollection` activates the action `ActionNotify` that is used to notify the data subject about the stated policy, the recipients of the data targeted by this policy and data subject's rights. Listing 12 in Appendix C shows an example of how this obligation can be translated into A-PPL.

Obligation 5: Data collection purposes. Listing 11 in Appendix C shows how the hospital can express a list of purposes (admin, research and survey purposes) in the `AuthzUseForPurpose` environment. This list of purposes is specified in lines 74 to 78.

Obligation 6: The right to access, correct and delete personal data. Lines 2 to 36 in Listing 11 from Appendix C present a read-write-delete access control rule that authorizes the data subjects to read, write and delete their personal data.

Obligation 7: Data storage period. A-PPL provides the action `ActionDeletePersonalData` that can be triggered when the storage period is over. Lines 63 to 71 in Listing 11 from Appendix C show an example a rule that specifies that the personal data must be deleted after two years.

Besides, we define the attribute `duration` that indicates the data storage period for data collected for particular purposes. These periods are shown in lines 74-78 in Listing 11 in Appendix C.

Obligation 8, 11-12: Security and privacy measures. As such, this obligation is too vague to be translatable in A-PPL. However, in order to protect the confidentiality of data, A-PPL allows to express access control rules using XACML expressions such the ones presented in Listing 11 of Appendix C. Other data handling and data protection rules mostly inherit from PPL, its ancestor, and the new elements defined within this new language.

Obligation 9-10: Rules for data processing by providers. This obligation can be expressed using the `AuthzDownstreamUsage` element which specifies under which conditions (e.g. the data processing rules to enforce) the personal data can be transmitted to sub-providers. These conditions are expressed via an A-PPL policy that is sent to these sub-providers and which is nested in an `AuthzDownstreamUsage` environment as shown in Listing 11 in Appendix C (lines 79-83).

Obligation 13-15: Consent to processing. In this case, we propose to use the new action `ActionRequestConsent` that will ask the data subject to give its consent in order to permit or deny the collection of data. An example of use of `ActionRequestConsent` can be found in Listing 13 of Appendix C.

Obligation 16: Informing DPAs. This obligation is simply translated using the trigger `TriggerOnDataCollection` and the action `ActionNotify` by specifying the DPA as the notification recipient. This can be expressed as in Listing 12 of Appendix C. WP:B-3 also proposes the

following contractual obligations in DB3.2 [BFO⁺14]. We present here how these can be translated into A-PPL statements.

Obligation 17: Informing about the use of sub-processors. An A-PPL data handling policy on the M platform can express this obligation by employing the trigger `TriggerPersonalDataSent` in conjunction with the action `ActionNotify` which will notify the hospital about the usage of personal data by X and Y. Listing 14 in Appendix C shows how this can be expressed in A-PPL.

Obligation 18: Security breach notification. The A-PPL statements use the `ActionNotify` element to send such notifications to the corresponding recipients. Listing 16 in Appendix C presents an example of security breach (data loss) notified to the data subject.

Obligation 19-20: Evidence on data processing and data deletion. The corresponding A-PPL statement makes use of the trigger `TriggerOnEvidenceRequestReceived` which will fire the action `ActionEvidenceCollection`. An example of an A-PPL rule for evidence of data deletion is described in Listing 17 in Appendix C.

Obligation 21: Data location. We propose in A-PPL a standard attribute `region` to specify the location in the `<Purpose>` elements that are placed inside a `<AuthzUseForPurpose>` environment. Thus we define the authorized locations of processing of the collected data for particular purposes. An example is presented in Listing 15 of Appendix C.

Finally, we present in the last paragraphs the translation (if applicable) of obligations coming from the ethical perspective into A-PPL statements.

Obligation 22: Informing about personal data processing. This obligation means for instance that the hospital is accountable to Cloud X and Y for informing them that they will use their services to process personal data. An A-PPL policy in the hospital side can trigger the action `ActionNotify` to notify X and Y that they are processing personal data.

Obligation 23: Personal data minimization and Obligation 24: Privacy-by-default. Since these obligations are related to the design of the cloud service, rather than to its actual operation, we consider them to be out of scope for policies.

Obligation 25: Specifying user preferences. A-PPL allows the data subjects to express their preferences. Unlike PPL, A-PPL does not perform any matching between user preferences and data controller's policies. This matching is performed at the AAL level. The output of the matching is part of the A-PPL policy.

Obligation 26: Monitoring of data practices. To keep such records, they can specify in an A-PPL policies the `ActionLog` to log the events and actions related to these practices. An example of use of such logging action is provided in Listing 11 in Appendix C (lines 52-61).

Obligation 27-28: Compliance with user preferences and privacy policies. This obligation

can be addressed via an evidence collection process. This can be ruled by the `ActionEvidenceCollection` element, like the example provided in Listing 17 in Appendix C.

Obligation 29-30: Informing about policy violations and privacy preferences violations.

Using the `ActionNotify` action element, these obligations can be easily expressed in an A-PPL policy. Listing 18 in Appendix C presents an example of policy violation notification.

Obligation 31: Remediation in case of incidents. During the current phase of the project we envision to send notifications for remediation in case of incidents. Using the A-PPL `ActionNotify` we can specify the object of the remediation (financial compensation, policy update, etc.). As a future work, we can think of specifying a new element `ActionRemediate` that will ease the definition of the action to be undertaken as remediation.

Table 1 intends to sum up the translations of the above examples using A-PPL language.

3.1.5. From AAL into A-PPL: A simple policy example

In addition to the previous examples which consist in translating each obligation in AAL and A-PPL, in this section, we illustrate the mapping of AAL into A-PPL with a concrete (simple) policy example.

In a first step, the hospital, namely the data controller, defines the following rules in its own policy:

- Any patient (DS) can have read/write access to his personal data. (cf. obligation 6)
- Relatives can have read/write access as well. (cf. obligation 6)
- Patients should be informed by mail whenever there was an attempt to access their data and this was permitted. (cf obligation 22)
- Any access attempt (be it permitted or denied) should be logged. (cf. obligation 22)
- Patients data should be deleted after 2 years. (cf. obligation 7)

When Kim is registered to the hospital as a patient he defines some specific preferences which are the following:

- Allow Sandra (my relative) to have only read access to my data.
- Store my data only for 6 months.
- Notify me whenever there was an access authorization to my data.

In the next paragraph, we first map both the hospital's rules and Kim's preferences into AAL clauses and further create the final policy including some of Kim's preferences.

Mapping obligations and preferences to AAL policies. A first translation of the hospital's clauses and Kim's preferences appears in Listing 6 and in Listing 7, respectively.

Listing 6: Initial provider clause

```
1 // The initial provider clause
2 ALWAYS (
3   FORALL p:Patient FORALL r:Agent FORALL a:Agent
4   FORALL file:Data WHERE file.subject=p
5   (PERMIT p.read[file]() AND PERMIT p.write[file]() AND
6    IF relative(r, p) THEN
7      (PERMIT r.read[file]() AND PERMIT r.write[file]())
8    AND (PERMIT d.read[file]() AND PERMIT d.write[file]())
9    AND PERMIT hospital.delete[file]()
10   AND IF (a.read[file]() AND PERMIT a.read[file]()) THEN
11     hospital.notify[file.subject]("Authorized␣Access")
12   AND IF (a.write[file]() AND PERMIT a.write[file]()) THEN
13     hospital.notify[file.subject]("Authorized␣Access")
14   AND IF (a.read[file]()) THEN hospital.log(a, 'read', file)
15   AND IF (a.write[file]()) THEN hospital.log(a, 'write', file)
16 )
17 AND (MUST hospital.delete[file]() AFTER 2 years)
```

Listing 7: Initial user preferences

```
1 // The initial user preferences
2 (ALWAYS FORALL file:Data FORALL a:Agent
3   (IF (file.subject=Kim) THEN
4     (PERMIT Sandra.read[file]() AND DENY Sandra.write[file]()
5      AND
6      (IF (a.read[file]() AND DENY a.read[file]()) THEN
7        hospital.notify[file.subject]("Authorized␣Access")
8      AND IF (a.write[file]() AND DENY a.write[file]()) THEN
9        hospital.notify[file.subject]("Authorized␣Access"))))
10  )
11  AND (MUST IF (file.subject=Kim) THEN hospital.delete[file]()
12        AFTER 6months)
```

Once this first version in AAL is provided, as explained in Chapter 2, we can use a logical prover to check the consistency of each policy. We skip here the details of the translation and the behavior of the consistency prover, TSPASS. During this first step, we can easily check that both clauses are logically satisfiable. However, we cannot prove that the hospital's clauses ensure the Kim's preferences. Indeed, the same prover further detects an inconsistency between both policies: indeed, while the hospital allows relatives to have read and write access over the patient's data, Kim requires that Sandra has a read access only; moreover, Kim also wishes the deletion of his data after 6 months instead of 2 years as stated at the hospital's policy. Following an agreement between the hospital and Kim, the final policy resulting from the matching of the hospital's clauses with Kim's preferences is shown in Listing 8. This final policy is now ready to be translated into the machine-readable A-PPL language.

Listing 8: Final policy

```
1 // The final policy
2 ALWAYS (
3   FORALL p:Patient FORALL a:Agent FORALL r:Agent
```

```

4  FORALL file:Data WHERE file.subject=p
5  PERMIT p.read[file]() AND PERMIT p.write[file]()
6  AND IF relative(r, p) THEN
7      (PERMIT r.read[file]() AND PERMIT r.write[file]() AND
8      IF(r = Sandra AND p = Kim) THEN DENY Sandra.write[file]()
9      )
10 AND (PERMIT d.read[file]() AND PERMIT d.write[file]())
11 AND PERMIT hospital.delete[file]()
12 AND IF (a.read[file]() AND PERMIT a.read[file]()) THEN
13     hospital.notify[file.subject]("Authorized Access")
14 AND IF (a.write[file]() AND PERMIT a.write[file]()) THEN
15     hospital.notify[file.subject]("Authorized Access")
16 AND IF (a.read[file]() THEN hospital.log(a, 'read', file
17 )
18 AND (MUST hospital.delete[file]() AFTER 2 years)

```

From AAL to the final A-PPL policy. Based on section 2.2, we present a snippet of the use case translation in A-PPL. The authorizations (read/write/delete) are translated in rules using the "effect" attribute to distinguish permissions and prohibitions (see rule 1 in 2.2). For example the translation of *PERMIT hospital.delete[file]()* is as following :

Listing 9: Hospital permissions

```

<Rule Effect="Permit" RuleId="Hospital_access">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="function:string-equal">
          <AttributeValue DataType="string">hospital</AttributeValue>
          <SubjectAttributeDesignator DataType="string" AttributeId="
            subject:subject-id"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Actions>
      <Action>
        <ActionMatch MatchId="function:string-equal">
          <AttributeValue DataType="string">delete</AttributeValue>
          <ActionAttributeDesignator DataType="string" AttributeId="
            action:action-id"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

```

On the other hand, whenever such a clause includes conditional expressions (like IF exp1 THEN exp2), then, following rule 6 in 2.2, these clauses are translated into Obligations (Trigger/Action pairs) in APPL. The following listing, translates the notification clauses defined in Listing 8 from Line 11 to Line 14.

```
<!-- Obligations set-->
<DataHandlingPolicy >
  <ObligationsSet>
    <!-- Notification of Patient on authorized access -->
    <Obligation>
      <TriggersSet>
        <!-- A-PPL trigger -->
        <TriggerPersonalDataAccessPermitted>
        </TriggerPersonalDataAccessPermitted>
      </TriggersSet>
      <!-- A-PPL action -->
      <ActionNotify>
        <Media>e-mail</Media>
        <Address>
          <Apply FunctionId="function:string-one-and-only">
            <AttributeDesignator DataType="string" Category="subject -
              category:access-subject" AttributeId="subject:subject-id
              "/>
          </Apply>
          <AttributeSelector Category="attribute-category:resource"
            Path="//db:patient/db:patientMail/text()" DataType="string
            "></AttributeSelector>
        </Address>
        <Recipients>
          <Apply FunctionId="function:string-one-and-only">
            <AttributeDesignator DataType="string" Category="subject -
              category:access-subject" AttributeId="subject:subject-id
              "/>
          </Apply>
          <AttributeSelector Category="attribute-category:resource"
            Path="//db:patient/db:patientId/text()" DataType="string"/
            >
        </Recipients>
        <Type>Authorized access</Type>
      </ActionNotify>
    </Obligation>
  </ObligationsSet>
</DataHandlingPolicy>
```

The final A-PPL policy is provided in appendix E.

3.2. Application to Data Transfer Laws

To permit a data transfer, the regulation requires that some rules should be satisfied by the transferor or/and the transferee of the transferred data. Appendix D shows a model for data transfer regulation proposed by Queen Mary University of London (QMUL). This model stems from articles of the reformed European Commission's data protection rules. The language we design should be able to translate into machine-readable policies such requirements. The aim of this section is to show whether it is possible to generate AAL and A-PPL statements of data transfer rules.

3.2.1. A selection of the data transfer rules

Partners from QMUL studied articles 40, 41, 42 and 44 of the European Commission's Proposal for a General Data Protection Regulation (Parliament LIBE text⁶). From this analysis, they produce a simplified list of five rules that represent the aforementioned articles. These rules define the conditions a data transfer is permitted or prohibited. For example, QMUL defines Rule 2 where an objective test is required to allow the transfer of a particular piece of data. In this rule, six tests are described: if one of these tests passes then the transfer is permitted. They consist in (i) checking that the receiving country ensures adequate level of protection according to the European Commission, (ii) checking that the relevant processing sector within the receiving country ensures adequate level of protection according to the Commission, (iii) checking that both transferor and transferee possesses a valid European Data Protection Seal, (iv) checking that the transfer is in accordance with binding corporate rules approved by transferor's data protection authority, (v) checking that the transfer is in accordance with contractual clauses approved by transferor's data protection authority and agreed with the transferee, (vi) and checking that the transfer is in accordance with standard data protection contractual clauses and included in the contract between the transferor and the transferee.

Similarly, QMUL defines Rule 3 in which a mix of objective and evaluative test is required to allow a transfer: the data subject must consent to transfer and a professional advisor should validate that the consent meets the law's requirements.

Professional advice is considered as evaluative evidence for a data transfer. They may be crucial for allowing a data transfer. For example, Rule 4 from QMUL specifies that a professional advice concerning the necessity of the transfer should be given.

3.2.2. The AAL point of view

We, of course, agree that it is impossible to code all the subtleties contained in law. Furthermore, there are things we want to automate and to represent in a machine understandable and processable way, and others we do not want (for instance "the judge deliberates and decides that you are guilty"). An open question is how to distinguish these two kinds of information? To improve and to make the description of rules more precise our suggestion is to give more details about the agents and their exact interactions. This may not be a usual practice for lawyers, but computer scientists can help in doing that. The idea is to define a kind of process flow between the agents, expressing the consent process, data communications, any things relevant in the context. Assuming that we have such an abstract design we provide a preliminary analysis of these rules with a first formalization of some of them.

Let us start with rule 3 which is stated as follows:

```
IF
  Data subject has consented to transfer
  AND
  Professional advice that consent meets the laws requirements
THEN
```

⁶<http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52012PC0011>

transfer is permitted

```
FORALL d:Data, ds:DataSubject, transferor, transferee : DataProcessor,
  EXISTS prof:ExternalAdvisor (personal(ds, d) AND ds.agree4Transfer(d,
    ...) AND prof.agree())
THEN PERMIT transferor.transfer[transferee](d)
```

The "consent protocol" can be more detailed and complex. For instance, the data subject or the ExternalAdvisor could use some checking tools helping in verifying the compliance at least for the part of the contract which is machine understandable.

Rule 4 has a similar structure and can be represented in AAL provided that we get the details of all the agents interactions. The first rule (called general rule) was described as follows:

```
IF
  Transferee is to a receiving country outside the EEA
  AND
  None of rules 2, 3 or 4 is satisfied
THEN
  Transfer is prohibited
```

Our AAL translation is as follows:

```
FORALL d:Data, transferor, transferee : DataProcessor
  (personal(ds, d) AND not (transferee.country in EEA) AND NOT eval-cond(
    rule2, ...))
  AND NOT EXISTS prof:ExternalAdvisor (personal(ds, d) AND ds.
    agree4Transfer(d, ...) AND prof.agree()) AND NOT eval-cond(rule4,
    ...))
THEN DENY transferor.transfer[transferee](d)
```

This rule is more complex as it refers to application conditions of other rules, namely 2, 3, and 4. We explicitly insert the condition coming from rule 3. But for readability and conciseness it seems desirable to define names for rules and a means to evaluate a rule condition. This is the intended meaning of `eval-cond(rule2, ...)` and `eval-cond(rule4, ...)`. This is not an issue but we should take care of quantifiers.

The two remaining rules (2 and 5) illustrates some classic difficulties. To give an AAL description of rule 2 it lacks some details. We could abstractly represent some external information, we can consider fake actors with the needed services.

The last rule (rule 5 onward transfer) is written as follows:

```
IF
  Rule 2, 3 or 4 were applied to permit the original transfer
  AND
  The transferee proposes to transfer the personal data
  to another receiving country outside the EEA
  AND
  Rule 2, 3 or 4 applies to permit the new transfer
THEN
  transfer is permitted
```

It raises some issues, mainly about understandability. Does it mean that conditions of "rule 2, 3, 4 are satisfied" or "rule 2, 3, 4 are really applied" ? Evaluating twice the rules 2, 3, 4 seems odd and surely not the real intention. One way to interpret it is:

```
IF
  Conditions from rules 2, 3 or 4 are satisfied by the original transfer
AND
  Conditions from rules 2, 3 or 4 are satisfied by the new transfer outside the EEA
THEN
  transfer is permitted
```

In other words that means that the normal conditions (from rules 2, 3, 4) for the transfer must be checked along the forward chain. With this interpretation there is no issue to translate it into AAL.

3.2.3. Our proposal of extension of A-PPL for data transfer rules

In this section, we describe a tentative of translation of these rules (Appendix D) into A-PPL statements. This task requires to extend A-PPL with new features that enable to specify the conditions under which data transfers are permitted.

The original PPL specification proposes the `AuthzDownstreamUsage` environment (in a Data Handling Policy) that allows to specify whether the transfer of data to a third party is permitted. This environment embeds an additional policy that the data controller must enforce when it transfers the data (e.g. access control policy). However the conditions of such an authorization are not specified in the policy. Indeed, the decision whether the authorization is `TRUE` or `FALSE` cannot be conditioned by rules specified in the policy (such as the data transfer rules).

A-PPL should allow the specification of the conditions that have to be satisfied prior to data transfer. Therefore, we propose to extend the `AuthzDownstreamUsage` element in such a way that the conditions of the transfer are clearly defined in the policy.

In the following we consider that the cloud service provider sends a data transfer request before it sends the actual data to third parties. This assumption eases the specification and the enforcement of data transfer rules. In case of access requests, PPL allows to specify provisional actions, inside the `<ProvisionalActions>` element, describing the actions (e.g., revealing attributes or signing statements) that have to be performed by the access requestor in order to be granted access to a piece of data. Therefore, in our A-PPL language, we propose to extend the existing `<ProvisionalActions>` element to enable the specification of the actions that have to be performed by the data transferor to be authorized for the transfer. These actions are related to the rules presented in the proposed data transfer regulation (see Appendix D). These rules consist, for instance, in checking that the receiving country's level of protection is in accordance with the regulation or in getting professional advice on the transfer. The extended `<ProvisionalActions>` element should be nested in an `<AuthzDownstreamUsage>` environment. The transfer is then permitted when the conditions on the provisional actions are satisfied. In order to specify conjunctive and disjunctive transfer rule conditions (such the ones

represented in Appendix D), the A-PPL policy uses the XACML <Condition> element in order to list the data transfer conditions, <Apply FunctionId=function:and> tag in order to represent a conjunctive set of rules and <Apply FunctionId=function:or> tag to specify a disjunctive set of rules.

In a nutshell, the data handling policy that states the data transfer rules looks like the policy listed in Listing 10.

Listing 10: Format of a data transfer policy

```
<DataHandlingPolicy>
  <AuthorizationsSet>
    <AuthzDownstreamUsage>
      <Condition>
        <Apply FunctionId=function:or>
          <ProvisionalActions>
            <ProvisionalActions>
          </Apply>
        </Condition>
      </AuthzDownstreamUsage>
    </AuthorizationsSet>
  </DataHandlingPolicy>
```

In particular, they consist on checks on particular conditions that have to be fulfilled in order to be granted the authorizations for transfer. We identify five actions to perform: *GetConsent*, that requests the consent from the data subject on the data transfer ; *CheckReceivingCountry*, that verifies that the receiving country is in the list of the countries approved by the European Commission ; *GetValidDPSeal*, that requests a valid European Data Protection Seal ; *CheckContract*, that verifies that the transfer is in accordance with statements specified in contracts and *GetProfessionalAdvice*, that request professional advice for data transfer compliance.

Table 4 in Appendix D lists and describes the aforementioned provisional actions we propose.

3.2.4. Difficulties and limitations in data transfer policy enforcement

Enforcing the specified rules using the proposed extension is a relevant issue that we do not focus on at the time of writing this document. However, we have an insight of how this can be performed in the A4Cloud project.

We recognize that controlling data transfers is a difficult function to ensure. This mechanism may require that the transferor requests to the data subject (or any competent authority) the authorization for the transfer before sending the data to third parties. On receiving such a request, the transfer controller evaluates the conditions of the transfer against data transfer policies. If the conditions match the data transfer rules, the transfer is permitted and the transferor can proceed the actual transfer. This setting may in certain cases be not realistic when transfers are made without requesting authorizations. On the other hand, we think that monitoring data transfers after the actual transfer should provide a detective control which aims to observe data transfer rules violations. The A4Cloud Data Transfer Monitoring Tool (DTMT - see Section 4.3.2) logs the actual transfer for the compliance checking of data transfer operations against policies.

On the other hand, certification may appear to be one possibility for enforcing the data transfer rules. For example, to check whether the transfer is in accordance with contractual clauses, one may request a certificate that validates the transfer. If we envision to get the certifications required by the rules on-the-fly, these rules will not easily be enforceable since the policy enforcement will require a human intervention that certifies the compliance of transfer with contractual clauses or that gives the required professional advice. We may envision instead an a priori certification of such possible transfers. The resulting certificates may be stored somewhere on the transferor's side. In other words, when a transfer is requested, the request may come with such professional advices or certifications of the conformity of the transfer. These ones may consist in a record that specifies, e.g. who gives the advice and how long the advice lasts. Therefore, the Policy Engine will just check the existence of such record in its repository.

4. Policy Enforcement Methodologies

There are multiple techniques that can be adopted for policy enforcement in the cloud, that range from “invasive” methods, requiring to rewrite code and to redeploy applications on the service provider side, which we call white-box policy enforcement, to less intrusive ones, that require some sort of black-box integration of additional security components, for instance, from the policy enforcement architecture such as Policy Information, Enforcement and Decision points as prescribed in the XACML standard [OAS]. The trade-offs of these two alternatives for policy enforcement are explained in Section 4.1. These trade-offs were taken into consideration for the design of the A-PPL engine, named A-PPLE in Section 4.2. In Section 4.3, we describe some enforcement mechanisms that could be integrated with A-PPLE to better meet accountability obligations as identified in C2. Notably, we present a set of preventive techniques that assure that data in the cloud is handled correctly, and a number of detective techniques that empower auditors and cloud customers alike with the ability to investigate and assess CSP’s compliance. Finally, Section 4.4 wraps up the chapter.

4.1. Policy Enforcement Approaches

4.1.1. White-Box Enforcement Methodology

The complexity and dynamics of cloud scenarios limit transparency and suggest to complementary introduce controls to achieve accountability. This need is emphasized by the fact that the majority of existing providers’ offers are focused on price which results in a cost pressure leading to obscure contracts and a lack of supporting services. Control mechanisms allow both the cloud provider and the customer to establish a model of the cloud’s behaviour, in order to customize a set of mechanisms that enact control on how data is being handled, how processes are executed, and by whom. Supporting the cloud consumers to impose control allows increasing the level of trust related to the expected behaviour of the provider’s services, and compels the cloud service provider to meet organizational compliance, and/or regulatory requirements [LdOS13]. One way of achieving this level of control is to invasively modify existing software code [Aßm03] to control the behaviour of the service according to the desired accountability policy.

Very probably CSPs in the vast majority of the cases will not allow Cloud Customers to deploy directly their own security controls in the cloud infrastructure (or platform or software depending on the delivery model). However a certification authority may be empowered to integrate policy engines that can modify the existing service. We explain an appropriate approach in the following.

As service-oriented architectures (SOAs) are central to the cloud delivery models, we introduced in a work prior to A4Cloud [ISR⁺11] a model for aspect-oriented composition for SOA, which is able to provide the level of control necessary in today’s cloud. It is founded on the concepts of horizontal and vertical compositions, providing the cloud consumer with control at different abstraction levels, targeting different cloud delivery models. Aspects are particularly suited to implement the type of controls proposed to support accountability, including filtering, notification, deletion and retention. These are generic controls that cross-cut platforms and ser-

vices (e.g., “filter all access requests to the persistence layer by the customer-specified policy”) and, founded on the concepts of aspect-orientated programming. Aspects reduce the effort to implement the controls, since they can be treated as a separate concern. This is also key to achieve scalability.

We also had a previous experience in the definition and implementation of accountability controls using service aspects by means of an example domain in [YSSdO12], where we enforced privacy obligations as an instance of accountability controls in the context of appropriate handling of personal information and compliance with data protection regulations.

The disadvantages of the white-box approach are the heterogeneity of the cloud computing technology platforms. In order to work correctly, the aspect oriented approach needs to rely on libraries and frameworks that depend on the programming language used to implemented the cloud services that need to be controlled. There are very few standards we can rely on, requiring to adapt the enforcement tools for each specific use case. The advantages are high degree of confidence in the enforcement, because there is more transparency about the policy enforcement points. Another advantage is increased performance, when compared to black-box approaches, introduced in the following.

4.1.2. Black-box Enforcement Methodologies

We consider in this section a number of approaches that require modification on the cloud service provider architecture without necessarily changing the core application behaviour, using a black-box methodology [dOCSS14, CDR⁺13, DSI⁺13, CS14].

In [DSI⁺13] we present an implementation relying on distributed reference monitors, which control the flow of information between services. A reference monitor works as an application-level gateway (ALG), that is, a firewall-like entity that is put between services and the rest of the network; its responsibility is to filter and mangle traffic according to the policy rules. ALGs for web services have been studied by academia [GL06], [BCEPM08], industry [Pat07], and standard bodies [SWS07]; commercial solutions [CIS10, CIT10] are also available. Compared to these approaches, the work in [DSI⁺13] provides a framework to configure reference monitors according to a set of more abstract overarching security policy, automating the application of matching policies in different reference monitors (e.g., message encryption on the sender side is matched with decryption at the receiver).

Another black box approach consists in defining accountability as the composition of non-functional concerns. In [dOCSS14] we present a model-driven approach to accountability in business processes. We focus on the technical realization through the composition of individual actions, regarding distinct security properties. The actions and their compositions are then combined with the base process model using the NComp modeling framework [SCM11]. This generic framework covers the life cycle of non-functional concerns from modeling to execution. It provides a rich toolset consisting of different modeling editors and code generators. NComp supports defining non-functional concerns (NFC's) and the actions realizing them in business processes. Further, it allows defining compositions of non-functional behaviours with each other and with the functional behaviour described for example using BPMN2 or WS-BPEL. In particular, we focus on the correct composition and representation of various security concerns such as identity management, secure logging, privacy, and auditing during the modeling phase

to achieve accountability.

A similar enforcement methodology is presented in [CS14], where abstract model for the enforcement of accountability policies was proposed. The work introduces the possibility to use libraries of patterns for relevant accountability attributes such as transparency and remediation, thus covering preventive and corrective mechanisms. In order to achieve that, the model involves the transformation of service compositions, introducing a predicate language for accountability, as well as scopes and schemes for the construction of executable accountable policies. In [CCS13] real-world service infrastructures are modeled in terms of three abstraction levels and that (partially invasive) adaptations involving all levels are used to handle the evolution scenarios. The method was implemented and tested over a popular web service framework with an application to secure logging.

The main advantage of using black-box methodologies to enforce accountability in the cloud is the reduction in the risk of rejection by cloud service providers. The less invasive approach to integrate policy engine components to the already existing service landscape would reduce the time to implement the accountability assurance brought by the policy engine. Another advantage is the minimization of problems raised by the technological heterogeneity of the service implementations.

In this section we presented two main approaches for policy enforcement in the cloud we experimented before [ISR⁺11, YSSdO12, CDR⁺13, DSI⁺13] and during the course of the work package [dOCSS14, CDR⁺13, CS14, CCS13]. This allowed us to analyze formalisms more suited to accountability enforcement. Both approaches are able to fulfill the A4Cloud goals concerning data governance policies - reliable enforcement, without compromise to auditability and certification. We believe that the black box methodology has more potential for adoption, since it raises fewer integration problems without rewriting existing software components on the cloud provider side, for the majority of the use cases. The work carried out in D3 - tools for policy enforcement and compliance is taking into account the results reported here for the definition and implementation of the policy engine architecture.

There are limitations of policy enforcement methodologies with respect to the security they can offer. We assume that accountable providers will seek to actively promote transparency. Thus, they will cooperate in the accountability policy enforcement. The cloud provider cannot play the attacker role in our threat model. The security mechanisms we designed cannot prevent access from law enforcement or intelligence agencies in all situations.

4.2. Approach taken in A4Cloud: A-PPLE

4.2.1. Overview

The A-PPL engine (A-PPLE) can use the white-box or black box approaches to enforce A-PPL policies. As the engine is in charge of analyzing A-PPL policies statements and executing the access and usage control as well as obligations related to personal data handling in the cloud, it is sufficient to configure the **Policy Decision Point** (PDP) and **Policy Enforcement Point** (PEP) in such a way that requests to the protected resources cannot be bypassed by the entities performing the access request. The enforcement strategy can be chosen depending on the scenario and on the trust relationship among the cloud actors and the delivery and service

models in question.

For instance, suppose that a data controller is using a public cloud infrastructure to deploy an application processing personal data. It can apply the white-box approach at the software level by integrating the engine directly to the application code (under its responsibility and ownership). On the other hand, the IaaS provider may agree to integrate the engine into its service architecture, but not allow it to modify the IaaS service code base, in this case taking a black box approach. In the remainder of this section, we explain the A-PPLE architecture. In summary, the engine's design (based on the XACML standard) is able to accommodate a combination of enforcement strategies that will better respond to the needs of a given target cloud service ecosystem.

The target use-case of A-PPLE is protecting the access to PII (personally identifiable information⁷) in a way that can be later on audited in order to ensure accountability. For that A-PPLE defines an interface consisting of a few methods for storing and retrieving PIIs. Upon storing the PII, the engine receives also a policy (written in A-PPL) that contains the rules related to how that piece of personal data has to be handled. Next we describe the components of A-PPLE (See Figure 7). When PII is retrieved from the engine the policy associated with the personal data is analyzed in by the **Policy Decision Point** that takes into account access and usage control rules.

Also additional obligations related to the personal data handling are executed when certain events occur inside the engine. That part of A-PPL policy is handled by the **Obligation Handler** that analyzes the triggers and actions which an A-PPL obligation consists of. Such obligations are related either to timely scheduled events or events related to PII life cycle, e.g. personal data retrieval or deletion.

All actions happening inside A-PPLE are logged by a central component called **Logging Handler**. It stores the log entries related to decisions and actions concerning the PIIs stored inside the engine. The log entries can be further used during either internal or external audit to analyze whether the policy statements were correctly enforced.

A-PPLE adopts a two-layer high-level architecture to enforce isolation of engine components and data: The core elements of the policy engine providing the enforcement functionality reside in the core layer. All personal data are stored in the **PII Store** that resides in the Persistence layer. Access to the persistence layer is mediated through a **Persistence Handler** component which abstracts the underlying location and storage data model to the core layer functions above.

The **Policy Decision Point** (PDP) is the central element of A-PPLE, responsible for taking access control decisions in regards to a request. A-PPLE's PDP implementation relies on the access control engine implementation based on HERAS (Holistic Enterprise-Ready Application Security Architecture Framework⁸) for the evaluation of the XACML part of PPL policy⁹. Apart from the standard attribute-based access control, the other information evaluated by the PDP

⁷For historical reasons, as the privacy language PPL reused many constructs from international standardization efforts, such as P3P, we use the term PII in the language. It may be understood as the equivalent term to personal data, used in European texts.

⁸<http://www.herasaf.org/about.html>

⁹More details about the requirements for the PDP (and other components of the XACML standard) are given at <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html>

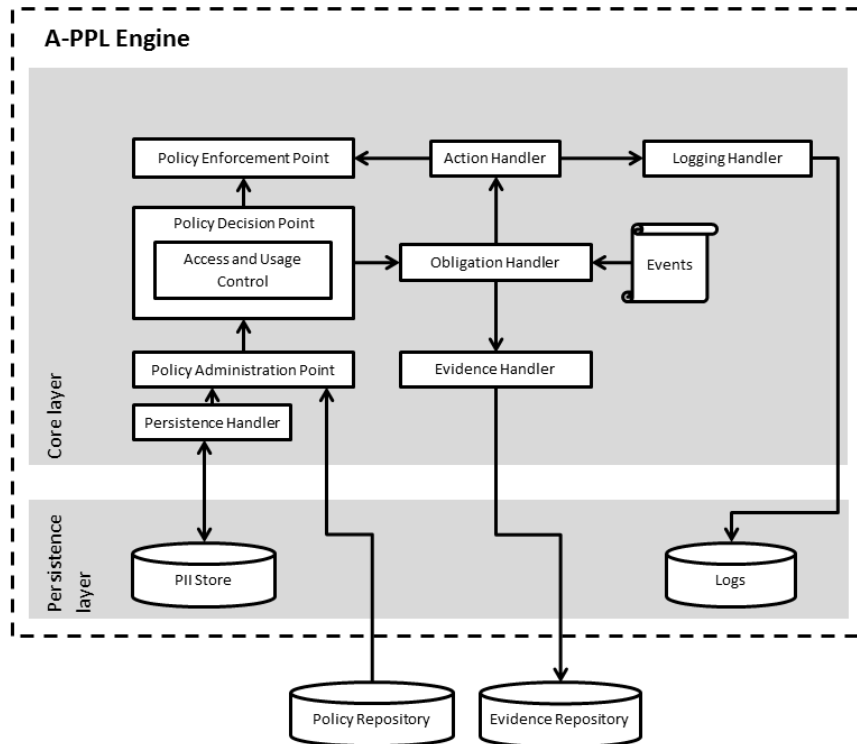


Figure 7: A-PPLE components

at the step of access control decision is usage authorization.

The **Policy Enforcement Point** (PEP) takes the decision made at the PDP and enforces it, meaning that if the PDP decided to allow access to a piece of data, this data is fetched, formatted and sent to the receiver through the PEP. The PEP acts as an orchestrator of the enforcement process, orchestrating two modules: the Action and Obligation Handlers. The main component taking responsibility of executing A-PPL obligations related to the personal data handling is the **Obligation Handler**. It is complemented by the two components which serve as the interface and the execution environment for the obligations: the **Event Handler** and the **Action Handler**. The three components are making up A-PPLE. As the engine needs to store the internal representation for the obligations, derived from the A-PPL schema, the **Obligation Handler** has three functions:

- Recording the obligations associated with the PII's added to the **Policy Administration Point**, which handles the parsing of the policy, given in XML format, i.e. unmarshaling the policy elements, interpreting the declarations contained in the them, and configuring the other components accordingly.
- Retrieving that obligation based on associated PII and the trigger.
- Removing the obligations associated with the given PII. It also has a build-in scheduling

mechanism for executing the actions associated with the time based triggers.

The **Event Handler**, which is part of the **Obligation Handler**, serves as the entry point to trigger the events which are responsible for the event based triggers. The actual action execution happens inside the **Action Handler** module.

The **Logging Handler** provides an extensible secure logging interface to A-PPLE which supports all logging related functionality during the policy enforcement and personal data handling process. The **Logging Handler** provides interfaces that other components, mainly the **Obligation Handler**, can use to register that a certain action related to the personal data life cycle took place. The **Logging Handler** can optionally support non-repudiation on top of the basic secure logging function by utilizing a separate Crypto Module to provide the necessary cryptographic operations needed.

The **Evidence Handler** is responsible for the compilation of evidence based on engine operations. Its purpose is to support scenarios where the policy requires evidence to be collected as part of its definition. As the work on evidence collection is at an early stage, at present it is only proposed that the Evidence Handler interacts with the Obligation Handler component.

The **Policy Repository** provides storage for generic A-PPL policies composed by the AccLab tool¹⁰ [BGR⁺14] or manually, which A-PPLE will retrieve and possibly modify based on the context of data disclosure. It is assumed that the Policy Repository is separate from the PII Store.

It is important to remark that the architecture described here is subject to evolution as the development of the A4Cloud toolkit progresses in work package D3 - Tools for Policy Enforcement and Compliance. The reader should refer to future D3 deliverables in order to have updated architectural descriptions as well as sequence diagrams explaining the runtime behaviour of the components.

4.2.2. Trust assurance

Certification One way of increasing trust in the policy enforcement engine itself or any proposed enforcement technique is of course certification. The certification process consists of evaluating the maturity of the implementation of the process. Current certification methods either assess the technical expertise and knowledge of the organisation implementing the process or verifies whether the system concerned meets the technical requirements: Professional certifications approve that the certified person or organisation is competent in performing the required task. For example, CSA's STAR Certification¹¹ evaluates the security of cloud services and certifies their level of maturity; on the other hand, product certifications test softwares and assure their implementation meets the functional specifications. One of the main aims of the A4Cloud Assertion Tool which is defined under work package D-6 is to validate that the A4Cloud tools provide "correct and trustworthy guarantees". Therefore, the A4Cloud Assertion Framework can also assess the correctness of the proposed policy enforcement tool.

¹⁰<http://www.emn.fr/z-info/acclab/> visited on 30 July 2014

¹¹<https://cloudsecurityalliance.org/star/certification/> accessed on 30 July 2014

Tamper-proof Hardware The Trusted Computing Base¹² (TCB) of a computer system is the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system. From the software standpoint, to rely on a TCB enables the execution of security sensitive APIs within a platform assuming that the underlying system has not been compromised. Security sensitive APIs typically encompass system kernel calls and by extension hypervisor services in the cloud domain, encryption services and secure storage (hard drive encryption), identity management services such as authentication (secure token), authorization, auditing, and digital right management (proof of ownership). Executing programs within a TCB is a very strong assumption especially for complex hardware and software stack that are routinely under hacker scrutiny.

Standardized low-level environments providing a minimal TCB environment based on specialized hardware are already widely available since the introduction of Trusted Platform Module in enterprise level computers in 2006. More recently, the evolution in hardware and hardware virtualization has enabled the development of chips with a specific set of instructions dedicated to security interacting with a TPM or with dedicated or virtualized components (e.g., AMD SKINIT, ARM TrustZone¹³, MIPS VZ, Intel TXT Trusted Execution Technology). Typically, these instructions provide a protected execution platform that allows for a secure booting, ensuring thus that all system software components are in a known and “trusted” state and enable attestation establishment together with cryptographic binding of the device and data sealing. The backbone of the available TCB approaches is cryptography which is used to ensure security and foster trust between different security modules underpinning the TCB. These modules can be offered by software, dedicated hardware or by virtualized hardware.

4.3. Suggested mechanisms to enforce accountability obligations

In this section, we suggest some enforcement techniques suitable to accountability obligations. We classify these techniques into two categories:

- **preventive** solutions that usually protect data in the cloud are usually implemented to meet data handling obligations such as access or usage control rules;
- **detective** mechanisms would usually complement preventive ones and assure transparency by implementing proposed logging/monitoring solutions.

4.3.1. Preventive techniques

One of the most important and popular contractual obligation is the implementation of appropriate security and privacy measures. Since the translation of such an obligation into machine readable policy is not straightforward, cryptographic enforcement solutions such as encryption or integrity techniques and proper access control mechanisms become mandatory. Obligations related to the protection of the data can therefore be partially met thanks to the use of such preventive solutions.

¹²http://en.wikipedia.org/wiki/Trusted_computing_base, accessed on 30 July 2014

¹³<http://www.openvirtualization.org/open-source-arm-trustzone.html>, accessed on 30 July 2014

Deliverable D:C-7.2 [FHNOP14] which provides some privacy by design guidelines for an accountability tool developer, describes a number of Privacy Enhancement Technologies (PETs) among which some of them can be considered as potential candidate for the policy enforcement engine. First of all encryption techniques of course become mandatory for the protection of the data stored in the cloud. A proper identity management system combined with some access and usage control solutions will define and verify specific credentials to grant and or deny the access and usage of specific data.

4.3.2. Detective techniques

One way to assure that cloud providers are compliant with their contractual obligations is to enable cloud customers, auditors and regulators to verify the compliance of the operations executed by the cloud with the regulations and SLAs. This calls for the use of a number of detective techniques that grant cloud customers and third party auditors the ability to assess and evaluate the cloud providers' compliance. These detective techniques range from cryptographic proofs of computation and storage to audit reports and secure logs, however in C-4, we only focus on solutions related to *secure logging* as it allows for a fine-grained analysis of cloud behaviour and performances.

Secure Logging: As a matter of fact, logging stands for the action of keeping a description of noteworthy events as they happen. Notably, logs contain relevant information that help IT administrators or auditors analyze and better understand the recorded performances and the detected security breaches. This explains why the integrity of logs is regarded as a critical functionality that has to be assured whenever possible. This has given rise to an active area of research on secure logging which aims at protecting logs from deliberate modification and unauthorized destruction. Despite the diversity of the solutions proposed for secure logging, two major lines of research can be identified:

- **Hardware-based secure logging:** As the name implies, solutions in this category relies on tamper proof hardware to ensure that unauthorized modification of logs cannot go undetected [CPH03, CW09]. Roughly speaking, the logs are stored on tamper-proof memory whose access is controlled, that is, only authorized entities can write into that memory.
- **Cryptography-based secure logging:** Most of logging solutions falling into this category assume that the logger (i.e. the entity writing the logs) is trusted yet it is prone to attacks. As a result, they only ensure the *forward integrity* of logs, which entails that only the integrity of logs recorded prior to a security breach is assured. To achieve forward integrity, these cryptographic solutions generally rely on key update mechanisms combined with hash chains and message authentication codes (MACs) [BY97, SK98, MT09]. However, the use of message authentication codes as the underlying integrity mechanism restricts the practicality of those solutions as the entity verifying the logs should be given the MAC secret key and consequently should be trusted. Therefore, a new line of research has followed which attempts to provide cryptographic primitives that allow

for the construction of publicly verifiable logs. For instance, [BM99] introduced forward secure digital signatures which can be integrated into secure logging solutions to replace message authentication codes, and therewith enable public verifiability of logs. Nevertheless, this solution is far from being practical as the size of public key of the digital signature grows linearly with the number of entries in logs.

The work of Crosby and Wallach [CW09] improves over existing solutions by considering a *stronger adversary model* where the logger is assumed to be malicious (i.e. the logger may attempt to tamper with the stored logs deliberately). The solution proposed in [CW09] builds upon Merkle trees to enable the logger to generate proofs of compliance at the auditors' request. Roughly speaking, the idea underlying this scheme is that once the logger tampers with the logs it stores, it can no longer provide valid proofs to the auditor, and consequently the log modification will be detected. Although this solution is practical as it takes into account a more realistic adversary model, it requires a synchronization between the different auditors verifying the logger's behaviour which may not be guaranteed all the time.

In what follows, we will present a two dedicated solutions for secure logging. The first one is Data Transfer Monitoring Tool (DTMT for short) which builds upon tamper proof hardware to assure that transfers of personal data in the cloud are in accordance with regulations. The second solution employs cryptographic methods to construct logs that are secure even in the face of a *malicious logger* and publicly verifiable by third party auditors.

Data Transfer Monitoring Tool (DTMT) The purpose of the DTMT is to enable cloud service providers to demonstrate compliance with personal data protection and other regulations regarding where and by whom the processing occurs. The tool automates the collection of evidence that obligations concerning personal data transfers are being carried out properly by generating logs about the operations performed on personal data involving transfer at the infrastructure level, for example when performing load-balancing or creating backups and storing them in different hosts. Our current prototype intercepts network traffic at a IaaS host running OpenStack¹⁴. An important part of our research consists in identifying which calls to the IaaS API software stack are relevant to monitor data transfers. Currently this is done manually using the available documentation and when it is insufficient, we inspect the source code of OpenStack. In order to generate the necessary logs we initially used the Wireshark tool¹⁵, but we are currently writing a new software component to facilitate integration. The approach of intercepting the communication at the network level has the advantage of being hard to bypass, offering reliability to the log generation process. On the other hand it may raise confidentiality issues, as data concerning multiple tenants are processed concurrently. In order to avoid confidentiality breaches, our plan is to use the Transparency Log tool to provide the necessary security features for the log handling in DTMT.

DTMT analyzes these logs using rules, helping an auditor to identify whether all transfers were compliant with the authorizations from the DPA obtained beforehand by the data con-

¹⁴<https://www.openstack.org/>

¹⁵<http://www.wireshark.org/>

troller. In this way, policy violations can be identified. 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 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. Notifications can be triggered by the auditor. The tool relies on A-PPLE to carry out the necessary actions to enforce the policies related to the events it generates and on other tools to act upon the detected violations (e.g. for notification or remediation).

Inescapable Logging with Dispute Resolution In this section, we introduce a solution for secure logging in cloud environments. More precisely, the propounded solution assures that cloud providers cannot deny that a cloud customer submitted an operation request, neither can cloud customers repudiate that they asked the cloud to execute some job on their behalf. We differentiate from the myriad of existing work on secure logging by assuming that:

- The cloud provider (i.e. the logger) and the cloud customer are malicious: in fact, the cloud provider is interested in tampering with the cloud customer's logs, whereas the cloud customer wants to falsely accuse the cloud provider of tampering in the aim of obtaining a financial compensation.
- There exists an honest judge (auditor) who is in charge of resolving any dispute between the cloud customer and the cloud provider by generating a *proof* of either the cloud provider's compliance or the cloud provider's infractions.

The idea is that whenever the cloud customer wants the cloud provider to execute an operation op_i on its behalf, the cloud customer provides a commitment to op_i to the cloud provider in the form of a signature $\sigma_{\text{customer}}(L_i)$ of the log entry L_i generated by the cloud customer. The log entry L_i includes the details of the operation op_i requested by the cloud customer, namely, the type of the operation op_i , the identifier of the resource on which op_i will be performed and a time-stamp...etc. We note here that the signature $\sigma_{\text{customer}}(L_i)$ can be used later (in the case of a dispute) by the cloud provider to prove to third parties that the cloud customer actually requested the execution of operation op_i . In the same manner, upon the receipt of an operation request, the cloud provider sends an acknowledgment of receipt to the cloud customer. This acknowledgment consists of a cloud provider's signature $\sigma_{\text{cloud}}(\mathcal{L}_i)$, where \mathcal{L}_i is the log entry corresponding to the execution of operation op_i . As with $\sigma_{\text{customer}}(L_i)$, the signature $\sigma_{\text{cloud}}(\mathcal{L}_i)$ can be employed by the cloud customer to resolve a dispute with the cloud provider.

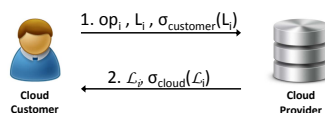


Figure 8: Signature exchange for inescapable logging

However, a simple exchange of signatures between the cloud customer and the cloud provider does not ensure that the cloud provider cannot deny that the cloud customer submitted an op-

eration to be executed. Notably, the cloud provider can always claim that it did not receive the cloud customer request due to network failures. To address this issue, we draw on techniques of fair exchange [ASW97, ASW98, Ate99, ASW00] that guarantee i.) that the cloud provider will not receive the cloud customer signature (i.e. the cloud customer commitment to the operation to be executed) unless it acknowledges the receipt of the operation request, and ii.) that the cloud customer cannot get the cloud provider signature unless it submits its signature. In fact, fair exchange techniques allow two parties to exchange partial signatures such that a party participating in the protocol will be able to decode the partial signature into a full one only if it behaves correctly.

An efficient approach to implement fair exchange of signatures is **Verifiable Encrypted Signatures** which were initially proposed by [Ate99]. As their name implies, verifiable encrypted signatures are a family of signatures that *can be verified by any entity even when encrypted*, as long as *suitable* public-key cryptosystems are used. While such mechanisms are quite efficient, they assume the existence of trusted third party \mathcal{M} whose *all its communication channels are reliable*, and which consequently acts as a *mediator* \mathcal{M} between the entities involved in the fair exchange protocol. Fortunately however, the mediator is called to intervene in the fair exchange protocol only in the case of a network failure or an attack.

As in previous work [Ate99], we assure a fair exchange of signatures between the cloud customer and the cloud provider by having the cloud customer sending its signature $\sigma_{\text{customer}}(L_i)$ encrypted under the public key of the mediator \mathcal{M} . This corresponds to $E_{\mathcal{M}}(\sigma_{\text{customer}}(L_i))$ in Figure 9. The cloud provider in return verifies the encrypted signature $E_{\mathcal{M}}(\sigma_{\text{customer}}(L_i))$ provided by the cloud customer. If the verification succeeds, then the cloud provider computes a signature $\sigma_{\text{cloud}}(L_i)$ of the log entry L_i , encrypts it under the mediator \mathcal{M} 's public key and forwards the resulting ciphertext $E_{\mathcal{M}}(\sigma_{\text{cloud}}(L_i))$ to the cloud customer. Once the cloud customer receives the cloud provider's encrypted signature $E_{\mathcal{M}}(\sigma_{\text{cloud}}(L_i))$, it checks its validity and forwards its signature $\sigma_{\text{customer}}(L_i)$ in cleartext to the cloud provider. The cloud provider accordingly acknowledges the receipt of the cloud customer's signature $\sigma_{\text{customer}}(L_i)$ by sending its signature $\sigma_{\text{cloud}}(L_i)$ also in cleartext, as illustrated in Figure 9.

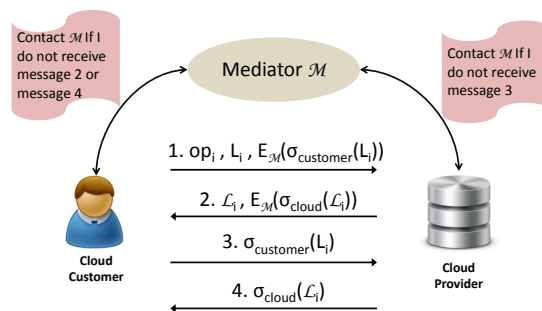


Figure 9: Fair exchange of signatures for inescapable logging

Before demonstrating that the protocol described above ensures fair exchange, we first recall that fair exchange based on verifiable encrypted signatures assumes that i.) the communication channel between \mathcal{M} and the cloud provider and the communication channel between \mathcal{M} and

the cloud customer are reliable, and as a result, ii.) the trusted third party \mathcal{M} can be involved at any stage of the protocol to warrant that the fair exchange of signatures is executed as expected.

Now if the cloud provider does not send its encrypted signature $\sigma_{\text{provider}}(\mathcal{L}_i)$, then the cloud customer contacts \mathcal{M} which makes sure that the cloud customer's request is delivered to the cloud provider. Also, if the cloud customer (the cloud provider respectively) does not send its signature $\sigma_{\text{customer}}(\mathcal{L}_i)$ in cleartext in the third round ($\sigma_{\text{provider}}(\mathcal{L}_i)$ in the fourth round resp.) of the protocol, then the cloud provider (the cloud customer resp.) asks the mediator \mathcal{M} to decrypt the encrypted signature $E_{\mathcal{M}}(\sigma_{\text{customer}}(\mathcal{L}_i))$ ($E_{\mathcal{M}}(\sigma_{\text{provider}}(\mathcal{L}_i))$ resp.). Thus, by having access to the verifiable encrypted signature $E_{\mathcal{M}}(\sigma_{\text{customer}}(\mathcal{L}_i))$ ($E_{\mathcal{M}}(\sigma_{\text{provider}}(\mathcal{L}_i))$ resp.) and with the help of the mediator \mathcal{M} , the cloud provider (the cloud customer resp.) can always derive $\sigma_{\text{customer}}(\mathcal{L}_i)$ ($\sigma_{\text{provider}}(\mathcal{L}_i)$ resp.) in cleartext.

It is important here to note that the trusted third party does not actively engage in the protocol execution, rather it either submits the cloud customer operation request to the cloud provider in case the channel between the cloud provider and the cloud customer fails or it decrypts a verifiable encrypted signature.

A shortcoming of the solution described in Figure 9 is that the cloud customer is required to store the log entries of all the operations it has requested and the corresponding cloud provider's signatures. This may be proved to be very impractical for cloud customers which in most cases only have access to low capacity devices such as laptops or smart-phones. To optimize the storage complexity at the cloud customer, we suggest that instead of keeping the log and the signatures of all the operations it has submitted, the cloud customer stores a hash-chain $H_i = h(\mathcal{L}_i || h(\mathcal{L}_{i-1} || \dots || h(\mathcal{L}_0)))$ of the log entries of the requested operations and the corresponding cloud provider's signature as depicted in Figure 10.

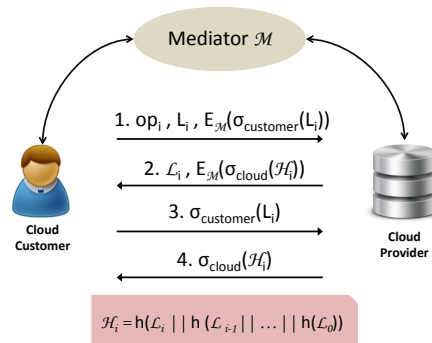


Figure 10: Inescapable logging with hash-chains

Now in the case of a dispute all the cloud customer and the cloud provider need to do is to supply the judge with the signatures of their counterparts.

4.4. Summary

In this chapter, we introduced A-PPLE which is the cornerstone of the accountability enforcement framework in the A4Cloud project. We particularly described the handlers composing the engine and the interactions that govern the engine operations. Besides the description of A-PPLE, the chapter took a look into other methods that may be used in conjunction with A-PPLE to fulfill the accountability obligations identified in C2, and proposed a number of solutions that can be employed to offer better assurance to cloud customers of the compliance of CSPs, and therefore foster trust between different entities in cloud ecosystems.

Conclusion

This deliverable describes the entire A4Cloud policy framework by first providing some refinements of the two early proposed policy languages, namely, AAL and A-PPL. Since AAL defines a light grammar and is human-readable, data subjects can easily define their preferences. In order to partially match these user preferences with the policy rules defined by the data controller, the deliverable also proposes an accountable component design which allows the verification that all defined AAL clauses are well-formed, well-connected and ready to be translated into A-PPL. This component will be implemented within the AccLab tool as part of work package D-3. Once AAL clauses are finalized, the deliverable describes a set of rules to transform these clauses into A-PPL policy rules. This policy representation framework is further demonstrated by mapping some obligations (legal or contractual) defined in work package B-3 (use cases) and some data transfer rules derived from the regulation and defined in work package B-5. These obligations which mainly derive from the EU Data Protection Directive 95/46/EC, help a cloud actor to define data handling and data protection policies, mostly. However, because A-PPL is extensible, some further defined obligations can also be addressed with an updated schema in the future.

Lastly, the deliverable focuses on the enforcement of accountability obligations by initially analyzing the white box and black box strategies. The approach taken within A4Cloud is to take advantage of both strategies and design a dedicated policy enforcement tool. The architecture of the proposed A-PPL engine (A-PPLE) developed as part of work package D-3 is summarized together with some suggestions to assure trustworthiness with respect to this engine. The document also suggests some additional enforcement solutions regrouped into two categories following the classification of the A4Cloud framework defined in work package C-2. These proposed techniques can either be directly integrated within A-PPLE or implemented as an independent tool. Preventive solutions mainly enforce data handling policies while detective solutions usually implement logging/monitoring solutions. The logging function being the most critical functionality, the deliverable addresses a particular attention on the security of the logging modules and the integrity of logs.

References

- [ABC⁺13] Monir Azraoui, Karin Bernsmed, Ronan-Alexandre Cherrueau, Rémi Douence, Kaoutar Elkhyaoui, Alexandr Garaga, Hervé Grall, Refik Molva, Melek Önen, Jean-Claude Royer, Anderson Santana de Oliveira, Mohamed Sellami, Jakub Sendor, and Mario Südholt. Policy representation framework. Technical Report D34.1, Accountability for Cloud and Future Internet Services - A4Cloud Project, December 2013.
- [ABDCDV⁺09] Claudio A. Ardagna, Laurent Bussard, Sabrina De Capitani Di Vimercati, Gregory Neven, Stefano Paraboschi, Eros Pedrini, Stefan Preiss, Dave Raggett, Pierangela Samarati, Slim Trabelsi, and Mario Verdicchio. Primelife policy language. <http://www.w3.org/2009/policy-ws/papers/Trabelisi.pdf>, 2009.
- [Ame89] Pierre America. A Behavioral Approach to Subtyping in Object-Oriented Programming languages. *Proceedings of the workshop on inheritance hierarchies in knowledge representation and programming languages*, pages 141–156, 1989. Viareggio Italy, february 6-8.
- [Aßm03] Uwe Aßmann. *Invasive software composition*. Springer, 2003.
- [ASW97] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM Conference on Computer and Communications Security, CCS '97*, pages 7–17, New York, NY, USA, 1997. ACM.
- [ASW98] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*, pages 86–99, May 1998.
- [ASW00] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *Selected Areas in Communications, IEEE Journal on*, 18(4):593–610, April 2000.
- [Ate99] Giuseppe Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *Proceedings of the 6th ACM Conference on Computer and Communications Security, CCS '99*, pages 138–146, New York, NY, USA, 1999. ACM.
- [BCEPM08] Robert Bunge, Sam Chung, Barbara Endicott-Popovsky, and Don McLane. An operational framework for service oriented architecture network security. In *HICSS*, page 312. IEEE Computer Society, 2008.
- [BCLM13] Denis Butin, Marcos Chicote, and Daniel Le Métayer. Log design for accountability. In *Security and Privacy Workshops (SPW), 2013 IEEE*, pages 1–7. IEEE, 2013.

- [BCM14] Denis Butin, Marcos Chicote, and Daniel Le Métayer. Strong Accountability: Beyond Vague Promises. In *Reloading Data Protection: Multidisciplinary Insights and Contemporary Challenges*, pages 343–369. 2014.
- [BFO⁺13] Karin Bernsmed, Massimo Felici, Anderson Santana De Oliveira, Jakub Sendor, Nils Brede Moe, Thomas Rbsamen, Vasilis Tountopoulos, and Bushra Hasnain. Use case descriptions. Deliverable, Cloud Accountability (A4Cloud) Project, 2013.
- [BFO⁺14] Karin Bernsmed, Massimo Felici, Anderson Santana De Oliveira, Jakub Sendor, Thomas Rbsamen, Vasilis Tountopoulos, Mohamed Sellami, and Jean-Claude Royer. Consolidated use case report. Deliverable, Cloud Accountability (A4Cloud) Project, 2014.
- [BGR⁺14] Walid Benghabrit, Hervé Grall, Jean-Claude Royer, Mohamed Sellami, Karin Bernsmed, and Anderson Santana De Oliveira. Abstract Accountability Language. In *IFIPTM - 8th IFIP WG 11.11 International Conference on Trust Management*, Singapore, Singapour, July 2014.
- [BGRS14] Walid Benghabrit, Hervé Grall, Jean-Claude Royer, and Mohamed Sellami. Accountability for Abstract Component Design. In *EUROMICRO DSD/SEAA 2014*, Verona, Italie, August 2014.
- [BM99] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael Wiener, editor, *Advances in Cryptology CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 431–448. Springer Berlin Heidelberg, 1999.
- [BMB10] Moritz Y Becker, Alexander Malkis, and Laurent Bussard. S4p: A generic language for specifying privacy preferences and policies. *Microsoft Research*, 2010.
- [BPSM⁺97] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml). *World Wide Web Journal*, 2(4):27–66, 1997.
- [BY97] Mihir Bellare and Bennet S. Yee. Forward integrity for secure audit logs. Technical report, 1997.
- [CCD⁺05] J.G. Cederquist, R. Corin, M.A.C. Dekker, S. Etalle, and J.I. Den Hartog. An audit logic for accountability. In *Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on*, pages 34–43. Ieee, 2005.
- [CCS13] Ronan-Alexandre Cherrueau, Omar Chebaro, and Mario Südholt. Flexible and expressive aspect-based control over service compositions in the Cloud. In *4th International Workshop on Variability & Composition*, Digital Library, Fukuoka, Japon, March 2013. ACM.

- [CDR⁺13] Ronan-Alexandre Cherrueau, Rmi Douence, Jean-Claude Royer, Mario Sdholt, Anderson Santana de Oliveira, Yves Roudier, and Matteo Dell'Amico. Reference monitors for security and interoperability in oauth 2.0. In *SETOP 2013*, Lecture Notes in Computer Science. Springer, 2013.
- [CGK⁺13] Sjoerd Cranen, Jan Friso Groote, Jeroen J. A. Keiren, Frank P. M. Stappers, Erik P. de Vink, Wieger Wesselink, and Tim A. C. Willemse. An overview of the mCRL2 toolset and its recent advances. In Nir Piterman and Scott A. Smolka, editors, *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2013.
- [CIS10] CISCO. CISCO ACE XML Gateway, <http://www.cisco.com/en/US/products/ps7314/index.html>, 2010.
- [CIT10] CITRIX. CITRIX NetScaler, <http://www.citrix.com/english/ps2/products/product.asp?contentid=21679>, 2010.
- [CPH03] CheunNgen Chong, Zhonghong Peng, and PieterH. Hartel. Secure Audit Logging with Tamper-Resistant Hardware. In Dimitris Gritzalis, Sabrina De Capitani di Vimercati, Pierangela Samarati, and Sokratis Katsikas, editors, *Security and Privacy in the Age of Uncertainty*, volume 122 of *IFIP – The International Federation for Information Processing*, pages 73–84. Springer US, 2003.
- [CS14] Ronan-Alexandre Cherrueau and Mario Südholt. Enforcing Expressive Accountability Policies. In *WETICE - IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Parma, Italie, June 2014.
- [CW09] Scott A. Crosby and Dan S. Wallach. Efficient Data Structures for Tamper-evident Logging. In *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM'09, pages 317–334, Berkeley, CA, USA, 2009. USENIX Association.
- [dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface Automata. In *Proc. of ESEC/FSE'01*, pages 109–120. ACM Press, 2001.
- [DFG⁺14] Brian Dziminski, Massimo Felici, Fredric Gittler, Bushra Hussain, Theo Koulouris, Ronald Leenes, Maartje Niezen, David Nu nez, Alain Pannetrat, Siani Pearson, Jean-Claude Royer, Dimitra Stefanatou, and Vasilis Tountopoulos. Ms:c-2.3. Technical Report MS:C-2.3, Conceptual Framework, March 2014.
- [dOCSS14] Anderson Santana de Oliveira, Anis Charfi, Benjamin Schmeling, and Gabriel Serme. A model-driven approach for accountability in business processes. In Selmin Nurcan, Pnina Soffer, Rainer Schmidt, and Ilia Bider, editors, *15th Workshop on Business Process Modeling, Development, and Support*, Lecture Notes in Business Information Processing. Springer, 2014.

- [DSI⁺13] Matteo Dell’Amico, Gabriel Serme, Muhammad Sabir Idrees, Anderson Santana de Oliveira, and Yves Roudier. Hipolds: A hierarchical security policy language for distributed systems. *Inf. Sec. Techn. Report*, 17(3):81–92, 2013.
- [FHnOP14] Simone Fischer-Hübner, David Nuñez, Melek Önen, and Tobias Pulls. Privacy Design Guidelines for Accountability Tools. Deliverable, Cloud Accountability (A4Cloud) Project, 2014.
- [Fis08] Michael Fisher. Temporal representation and reasoning. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, pages 513–550. Elsevier, Amsterdam, 2008.
- [GL06] Nils Gruschka and Norbert Luttenberger. Protecting web services from dos attacks by soap message validation. In Simone Fischer-Hübner, Kai Rannenberg, Louise Yngström, and Stefan Lindskog, editors, *SEC*, volume 201 of *IFIP*, pages 171–182. Springer, 2006.
- [HGKW13] Martin Henze, Marcel Großfengels, Maik Koprowski, and Klaus Wehrle. Towards data handling requirements-aware cloud computing. In *2013 IEEE International Conference on Cloud Computing Technology and Science (Cloud-Com)*, 2013.
- [ISR⁺11] Muhammad Sabir Idrees, Gabriel Serme, Yves Roudier, Anderson Santana de Oliveira, Hervé Grall, and Mario Südholt. Evolving security requirements in multi-layered service-oriented-architectures. In Joaquín García-Alfaro, Guillermo Navarro-Arribas, Nora Cuppens-Boulahia, and Sabrina De Capitani di Vimercati, editors, *DPM/SETOP*, volume 7122 of *Lecture Notes in Computer Science*, pages 190–205. Springer, 2011.
- [LdOS13] Volkmar Lotz, Anderson Santana de Oliveira, and Jakub Sendor. Control as a means towards accountable services in the cloud. *International Journal of Computer Systems Science and Engineering - Special issue: 2nd International Symposium on Secure Virtual Infrastructures*, 28(6), 2013.
- [LP00] Gary T. Leavens and Don Pigozzi. A complete algebraic characterization of behavioral subtyping. *Acta Informatica*, 36:617–663, 2000.
- [LW94] Barbara Liskov and Jeannette Wing. A Behavioral Notion of Subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, 1994.
- [MT09] Di Ma and Gene Tsudik. A new approach to secure logging. *Trans. Storage*, 5(1):2:1–2:21, March 2009.
- [Nie95] Oscar Nierstrasz. *Regular Types for Active Objects*, chapter 4, pages 99–121. PH, 1995. Revised and corrected version of a previously published paper.

- [OAS] OASIS Standard. XACML - eXtensible access control markup language. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.
- [Pat07] Don Patterson. Xml firewall architecture and best practices for configuration and auditing. Technical Report 2007, SANS Institute, 2007.
- [Pri11] PrimeLife WP 5.2. Draft 2nd design for policy languages and protocols (heartbeat h5.2.2). Deliverable, The PrimeLife project, 2011.
- [PSSW09] A. Pretschner, F. Schütz, C. Schaefer, and T. Walter. Policy evolution in distributed usage control. *Electron. Notes Theor. Comput. Sci.*, 244:109–123, August 2009.
- [SC00] João Costa Seco and Luís Caires. A basic model of typed components. In Elisa Bertino, editor, *ECOOOP 2000 - Object-Oriented Programming, 14th European Conference, Sophia Antipolis and Cannes, France, June 12-16, 2000, Proceedings*, volume 1850 of *Lecture Notes in Computer Science*, pages 108–128. Springer, 2000.
- [SCM11] Benjamin Schmeling, Anis Charfi, and Mira Mezini. Composing Non-Functional Concerns in Composite Web Services. In *IEEE International Conference on Web Services (ICWS'11)*, Washington DC, USA, 2011. IEEE Computer Society.
- [SK98] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7, SSYM'98*, pages 4–4, Berkeley, CA, USA, 1998. USENIX Association.
- [SWS07] A. Singhal, T. Winograd, and K. Scarfone. Guide to secure web services. Technical Report 2007, NIST, 2007.
- [TNR11] Slim Trabelsi, Gregory Neven, and Dave Raggett. Report on design and implementation. http://primelife.ercim.eu/images/stories/deliverables/d5.3.4-report_on_design_and_implementation-public.pdf, 2011.
- [YSSdO12] Peng Yu, Jakub Sendor, Gabriel Serme, and Anderson Santana de Oliveira. Automating privacy enforcement in cloud platforms. In Roberto Di Pietro, Javier Herranz, Ernesto Damiani, and Radu State, editors, *DPM/SETOP*, volume 7731 of *Lecture Notes in Computer Science*, pages 160–173. Springer, 2012.

A. List of Triggers and Actions in A-PPL

Table 2 presents all the Triggers that can be used in an A-PPL policy.

Table 3 lists the actions available in A-PPL.

B. From AAL to A-PPL Examples

B.1. Rule 1: AAL expression with a PERMIT authorization

- **Description:**

The user *admin* is allowed to use the *read* service provided by the resource *database*.

- **AAL expression:**

PERMIT admin.read[database] ()

- **A-PPL `< rule >` element:**

```
<Rule RuleId="rule1" Effect="Permit">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:
          function:string-equal">
          <AttributeValue DataType="http://www.w3.org
            /2001/XMLSchema#string">admin</AttributeValue>
          <SubjectAttributeDesignator AttributeId="urn:
            oasis:names:tc:xacml:1.0:subject:subject-id"
            DataType="http://www.w3.org/2001/XMLSchema#
              string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml
          :1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org
            /2001/XMLSchema#string">database</
              AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:
            oasis:names:tc:xacml:1.0:resource:resource-id
              " DataType="http://www.w3.org/2001/XMLSchema#
                string"/>
          </ResourceMatch>
        </Resource>
      </Resources>
    <Actions>
      <Action>
```

```

        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:
            function:string-equal">
            <AttributeValue DataType="http://www.w3.org
                /2001/XMLSchema#string">read</AttributeValue>
            <ActionAttributeDesignator AttributeId="urn:
                oasis:names:tc:xacml:1.0:action:action-id"
                DataType="http://www.w3.org/2001/XMLSchema#
                string"/>
        </ActionMatch>
    </Action>
</Actions>
</Target>
</Rule>

```

B.2. Rule 2: AAL expression with a DENY authorization

- **Description:**

The user *admin* is not allowed to use the *delete* service provided by the resource *database*.

- **AAL expression:**

DENY admin.delete[database] ()

- **A-PPL *< rule >* element:**

```

<Rule RuleId="rule1" Effect="Deny">
    <Target>
        <Subjects>
            <Subject>
                <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:
                    function:string-equal">
                    <AttributeValue DataType="http://www.w3.org
                        /2001/XMLSchema#string">admin</AttributeValue>
                    <SubjectAttributeDesignator AttributeId="urn:
                        oasis:names:tc:xacml:1.0:subject:subject-id"
                        DataType="http://www.w3.org/2001/XMLSchema#
                        string"/>
                </SubjectMatch>
            </Subject>
        </Subjects>
        <Resources>
            <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:
                    1.0:function:string-equal">
                    <AttributeValue DataType="http://www.w3.org
                        /2001/XMLSchema#string">database</
                        AttributeValue>
                </ResourceMatch>
            </Resource>
        </Resources>
    </Target>
</Rule>

```

```

        <ResourceAttributeDesignator AttributeId="urn:
            oasis:names:tc:xacml:1.0:resource:resource-id
            " DataType="http://www.w3.org/2001/XMLSchema#
            string"/>
    </ResourceMatch>
</Resource>
</Resources>
<Actions>
    <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:
            function:string-equal">
            <AttributeValue DataType="http://www.w3.org
                /2001/XMLSchema#string">delete</
                AttributeValue>
            <ActionAttributeDesignator AttributeId="urn:
                oasis:names:tc:xacml:1.0:action:action-id"
                DataType="http://www.w3.org/2001/XMLSchema#
                string"/>
        </ActionMatch>
    </Action>
</Actions>
</Target>
</Rule>

```

B.3. Rule 4: a conjunctive set of AAL expressions (AND)

- **Description:**

The user *admin* is allowed to use the *read* **AND** the *write* services provided by the resource *database*.

- **AAL expression:**

PERMIT admin.read[database]() AND PERMIT admin.write[database]()

- **A-PPL < policy >:**

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
    xmlns:ns2="http://www.primelife.eu/ppl/credential" xmlns:ns3="
    http://www.primelife.eu/ppl" xmlns:ns4="http://www.primelife.eu/
    ppl/obligation" PolicyId="policy1" RuleCombiningAlgId="urn:oasis:
    names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
    <ns3:Rule RuleId="rule1" Effect="Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId="urn:oasis:names:tc:xacml
                        :1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org
                            /2001/XMLSchema#string">admin</
                            AttributeValue>
                    </SubjectMatch>
                </Subject>
            </Subjects>
        </Target>
    </ns3:Rule>
</ns3:Policy>

```

```

        <SubjectAttributeDesignator AttributeId="urn
            :oasis:names:tc:xacml:1.0:subject:subject
            -id" DataType="http://www.w3.org/2001/
            XMLSchema#string"/>
    </SubjectMatch>
</Subject>
</Subjects>
<Resources>
    <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml
            :1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org
                /2001/XMLSchema#string">database</
                AttributeValue>
            <ResourceAttributeDesignator AttributeId="
                urn:oasis:names:tc:xacml:1.0:resource:
                resource-id" DataType="http://www.w3.org
                /2001/XMLSchema#string"/>
        </ResourceMatch>
    </Resource>
</Resources>
<Actions>
    <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml
            :1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org
                /2001/XMLSchema#string">read</
                AttributeValue>
            <ActionAttributeDesignator AttributeId="urn:
                oasis:names:tc:xacml:1.0:action:action-id
                " DataType="http://www.w3.org/2001/
                XMLSchema#string"/>
        </ActionMatch>
    </Action>
</Actions>
</Target>
</ns3:Rule>
<ns3:Rule RuleId="rule2" Effect="Permit">
    <Target>
        <Subjects>
            <Subject>
                <SubjectMatch MatchId="urn:oasis:names:tc:xacml
                    :1.0:function:string-equal">
                    <AttributeValue DataType="http://www.w3.org
                        /2001/XMLSchema#string">admin</
                        AttributeValue>
                    <SubjectAttributeDesignator AttributeId="urn
                        :oasis:names:tc:xacml:1.0:subject:subject
                        -id" DataType="http://www.w3.org/2001/
                        XMLSchema#string"/>
                </SubjectMatch>
            </Subject>
        </Subjects>
    </Resources>

```

```

    <Resource>
      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">database</AttributeValue>
        <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </ResourceMatch>
    </Resource>
  </Resources>
  <Actions>
    <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </ActionMatch>
    </Action>
  </Actions>
</Target>
</ns3:Rule>
<ns3:Rule RuleId="rule3">
  <Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-set-equals">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
        </Apply>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
      </Apply>
    </Apply>
  </Condition>
</ns3:Rule>
</ns3:Policy>

```

B.4. Rule 5: a disjunctive set of AAL expressions (OR)

- **Description:**

The user *admin* is allowed to use the *read* **OR** the *write* services provided by the resource *database*.

- **AAL expression:**

PERMIT admin.read[database]() OR PERMIT admin.write[database]()

- **A-PPL < policy >:**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
  xmlns:ns2="http://www.primelife.eu/ppl/credential" xmlns:ns3="
  http://www.primelife.eu/ppl" xmlns:ns4="http://www.primelife.eu/
  ppl/obligation" PolicyId="policy1" RuleCombiningAlgId="urn:oasis:
  names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
  <ns3:Rule RuleId="rule1" Effect="Permit">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml
            :1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org
              /2001/XMLSchema#string">admin</
              AttributeValue>
            <SubjectAttributeDesignator AttributeId="urn
              :oasis:names:tc:xacml:1.0:subject:subject
              -id" DataType="http://www.w3.org/2001/
              XMLElement#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
        <Resource>
          <ResourceMatch MatchId="urn:oasis:names:tc:xacml
            :1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org
              /2001/XMLSchema#string">database</
              AttributeValue>
            <ResourceAttributeDesignator AttributeId="
              urn:oasis:names:tc:xacml:1.0:resource:
              resource-id" DataType="http://www.w3.org
              /2001/XMLSchema#string"/>
          </ResourceMatch>
        </Resource>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch MatchId="urn:oasis:names:tc:xacml
            :1.0:function:string-equal">
```

```

        <AttributeValue DataType="http://www.w3.org
            /2001/XMLSchema#string">read</
            AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:
            oasis:names:tc:xacml:1.0:action:action-id
            " DataType="http://www.w3.org/2001/
            XMLElement#string"/>
    </ActionMatch>
</Action>
</Actions>
</Target>
</ns3:Rule>
<ns3:Rule RuleId="rule2" Effect="Permit">
    <Target>
        <Subjects>
            <Subject>
                <SubjectMatch MatchId="urn:oasis:names:tc:xacml
                    :1.0:function:string-equal">
                    <AttributeValue DataType="http://www.w3.org
                        /2001/XMLSchema#string">admin</
                        AttributeValue>
                    <SubjectAttributeDesignator AttributeId="urn
                        :oasis:names:tc:xacml:1.0:subject:subject
                        -id" DataType="http://www.w3.org/2001/
                        XMLElement#string"/>
                </SubjectMatch>
            </Subject>
        </Subjects>
        <Resources>
            <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml
                    :1.0:function:string-equal">
                    <AttributeValue DataType="http://www.w3.org
                        /2001/XMLSchema#string">database</
                        AttributeValue>
                    <ResourceAttributeDesignator AttributeId="
                        urn:oasis:names:tc:xacml:1.0:resource:
                        resource-id" DataType="http://www.w3.org
                        /2001/XMLSchema#string"/>
                </ResourceMatch>
            </Resource>
        </Resources>
    </Target>
    <Actions>
        <Action>
            <ActionMatch MatchId="urn:oasis:names:tc:xacml
                :1.0:function:string-equal">
                <AttributeValue DataType="http://www.w3.org
                    /2001/XMLSchema#string">write</
                    AttributeValue>
                <ActionAttributeDesignator AttributeId="urn:
                    oasis:names:tc:xacml:1.0:action:action-id
                    " DataType="http://www.w3.org/2001/
                    XMLElement#string"/>
            </ActionMatch>
        </Action>
    </Actions>
</ns3:Rule>

```

```

        </Action>
      </Actions>
    </Target>
  </ns3:Rule>
  <ns3:Rule RuleId="rule3">
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
        not">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
          string-subset">
            <ActionAttributeDesignator AttributeId="urn:oasis:
              names:tc:xacml:1.0:action:action-id" DataType="
                http://www.w3.org/2001/XMLSchema#string"
                MustBePresent="true"/>
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
              function:string-bag">
              <AttributeValue DataType="http://www.w3.org/2001/
                XMLSchema#string">read</AttributeValue>
              <AttributeValue DataType="http://www.w3.org/2001/
                XMLSchema#string">write</AttributeValue>
            </Apply>
          </Apply>
        </Apply>
      </Condition>
    </ns3:Rule>
  </ns3:Policy>

```

C. A-PPL Examples from the Use Cases

We list here some examples of A-PPL statements that are mentioned in section 3.1.4.

Listing 11 is an example of a data handling policy proposed by the hospital about the processing of the collected personal data.

Listing 11: Example of the hospital's policy

```

1 <Policy>
2   <Rule Effect="Permit" RuleId="DS_access">
3     <Target>
4       <Subject>
5         <SubjectMatch MatchId="function:string-equal">
6           <AttributeValue DataType="string">Data Subject</AttributeValue>
7           <SubjectAttributeDesignator DataType="string"
8             AttributeId="subject:role-id"/>
9         </SubjectMatch>
10      </Subject>
11    <Resources><AnyResource/></Resources>
12    <Actions>
13      <Action>
14        <ActionMatch MatchId="string-equal">
15          <AttributeValue DataType="string">read</AttributeValue>

```

```

16      <ActionAttributeDesignator DataType="string"
17          AttributeId="action-id"/>
18    </ActionMatch>
19  </Action>
20  <Action>
21    <ActionMatch MatchId="string-equal">
22      <AttributeValue DataType="string">write</AttributeValue>
23      <ActionAttributeDesignator DataType="string"
24          AttributeId="action-id"/>
25    </ActionMatch>
26  </Action>
27    <Action>
28      <ActionMatch MatchId="string-equal">
29        <AttributeValue DataType="string">delete</AttributeValue>
30        <ActionAttributeDesignator DataType="string"
31            AttributeId="action-id"/>
32      </ActionMatch>
33    </Action>
34  </Actions>
35 </Target>
36 </Rule>
37 <DataHandlingPolicy>
38   <ObligationsSet>
39     <Obligation>
40       <TriggerPersonalDataAccessed>
41       </TriggerPersonalDataAccessed>
42       <ActionNotify>
43         <Media>e-mail</Media>
44         <Address>data.subject@example.com</Address>
45         <Recipient>Data Subject</Recipient>
46         <Type>Data Handling Policy</Type>
47       </ActionNotify>
48     </Obligation>
49     <Obligation>
50       <TriggerPersonalDataDeleted>
51       </TriggerPersonalDataDeleted>
52       <ActionLog>
53         <Timestamp>true</Timestamp>
54         <Action>true</Action>
55         <Purpose>false</Purpose>
56         <Subject>true</Subject>
57         <Resource>true</Resource>
58         <Location>false</Location>
59         <Expiration>false</Expiration>
60         <Flag>true</Flag>
61       </ActionLog>
62     </Obligation>
63   </ObligationsSet>
64   <TriggerAtTime>
65     <Start><StartNow/></Start>
66     <MaxDelay>
67       <Duration>2Y</Duration>
68     </MaxDelay>
69   </TriggerAtTime>

```

```

70     <ActionDeletePersonalData/>
71   </Obligation>
72 </ObligationsSet>
73 <AuthorizationsSet>
74   <AuthzUseForPurpose>
75     <Purpose duration=2Y region=Europe>admin</Purpose>
76     <Purpose duration=2Y region=Europe>research</Purpose>
77     <Purpose duration=30D region=Europe>surveys</Purpose>
78   </AuthzUseForPurpose>
79   <AuthzDownstreamUsage allowed="true">
80     <Policy>
81       <!-- Policy for sub-providers -->
82     </Policy>
83   </AuthzDownstreamUsage>
84 </AuthorizationsSet>
85 </DataHandlingPolicy>
86 </Policy>

```

Listing 12 shows an example of how an A-PPL policy can specify the notification about the processing policy.

Listing 12: Obligation 1: Informing about processing

```

<!-- Notify the data subject about the processing policy -->
<Obligation>
  <TriggerOnDataCollection>
    <MaxDelay>
      <!-- Notify within 2 minutes -->
      <Duration>0Y0M0DT0H2M0S</Duration>
    </MaxDelay>
  </TriggerOnDataCollection>
  <ActionNotify>
    <Media>e-mail</Media>
    <Address>data.subject@example.com</Address>
    <Recipient>Data Subject</Recipient>
    <Type>Data Handling Policy</Type>
  </ActionNotify>
</Obligation>

```

Listing 13 shows how ActionRequestConsent can be used to request the consent from the data subject.

Listing 13: Obligation 13-15: Consent to processing

```

<Obligation>
  <TriggerAtTime>
    <Start><StartNow/></Start>
  </TriggerAtTime>
  <ActionRequestConsent/>
</Obligation>

```

Listing 14 shows how ActionNotify can be used to inform about the use of sub-processors.

Listing 14: Obligation 17: Informing about the use of sub-processors

```
<Obligation>
  <TriggerPersonalDataSent>
    <Id>Personal data of patient</Id>
    <MaxDelay>
      <!-- Notify within 2 minutes -->
      <Duration>2M</Duration>
    </MaxDelay>
  </TriggerPersonalDataSent>
  <ActionNotify>
    <Media>e-mail</Media>
    <Address>hospital@example.com</Address>
    <Recipient>Hospital</Recipient>
    <Type>Usage of processing</Type>
  </ActionNotify>
</Obligation>
```

Specifying data location can be done as described in Listing 15.

Listing 15: Obligation 21: Data location

```
<AuthzUseForPurpose>
  <Purpose duration=2Y region=Europe>admin</Purpose>
  <Purpose duration=2Y region=Europe>research</Purpose>
</AuthzUseForPurpose>
```

Listing 16 presents an example of security breach (data loss) notified to the data subject.

Listing 16: Obligation 18: Security breach notification (data loss)

```
<Obligation>
  <TriggerDataLost>
    <MaxDelay>
      <!-- Notify within 2 minutes -->
      <Duration>0Y0MODT0H2MOS</Duration>
    </MaxDelay>
  </TriggerDataLost>
  <ActionNotify>
    <Media>e-mail</Media>
    <Address>data.subject@example.com</Address>
    <Recipient>Data Subject</Recipient>
    <Type>Data Lost</Type>
  </ActionNotify>
</Obligation>
```

An example of an A-PPL rule for evidence of data deletion is described in Listing 17.

Listing 17: Obligation 19-20: Evidence of data deletion

```
<Obligation>
  <TriggerOnEvidenceRequestReceived>
  </TriggerOnEvidenceRequestReceived>
  <ActionEvidenceCollection>
    <Evidence>
      <Attribute AttributeId="evidence-type" DataType="string">
        <AttributeValue>Logs of deletion</AttributeValue>
      </Attribute>
    </Evidence>
  </ActionEvidenceCollection>
</Obligation>
```

```
</Evidence>
<Resource>
  <Attribute AttributeId="resource-id" DataType="string">
    <AttributeValue>Personal Data</AttributeValue>
  </Attribute>
</Resource>
<Subject>
  <Attribute AttributeId="subject-id" DataType="string">
    <AttributeValue>Data subject</AttributeValue>
  </Attribute>
</Subject>
<Recipient>
  <Attribute AttributeId="recipient-id" DataType="string">
    <AttributeValue>Hospital</AttributeValue>
  </Attribute>
</Recipient>
</ActionEvidenceCollection>
</Obligation>
```

Informing about policy violations can be expressed within an A-PPL policy as shown in Listing 18.

Listing 18: Obligation 29: Informing about policy violations

```
<Obligation>
  <TriggerOnViolation>
    <MaxDelay>
      <!-- Notify within 2 minutes -->
      <Duration>0Y0M0DT0H2M0S</Duration>
    </MaxDelay>
  </TriggerOnViolation>
  <ActionNotify>
    <Media>e-mail</Media>
    <Address>data.subject@example.com</Address>
    <Recipient>Data Subject</Recipient>
    <Type>Policy violation</Type>
  </ActionNotify>
</Obligation>
```

D. Model for Data Transfer Regulation

B-4 partners, and especially QMUL, propose a model for data transfer rules. The outcome of their work is reproduced below.

D.1. Data Transfer Rules

The model for data transfer regulation stems from articles 40, 41, 42 and 44 of the Proposed Regulation (Parliament LIBE text¹⁶).

¹⁶<http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52012PC0011>

Rule 1 (general rule)

IF

Transferee is to a receiving country outside the EEA

AND

None of rules 2, 3 or 4 is satisfied

THEN

Transfer is prohibited

Notes to rule 1:

See discussion of 'receiving country' in notes to rule 2 for meaning of 'outside the EEA'.

There are rules about the transfer of personal data to international organisations which we have omitted because they are not the normal case, and would add substantial complexity. A decision about whether to include them can be made later.

Rule 2 (transfers where there is an objective test to allow transfer)

IF

a. Commission decision that receiving country ensures adequate level of protection

OR

b. Commission decision that relevant processing sector within receiving country ensures adequate level of protection

OR

c. Both transferor and transferee possesses a valid European Data Protection Seal

OR

d. Transfer is in accordance with standard data protection contractual clauses adopted by transferor's DPA and included in the contract between transferor and transferee

OR

e. Transfer is in accordance with binding corporate rules approved by transferor's DPA

OR

f. Transfer is in accordance with contractual clauses individually approved by transferor's DPA and agreed with transferee

THEN

Transfer is permitted

Notes to rule 2:

(parts a & b) The text of the Proposed regulation does not make it clear whether the 'receiving country' is the geographical location of the server to which data is sent, or the place of establishment (registered company office) of the legal entity which is to receive the data. It might be sensible to code on the basis that it is both! (we know that in computing terms things are never A and B).

(parts a & b) These decisions will be published in the Official Journal, and a list of approved countries will be on the Commission website. However, it might be simplest to keep a master list, rather than trying to scrape the website each time the rule is activated, as the website is modified randomly so that the relevant page may move or change format.

(part c) Not clear where this will be published, but probably on the Commission website. Revo-

cation checking might be difficult - is this something the transferor could simply self-certify?
(part d) Not clear where this will be published, but probably on the relevant DPA website. Re-
vocation checking might be difficult - is this something the transferor could simply self-certify?
(parts e & f) These will be individual approvals, and probably not published on the DPA website.
Transferor self-certification may be the only possibility here.
(parts d,e &f) Again, the transferor's self-certification will be the only way to assert that the
relevant contractual clauses or corporate rules are in place.

Rule 3 (transfers where there is a mix of objective and evaluative evidence)

IF

Data subject has consented to transfer

AND

Professional advice that consent meets the law's requirements

THEN

Transfer is permitted

Notes to rule 3:

Professional advice might be taken from an external advisor(lawyer, accountancy firm, etc) or
from an in-house professional (DP compliance officer). It will not be appropriate for the system
to disclose the advice text for reasons of commercial and professional confidentiality, and pos-
sible legal professional privilege. Thus self-certification by the transferor is the only possible
evidence here.

If circumstances change (eg contract terms with any party involved, processing purposes, etc)
then the advice should be updated - this is something the system might flag up, or even enforce
(perhaps by linking the status of the advice [current/outdated] to the state of the processing op-
erations/contracts/etc at the time the advice was given).

Rule 4 (evaluative evidence only)

IF

a. Professional advice that the transfer is necessary for the performance of a contract be-
tween the data subject and the controller or the implementation of pre-contractual measures
taken at the data subject's request

OR

b. Professional advice that the transfer is necessary for the conclusion or performance of a
contract concluded in the interest of the data subject between the controller and another natural
or legal person

OR

c. Professional advice that the transfer is necessary for important grounds of public interest
OR

d. Professional advice that the transfer is necessary for the establishment, exercise or de-
fence of legal claims

OR

e. Professional advice that the transfer is necessary in order to protect the vital interests of
the data subject or of another person, where the data subject is physically or legally incapable

of giving consent
 THEN
 transfer is permitted

Notes to rule 4:

This part of the law requires a judgment on the part of the transferor that the transfer is 'necessary' for a particular reason. The language used here is that of the Proposed Regulation, and is what the professional adviser should consider. We do not think it is possible to break down these elements any further.

It might be possible to substitute a single rule:

IF
 Professional advice that transfer is necessary for a reason set out in Article 44(1)(b)-(f)
 THEN
 transfer is permitted

However, this would tell the data subject nothing about the reason used to justify the transfer, and would thus render the account pretty meaningless. Thus we have broken down the Article as above to achieve greater transparency in the account.

Rule 5 (onward transfers)

IF
 Rule 2, 3 or 4 were applied to permit the original transfer
 AND
 The transferee proposes to transfer the personal data to another receiving country outside the EEA
 AND
 Rule 2, 3 or 4 applies to permit the new transfer
 THEN
 transfer is permitted

D.2. Example of A-PPL rules for data transfer

Table 4 lists and describes the provisional actions we propose in section 3.2.3.

We present hereunder an example of how Rule 2 and 3 can be translated into A-PPL policies.

Listing 19: Translation of some data transfer rules into A-PPL policies

```

1 <DataHandlingPolicy>
2   <AuthorizationsSet>
3     <AuthzDownstreamUsage allowed = "true" >
4       <Condition ConditionId="DataTransferRule">
5         <Apply FunctionId="function:or">
6           <ProvisionalAction ActionId="CheckReceivingCountry">
7             <AttributeValue DataType="url">
8               http://ec.europa.eu/list_of_countries
9             </AttributeValue>
10          </ProvisionalAction>

```

```

11     <ProvisionalAction ActionId="GetValidSeal">
12     </ProvisionalAction>
13     <ProvisionalAction ActionId="CheckContract">
14         <AttributeValue DataType="string">
15             Standard data protection clauses adopted by DPA, included in the
16                 contract between transferor and transferee
17         </AttributeValue>
18         <AttributeValue DataType="string">
19             Self-Certification
20         </AttributeValue>
21     </ProvisionalAction>
22     <ProvisionalAction ActionId="CheckContract">
23         <AttributeValue DataType="string">
24             Binding Corporate Rules
25         </AttributeValue>
26         <AttributeValue DataType="string">
27             Self-Certification
28         </AttributeValue>
29     </ProvisionalAction>
30     <ProvisionalAction ActionId="CheckContract">
31         <AttributeValue DataType="string">
32             Contractual clauses individually approved by DPA, agreed with
33                 transferee
34         </AttributeValue>
35         <AttributeValue DataType="string">
36             Self-Certification
37         </AttributeValue>
38     </ProvisionalAction>
39     <Apply FunctionId="function:and">
40         <ProvisionalAction ActionId="GetConsent">
41         </ProvisionalAction>
42         <ProvisionalAction ActionId="GetProfessionalAdvice">
43             <AttributeValue DataType="string">
44                 Consent meets the requirements of the law
45             </AttributeValue>
46             <AttributeValue DataType="string">
47                 Lawyer
48             </AttributeValue>
49             <AttributeValue DataType="string">
50                 Self-Certification
51             </AttributeValue>
52         </ProvisionalAction>
53     </Apply>
54 </Condition>
55 </AuthzDownstreamUsage>
56 </AuthorizationsSet>
57 </DataHandlingPolicy>

```

Line 3 (terminated on line 54) defines the conditions under which a *downstream usage* is authorized, that is the conditions under which a transfer is permitted.

Line 4 opens a Condition tag (which ends on line 53) to specify the conditions of the transfer.

Line 5 creates a set of disjunctive rules using the function `or`. There are 6 of them:

- Lines 6-10, they correspond to Rules 2.a and 2.b, in which we require the provisional action that verifies that the receiving country is in the list of countries approved by the European Commission.;
- Lines 11-12, they express the Rule 2.c to request a valid data protection seal for the transfer.
- Lines 13-20, they request a verification on whether the transfer is in accordance with contractual clauses as stated in Rule 2.d.
- Lines 21-28, similarly, they request a verification on whether the transfer is in accordance with binding corporate rules as stipulated in Rule 2.e.
- Lines 29-36, they request a verification of the transfer according to Rule 2.f.
- Lines 37-51, they correspond to Rule 3. As this rule is a conjunction of two conditions, Line 37 creates a set of conjunctive of two rules using the function `and`.
 - Lines 38-39, they request the consent from the data subject on the transfer of his personal data as mentioned in Rule 3.a
 - Lines 40-50, they request the professional advice about the consent given by the data subject as stated in Rule 3.b

E. From AAL to A-PPL: the final A-PPL policy

Listing 20: final A-PPL policy

```

1 <Policy PolicyId="PIIPolicy" RuleCombiningAlgId="rule-combining-algorithm
  :permit-overrides">
2
3 <!-- Kim is the owner of the PII -->
4 <Rule Effect="Permit" RuleId="Kim_access">
5   <Target>
6     <Subjects>
7       <Subject>
8         <SubjectMatch MatchId="function:string-equal">
9           <AttributeValue DataType="string">Kim</AttributeValue>
10          <SubjectAttributeDesignator DataType="string" AttributeId="
              subject:subject-id"/>
11        </SubjectMatch>
12      </Subject>
13    </Subjects>
14    <Actions>
15      <Action>
16        <ActionMatch MatchId="function:string-equal">
17          <AttributeValue DataType="string">read</AttributeValue>
18          <ActionAttributeDesignator DataType="string" AttributeId="action:
              action-id"/>

```

```

19     </ActionMatch>
20 </Action>
21 <Action>
22     <ActionMatch MatchId="function:string-equal">
23         <AttributeValue DataType="string">write</AttributeValue>
24         <ActionAttributeDesignator DataType="string" AttributeId="action:
            action-id"/>
25     </ActionMatch>
26 </Action>
27 </Actions>
28 </Target>
29 </Rule>
30
31 <!-- Sandra has only read access -->
32 <Rule Effect="Permit" RuleId="Sandra_access">
33     <Target>
34         <Subjects>
35             <Subject>
36                 <SubjectMatch MatchId="function:string-equal">
37                     <AttributeValue DataType="string">Sandra</AttributeValue>
38                     <SubjectAttributeDesignator          DataType="string" AttributeId="
                        subject:subject-id"/>
39                 </SubjectMatch>
40             </Subject>
41         </Subjects>
42     <Actions>
43         <Action>
44             <ActionMatch MatchId="function:string-equal">
45                 <AttributeValue DataType="string">read</AttributeValue>
46                 <ActionAttributeDesignator DataType="string" AttributeId="action:
                    action-id"/>
47             </ActionMatch>
48         </Action>
49     </Actions>
50 </Target>
51 </Rule>
52
53 <!-- Obligations set-->
54 <DataHandlingPolicy>
55     <ObligationsSet>
56
57         <!-- Notification of Kim on authorized access -->
58         <Obligation>
59             <TriggersSet>
60                 <!-- A-PPL trigger -->
61                 <TriggerPersonalDataAccessPermitted>
62                 </TriggerPersonalDataAccessPermitted>
63             </TriggersSet>
64             <!-- A-PPL action -->
65             <ActionNotify>
66                 <Media>e-mail</Media>
67                 <Address>kim@a4cloud.com</Address>
68                 <Recipients>Kim</Recipients>
69                 <Type>Authorized access</Type>

```

```
70     </ActionNotify>
71 </Obligation>
72
73 <Obligation>
74   <TriggersSet>
75     <!-- A-PPL trigger -->
76     <TriggerPersonalDataAccessPermitted>
77     </TriggerPersonalDataAccessPermitted>
78   </TriggersSet>
79   <!-- A-PPL log action -->
80   <ActionLog>
81     <Timestamp>true</Timestamp>
82     <Action>true</Action>
83     <Purpose>false</Purpose>
84     <Subject>true</Subject>
85     <Resource>true</Resource>
86     <Location>false</Location>
87     <Expiration>false</Expiration>
88     <Flag>true</Flag>
89   </ActionLog>
90 </Obligation>
91
92 <!--Data retention-->
93 <!--Delete data after a specific date-->
94 <Obligation>
95   <TriggersSet>
96     <TriggerAtTime>
97       <Start>
98         <StartNow />
99       </Start>
100     <MaxDelay>
101       <Duration>"2Y"</Duration>
102     </MaxDelay>
103   </TriggerAtTime>
104 </TriggersSet>
105   <!-- A-PPL action -->
106   <ActionDeletePersonalData>
107   </ActionDeletePersonalData>
108 </Obligation>
109
110 </ObligationsSet>
111 </DataHandlingPolicy>
112 </Policy>
```

D:C-4.2 Policy representation and enforcement techniques

Obligations	A-PPL elements for example
1-4: Informing about processing, purposes, recipients and rights.	TriggerOnDataCollection in combination with ActionNotify with data subjects as recipients
5: Data collection purposes.	List of Purposes in the element AuthzUseForPurpose
6: The right to access, correct and delete personal data.	XACML access control rules
7: Data storage period.	ActionDeletePersonalData in a combination with a trigger like TriggerOnDataCollection or TriggerOnTime and specify duration in a <Purpose> element, inside a <AuthzUseForPurpose> environment.
8, 11-12: Security and privacy measures.	XACML rules for confidentiality
9-10: Rules for data processing by providers.	Use of AuthzDownstreamUsage
13-15: Consent to processing.	Use of ActionRequestConsent with a trigger like TriggerOnTime
16: Informing DPAs.	TriggerOnDataCollection in combination with ActionNotify with DPA as recipients
17: Informing about the use of sub-processors.	Use of ActionNotify
18: Security breach notification.	Use of ActionNotify in combination of a trigger such as TriggerDataLost or TriggerOnPolicyViolation
19-20: Evidence on data processing and data deletion.	TriggerOnEvidenceRequestReceived in conjunction with ActionEvidenceCollection
21: Data location.	Specify region in a <Purpose> element, inside a <AuthzUseForPurpose> environment.
22: Informing about personal data processing.	Use of ActionNotify
23: Personal data minimization and 24: Privacy-by-default.	Out of scope
25: Specifying user preferences.	In this A-PPL policy that results from the translation of AAL matching policy
26: Monitoring of data practices.	Logging rules with ActionLog
27-28: Compliance with user preferences and privacy policies.	ActionAudit and ActionEvidenceCollection
29-30: Informing about policy violations and privacy preferences violations.	Use of ActionNotify
31: Remediation in case of incidents.	Use of ActionNotify

Table 1: Summary of A-PPL examples

D:C-4.2 Policy representation and enforcement techniques

Name	Description
TriggerAtTime	Occurs based on a particular defined time
TriggerPeriodic	Occurs repeatedly according to a well-specified period
TriggerPersonalData-AccessedForPurpose	Occurs each time the personal data bound to the obligation is accessed of one of the defined purposes
TriggerPersonalDataDeleted	Occurs when the personal data associated with the obligation is deleted
TriggerPersonalDataSent	Occurs when the personal data akin to the obligation is forwarded to a third-party
TriggerDataLost	Occurs when the personal data is lost
TriggerPolicyViolation	Occurs when a policy violation is detected
TriggerOnDataCollection	Occurs when the data controller collects the data, after receiving consent from the data subject
TriggerPersonalData-AccessPermitted	Occurs when an access to data subject's data is permitted
TriggerPersonalData-AccessDenied	Occurs when an access to data subject's data is denied
TriggerOnPolicyUpdate	Occurs when a policy is updated
TriggerOnPreferencesUpdate	Occurs when the data subject updates its preferences
TriggerOnComplaint	Occurs when a particular entity receives a complaint about the processing of one's personal data
TriggerOnEvidence-RequestReceived	Occurs when an auditee received an evidence request

Table 2: List of Triggers in A-PPL

Name	Description
ActionDeletePersonalData	This action deletes personal data, intended for handling data retention
ActionAnonymizePersonalData	This action anonymizes personal data
ActionNotify	This action notifies the recipient of the notification when triggered
ActionLog	This action logs an event
ActionAudit	This action creates an evidence request
ActionEvidenceCollection	This action collects requested evidence
ActionRequestConsent	This action requests the data subjects for consent of processing their personal data

Table 3: List of Actions in A-PPL

Name	Description	Attributes	
GetConsent	This action requests the consent from the data subject on data transfer of his/her personal data. The consent should be acquired in order to get authorization of transfer.	/	
CheckReceivingCountry	This action verifies whether the receiving country is in the list of countries approved by the European Commission. The authorization of transfer is given if the country is in the list.	List	Link to the list of approved countries.
GetValidDPSeal	This action requests the transferee to show its valid European Data Protection Seal.	/	
CheckContract	This action verifies whether the transfer is in accordance with contractual clauses.	Statement	Defines the statement with which the transfer should comply, see Rule 2 d, e and f. (see Appendix D)
		Type	Defines the type of accordance, "self- certification" in most of cases.
GetProfessionalAdvice	This action requests professional advice for data transfer evaluation.	Type	Defines the type of expected professional advice. This may be "self-certification" in most of cases.
		Advisor	The identity of the advisor.
		Reason	This defines the reason why the data controller should ask for advice (see Rules 3 and 4).

Table 4: List of provisional actions for data transfer rules.