CLOUD
ACCOUNTABILITY
PROJECT

D:C-4.1: Policy Representation Framework

Deliverable Number: D34.1
Work Package: WP 34
Version: Final
Deliverable Lead Organisation: Armines
Dissemination level: PU
Contractual Date of Delivery (release): 30/11/2013
Date of Delivery: 23/12/2013

Editors
Jean-Claude Royer (Armines)

Contributors
Ronan-Alexandre Cherrueau, Rémi Douence, Hervé Grall, Jean-Claude Royer, Mohamed Sellami, Mario Südholt (EMN); Monir Azraoui, Kaoutar Elkhiyaoui, Refik Molva, Melek Önen (EURECOM); Alexandr Garaga, Anderson Santana de Oliveira, Jakub Sendor (SAP); Karin Bernsmed (SINTEF)

Reviewers
Tobias Pulls (Karlstad University), Dimitra Stefanatou (Tilburg University)

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

FP7-ICT-2011-8-317550-A4CLOUD

SEVENTH FRAMEWORK PROGRAMME

Executive Summary

Policies governing the use and handling of personal and business confidential data have to meet a number of requirements that are deriving from accountability obligations. We therefore develop the A4Cloud policy framework that is intended to extend traditional security policies by general means for the definition and the enforcement of accountability obligations. We expect not only to enforce accountability obligations but also to provide assistance in writing, checking and translating these obligations. From our review of the state of the art on policy languages, we identify that the PrimeLife Policy Language (PPL) could be extended to cope with accountability obligations. We therefore propose A-PPL (Accountable PPL) which enriches PPL with new roles and features such as notification, detailed logging and evidence collection as required by different accountability obligations.

On the other hand, since mapping obligations into a machine understandable language such as A-PPL could be tedious and error prone, we propose to define a higher level language: the Abstract Accountability Language (AAL). AAL aims at removing the gap between obligations and policies defined in natural languages and concrete operational languages. Therefore, AAL is defined as being a domain specific language for the declarative definition of obligations in terms of temporal relationships over basic operations that are fundamental to security, privacy and accountability. This deliverable introduces these two main languages and is structured as follows:

- Chapter 1 introduces the requirements that motivate the development of a policy framework for accountability properties and provides an overview of the framework.
- Chapter 2 presents AAL, a brief introduction and a simple grammar. The chapter also describes the semantics based on a temporal logic and a policy calculus. Finally, we show that AAL and its semantics meets the mandatory requirements of an accountability framework introduced in Chapter 1.
- Chapter 3 presents the extended A-PPL language which in addition to traditional security and privacy requirements (mostly covered by the existing PPL) also addresses accountability specific obligations.
- In Chapter 4, in order to validate the proposed framework, we show how accountability obligations are expressed both in AAL and in A-PPL based on the health care use case developed in deliverable DB3.1 [BFO+13].

The A4Cloud policy framework is related to several other conceptual results and software artifacts that are delivered as part of other work packages in A4Cloud. For instance, the proposed two languages and the corresponding enforcement mechanisms will be implemented within the D-3 work package on tools for policy enforcement and compliance. Furthermore, the A4Cloud policy framework is one of the major parts of the A4Cloud accountability toolkit; it is therefore an integral part of the A4Cloud architecture that is under definition as part of WP C-2, has to interoperate with the other A4Cloud tools as defined in WP D-3 and will be validated as part of the project's accountability demonstrator in D-7 and in relation with the use cases derived from WP B-3.

Contents

List of figures

List of tables

Chapter 1

Policy Requirements and Overview of the Framework

In the A4Cloud Glossary [AAB+13], the term *policy* has been defined as "a set of rules related to a particular purpose" [CFH+13b]. Moreover the glossary points out that "A rule can be expressed as an obligation, an authorization, a permission, or a prohibition. Not every policy is a constraint. Some policies represent an empowerment" [CFH+13b]. In this deliverable, we consider *accountability policies* as a set of rules to (i) allow end users or businesses to define how personal and/or confidential data in cloud environments can be processed and (ii) allow cloud service providers to give an account for all operations. Thus, accountability policies will include not only obligations relating to security and privacy, but also obligations relating to the accountability attributes currently identified in WP:C-2 (i.e. assurance, verifiability, observability, etc. see [CFH+13b] for more details).

In the security domain, a policy model is often seen as a technical and abstract implementation of the core security objective of a class of policies. For example, the Bell-La Padula model [BL96] is a model for confidentiality policies and the Biba model [Bib77] is a model for integrity policies. In the context of A4Cloud, we define an *accountability policy model* as a technical and abstract implementation of the core accountability attributes such as responsibility, assurance, transparency and remediation [CFH+13a].

A policy language allows concrete policies to be represented. The new A4Cloud language needs to be able to represent accountability rules for governance of personal data and business confidential information for any kind of cloud service (and composition of such services). In addition, to simplify automated enforcement of accountability policies, the language needs to be represented in a machine-readable format.

In Section 1.1 we introduce the requirements the A4Cloud policy representation framework should address and we further present an overview on the framework defining the two main languages, namely AAL and A-PPL in Section 1.2.

1.1 A4Cloud Policy Requirements

In this section we summarize the requirements that influenced our work during the design of the A4Cloud policy framework. We define these requirements by analyzing the A4Cloud conceptual framework [CFH+13b] and the outcomes of the first stakeholder workshop for elicitation of requirements [MJH+13]. A detailed analysis of the requirements for the accountability policy framework is presented in an internal report [ABE+13], which we sum up in the current section.

We have classified requirements into two main categories: (i) requirements related to data handling: for example, the new language should express user preferences/constraints, usage control rules, and anonymity related options; (ii) specific requirements related to accountability needs, often absent in existing policy models and languages, such as audit, notification, etc. Some requirements were considered as optional since such framework features are not fundamental for ensuring accountability according to the WP C4 participants, but will require considerable research effort. Given that our main objectives are already considerably ambitious and that the effort allocated is limited, optional requirements will receive lower priority for integration in the A4Cloud policy framework.

Requirement 1 (Capturing privacy preferences) *The language must allow to express user preferences about the usage of their data and personal information. In particular, a data subject*[1] *should be able to specify how her personal data is to be processed, to whom it can be disclosed, and under which conditions.*

Requirement 2 (Anonymity/Pseudonymity) *As an optional feature, the accountability policy language of A4Cloud may allow anonymous or pseudonymous access control to personal data. For an anonymous access, a data subject is not identifiable within a set of subjects. The data controller*[2] *cannot identify a given data subject among other subjects. For a pseudonymous access control, pseudonyms are used as identifiers to prevent the data controller from knowing the real identity of a data subject.*

Requirement 3 (Data Minimization) *The policy language may enable the personal data minimization principle as an optional requirement. The principle of data minimization means that a data controller should limit personal data collection the minimum amount of data required to meet the particular purposes for which consent was obtained.*

Requirement 4 (Strong Policy Binding) *Optionally, attaching policies to data may be considered in the design of the A4Cloud policy language, as accountability is independent from such mechanism. A sticky policy, agreed between a data controller and a data subject, is attached to a resource and drives access control and usage control decision as well as privacy enforcement. We use the term resource here in a very broad sense. It can be understood as a computation or storage virtual device in the cloud, but personal data here can be also understood as a resource, whose owner is the data subject. The language may provide mechanisms to identify what the data subject allows regardless of who transmits the data (originator's policy).*

Requirement 5 (Data Retention Periods) *The accountability policy language must be able to express rules about data and metadata retention such as retention time. Notice that some privacy*

[1]Data subject: Identified or identifiable natural person (see EU Directive 95/46/EC [Eur95]) from whom data is collected and/or about whom that data is processed [PMA+09].

[2]Data controller: The natural or legal person, public authority, agency or any other body which alone or jointly with others determines the purposes and means of the processing of personal data; where the purposes and means of processing are determined by national or Community laws or regulations, the controller or the specific criteria for his nomination may be designated by national or Community law [Eur95].

policy languages already have such features such as PPL [ABDCDV$^+$09] whose "generic policies", cover "generic" metadata.

Requirement 6 (Expressing Regulatory Constraints) *The language must be able to express regulatory obligations concerning the data handling and accountability obligations. It must provide constructs to express such regulatory constraints in a machine-readable way.*

Requirement 7 (Access Control rules) *The language should be able to capture access control policies: which users should be granted which type of access (read, write, etc.) to a resource. The subject requesting access may be defined by a set of attributes such as name, pseudonym, role, etc. The language must support role definition according to the cloud accountability conceptual framework. Specific obligations and access rights for data controllers and data processors*[3] *need to be clearly defined in the accountability policy.*

Requirement 8 (Delegation of Rights) *In a distributed context, where applications span security boundaries, delegation is needed in order to generate a chain of trust between the end user and the other entities involved in a service delivery chain. The language must provide delegation capabilities so that the administration and the management of authorization decisions can be decentralized and distributed. For instance, the owner of a resource can delegate parts of their administrative rights on that resource to a third-party. The language must allow to express which rights are delegated, and perhaps which rights can be delegated (this will need to be configured according to the regulatory regime in place).*

Requirement 9 (Auditability) *The policy language must describe the clauses in a way that actions taken upon enforcing the policy can be audited in order to ensure that the policy was adhered to. The policy language must support ways to define what types of operations (such as access, deletion, delegation, modification of a resource) need to be audited and for which purposes - some languages in our review of the state of the art (see Appendix A) have primitive logging mechanisms that need to be extended for auditability.*

Requirement 10 (Controlling Policy Disclosure) *An accountability policy (containing access, usage, and privacy aspects) may reveal sensitive information about the data itself it is related to. It is important to understand under which circumstances an accountability policy may allow disclosure of personal data or other business sensitive information without increasing risks. Therefore we consider as an optional feature of the policy language, rules indicating to whom and for which purposes the access/usage control policy should be revealed in cloud scenarios.*

Requirement 11 (Reporting and notifications) *The policy language must be able to describe notifications to be sent to data subjects and third parties in case of policy violation or security incidents, for instance unauthorized access to personal data.*

Requirement 12 (Redress policies) *The policy language must be able to declare recommendations for redress, in order to handle failures to act properly with respect to the accountability obligations. In other words, we will need a formalism for specifying remedies, actions that need to be taken*

[3]Data Processor: A natural or legal person, public authority, agency or any other body which processes personal data on behalf of the controller.

when something goes wrong, e.g. compensation for customers, fines, service suspension, etc. Notice that not all redress actions can be executed in an automated way. Some languages we reviewed support obligations, but not explicitly penalties (see Appendix A). The language must be able to express the cloud provider's obligations regarding the recovery from security attacks [MJH+13][R13]. Additionally, the language must be able to express what kind of evidences are required for the remediation actions, when recovering from a vulnerability or failure [MJH+13][R15].

Requirement 13 (Revocability) *The language must be able to describe the revocation of granted access and authorized usage of resources.*

Requirement 14 (Logging) *The policy language must specify which events have to be logged and what information related to the logged event have to be added to the log (such as, access time, name or pseudonym of the entity in charge of processing the resource, the role of that entity, the purpose of usage of the accessed resource, etc.). The policy language should express the way the information is logged (how, when, who carried out the action).*

Requirement 15 (Controlling Data Location and Transfer) *Clear whereabouts of location depending on the type of data stored or processed (which may be subject to specific regulations) must be provided. Accountability policies for cloud services must be able to express rules about data location allowing accountable services to signal violations of data transfer rules when they occur.*

Requirement 16 (Usage Control rules) *The policy language must be able to express usage control rules. Usage control extends access control over time, by also establishing obligations for the usage in the future. Thus usage control deals not only with the access of data but also with the way data may be used afterwards.*

Table 1.1 summarizes the requirements we considered for the A4Cloud policy framework. A mandatory requirement means that we will consider it as a priority in the design of the policy language and on its prototyping. An optional requirement will receive additional research effort but such requirements are considered as secondary objectives for the time being. This list of requirements further conducts our review of existing policy languages and the design of the A4Cloud policy framework. The study on the state of the art is introduced in Section 1.2.1.

1.2 A4Cloud Policy Representation Framework

1.2.1 Existing policy representation languages

A number of policy languages have been proposed in recent years for policy representation. We reviewed several existing policy languages, defined either as standards (XACML [OAS13], WS-* standards [OAS06, OAS12, OAS04] and P3P [P3P]) or as academic/industrial proposals (PPL [ABDCDV+09], USDL [BO12], SLAng [LSE03], SecPal4P [BMB10], Ponder [DDLS01] and ConSpec [AN08]) and we studied their suitability for accountability representation according to the accountability requirements identified in Section 1.1. The detailed state of the art study is provided in Appendix A and a summary of our review is given in Table 1.2.

Requirement	Mandatory	Requirement Category
(R1) Capturing Privacy Preferences	yes	Data Handling
(R2) Anonymity/Pseudonymity	optional	Data Handling
(R3) Data minimization	optional	Data Handling
(R4) Strong policy binding	optional	Data Handling
(R5) Data Retention Period	yes	Data Handling
(R6) Expressing Regulatory Constraints	yes	Accountability
(R7) Access Control rules	yes	Data Handling
(R8) Delegation of rights	yes	Data Handling
(R9) Auditability	yes	Accountability
(R10) Controlling Policy Disclosure	optional	Accountability
(R11) Reporting	yes	Accountability
(R12) Redress policies	yes	Accountability
(R13) Revocability	yes	Data Handling
(R14) Logging	yes	Accountability
(R15) Controlling Data location	yes	Accountability
(R16) Usage Control rules	yes	Data Handling

Table 1.1: Summary of the A4Cloud policy framework requirements

In this review, the main question was to determine the ability of the existing policy languages to represent accountability concepts as defined in Section 1.1. It turns out that none of the existing standards could satisfy all the accountability requirements we previously defined. While access control rules are generally covered by existing standards, delegation, chain of trust and auditing are only addressed by some academic proposals. Furthermore, the related work does not support the definition of actor roles (i.e. data subject, data controller, etc.) as defined by the EU data protection directive [Eur95].

On the other hand, while analyzing the suitability of existing languages to accountability requirements, we realize that PPL can be considered as a good candidate for the new A4Cloud policy framework since it partially covers data handling issues and can be extended to address accountability related requirements. Additionally, since the mapping of natural language obligations into machine readable languages could be error prone we have also studied some academic approaches to enable an easy and efficient representation of accountability obligations. Accordingly, we provide in the next section a preliminary sketch of the design process that we will follow to develop our policy representation framework and its associated policy languages.

Table 1.2: State of the art analysis according to our requirements

	XACML		PPL		WS-* Sec.		USDL		SLAng		P3P		SecPal4P		Ponder		ConSpec	
Requirement	**C**	**E**	**C**	**E**	**C**	**E**	**C**	**E**	**C**	**E**	**C**	**E**	**C**	**E**	**C**	**E**	**C**	**E**
(R1) Capturing Privacy Preferences	/	Y	/	Y				Y		Y	Y	Y	Y			Y	Y	
(R2) Anonymity/Pseudonymity	Y		Y							Y						Y		Y
(R3) Data minimization		Y	Y					Y			Y	Y				Y	Y	
(R4) Strong policy binding		Y	Y					Y				Y				Y		
(R5) Data Retention Period	/	Y	/	Y			/	Y		Y	Y	Y	Y			Y	Y	
(R6) Expressing Regulatory Constraints	Y		Y				Y									Y		
(R7) Access Control rules	Y		Y				/	Y			Y				/	Y	Y	
(R8) Delegation of rights	/	Y	/	Y				Y							Y			
(R9) Auditability		Y		Y		Y	/	Y	/	Y	/	Y			Y			
(R10) Controlling Policy Disclosure		Y	Y			Y		Y		Y						Y		
(R11) Reporting		Y	/	Y			/	Y			/				Y			Y
(R12) Redress policies		Y		Y		Y	/	Y								Y		
(R13) Revocability		Y		Y				Y							Y			Y
(R14) Logging		Y	/	Y		Y	/	Y		Y					Y			Y
(R15) Controlling Data location		Y		Y				Y		Y		Y	Y			Y		Y
(R16) Usage Control rules		Y	Y				/	Y				Y			/	Y	Y	

Legend
C: Covered
E: Extensibility
Y: Yes
 : No
/ : Partly covered

1.2.2 Overview on the proposed framework

In this work, we aim to provide a policy representation framework taking into account our identified accountability requirements. Accountability obligations are often described using natural sentences in regulations and contractual agreements, while they may also reflect social norms. The purpose of the framework is to map these obligations to machine readable sentences expressing security policies and other requirements contained in the obligations. As our analysis of the state of the art shows, there is currently no language that covers all the accountability requirements. For instance, PPL seems to be the most adequate language but it fails to meet, exactly, some of our accountability requirements (R1, R5, R8, R9, R11, R12, R13, R14 and R15, see Appendix A for details). As previously mentioned, mapping accountability obligations to machine readable sentences is easier if we define an intermediate pivotal language, which is the *Abstract Accountability Language* (AAL for short). This can be viewed as a domain specific language for accountability obligations and is related to the approaches proposed in SecPAL [BFG10] for security, with privacy in [LM09] and in [DGJ$^+$10]. Authors of [LM09] have suggested to use a limited but formal language adapted to the legal context. DeYoung et al. in [DGJ$^+$10] define a formal language with temporal modalities and demonstrates it application to the formalisation of the HIPPA laws. Translating natural texts using natural language processing techniques is a hard task and following the experience from [PT12, FJG$^+$13] it is sufficient to use more lightweight and rigid but dedicated techniques. Our analysis of the state of the art regarding the means to do a seamless connection between legal texts and policy languages is explored in Appendix A. The new AAL language should be simple, regular and close to natural expression of accountability obligations. As a first validation step, in Section 4 we list the obligations of the health care use case and we use AAL to express these obligations. We also review the Privacy Level Agreements (PLAs) from CSA[4] to study the expressive power of AAL (see details in Section 2.4.1). These agreements are intended to be used in cloud service contracts to describe the level of privacy protection the provider will ensure. PLAs address information privacy and personal data protection practices but without a precise and formal syntax. For a more formal approach, we also looked into the ENDORSE[5] project which defines a precise grammar with notions of actor, action, data object, purpose and obligation modalities.

Conceptually, we consider two levels: the obligation level and the enforcement level. The obligation level is closely related to natural language obligations for accountability, while the enforcement level should be machine readable for sake of automation. The general view of the mapping process is split in two parts: From accountability obligations to AAL and from AAL to machine understandable policy languages at the enforcement level. In this document we mainly focus on the language descriptions, the mapping from obligations to machine-understandable languages is the purpose of the next deliverable.

Our framework must allow privacy officers and consumers to **easily** express their accountability obligations and preferences respectively and even be **complete** and **rigorous** enough to be interpreted by a policy execution engine. Hence we are faced with the following dilemma: policies (resp. preferences) must be written by cloud services providers or data subjects, who do not necessarily have skills in a certain policy language and the policies/preferences must be machine understandable at the same time. In this context, we propose a policy representation framework (see Figure 1.1)

[4]`https://cloudsecurityalliance.org/research/pla/`, last visited on 21/11/2013.
[5]`http://ict-endorse.eu/`, Deliverable D4.5, last visited on 21/11/2013.

that allows a user, step (1) in Figure 1.1, to express his accountability needs in a human readable fashion and (2) offers the necessary means to translate them semi-automatically[6] to a machine understandable format. Of course, there will be nothing preventing advanced users to manually map obligations to A-PPL, however, we suspect that most users will prefer the more human readable AAL. In the following, we detail steps (1) and (2).

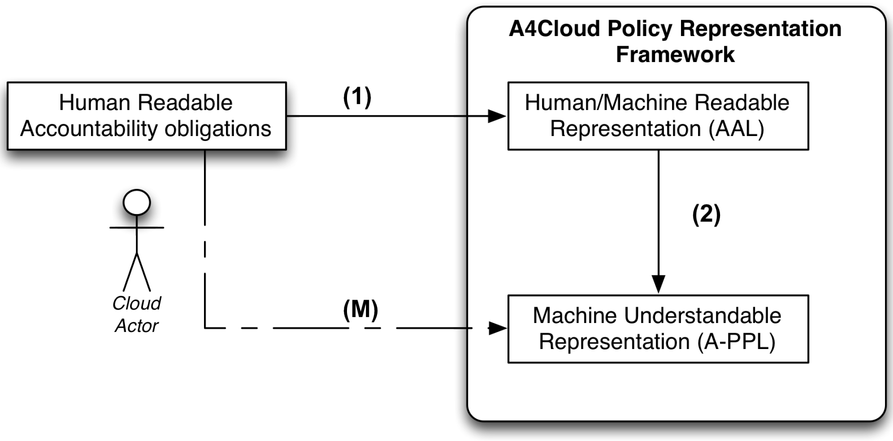
A4Cloud Policy Representation Framework
Human Readable Accountability obligations
(1)
Human/Machine Readable Representation (AAL)
(2)
Cloud Actor
(M)
Machine Understandable Representation (A-PPL)

Figure 1.1: Overview on the A4Cloud policy representation framework

Step (1). Human/machine readable representation of the accountability needs
To express the accountability needs we define the Abstract Accountability Language (AAL), which is devoted to expressing accountability properties in an unambiguous style and which is closer to what the data subject or the cloud service provider needs and understand. As this is the human readable level, this language should be simple, akin to a natural logic, that is a logic expressed in a subset of a natural language.

Step (2). Translation into a machine understandable representation
In this step, the end user's policy expressed in AAL is semi-automatically translated into a machine understandable policy. The AAL obligation from step (1), for example "`MUST NOT ANY:User.READ(d:Data)`", will be translated into A-PPL which is similar to a XACML [OAS13] machine readable representation. In Figure 1.2 we give as an example a pseudo-schema of such a policy.

Manual translation, from legal texts or AAL clauses to A-PPL, is possible but can be cumbersome and error-prone. Despite the expected simplicity of AAL, such automatic translation cannot be done directly in a simple step. Hence we propose to first map the abstract policy represented in AAL to a more formal form.

Figure 1.3 provides an overview of this second step. According to this figure, we can see that going from an AAL representation to a machine understandable representation of the accountability obligations, (2) in Figure 1.3 , is done through three steps:

[6]Our goal is to make a fully automatic mapping, however, user interaction may be required in certain cases. This mapping is part of our future work.

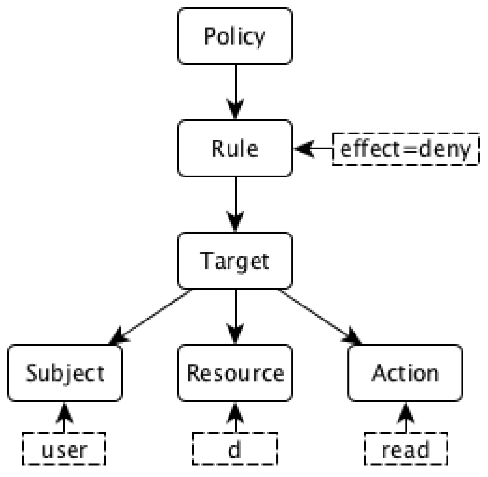

Figure 1.2: Pseudo-schema of a A-PPL policy

- **(2'.1).** First, a temporal logic is used to make more concrete AAL sentences as temporal logic properties. In an accountability policy we should introduce notions of permission, obligation and prohibition. These deontic notions are classical in contracts and in regulation, and appear under different forms in ENDORSE, PPL, PrivacyLFP, or SecPAL to name a few. In addition, there is a need to express conditions and various logical combinations. Furthermore, one important thing is to have time, at least logical discrete time, for instance to write: "the data processor writes data and **then** stores the associated logs", "the data subject sends his personal data **only after** giving his informed consent to the data controller".
- **(2'.2).** Second, a policy calculus is used to describe the operational semantics associated to the concrete properties defined in (2'.1). This operational semantics provides means for abstractly executing the temporal logic expressions. This process is generally known as "program synthesis", starting from a property it generates a program ensuring the property.
- **(2'.3).** Finally, the generated policy (with concrete properties written in temporal logic and with an operational model described using our policy calculus) is semi-automatically translated to a machine understandable policy based on predefined transformation rules. Currently the target is A-PPL to support accountability requirements.

In parallel with the design of AAL, the underlying temporal logic language and the policy calculus, we also focus on the extension of PPL to address accountability requirements. Translating accountability obligations/rules into A-PPL can also be done manually when the user/cloud actor has the necessary skills. PPL seems being the most convenient language to extend for A4Cloud in order to address specific requirements such as auditability, notification or logging obligations. Therefore, this document describes the proposed extensions to PPL in order to map A4Cloud obligations.

Note that, in this deliverable we focus on the language descriptions, the mapping overview and illustrative examples derived from WP:B-3. How to actually enforce will be part of our future work in which we will also detail the three steps of Figure 1.3 and the A-PPL enforcement engine.

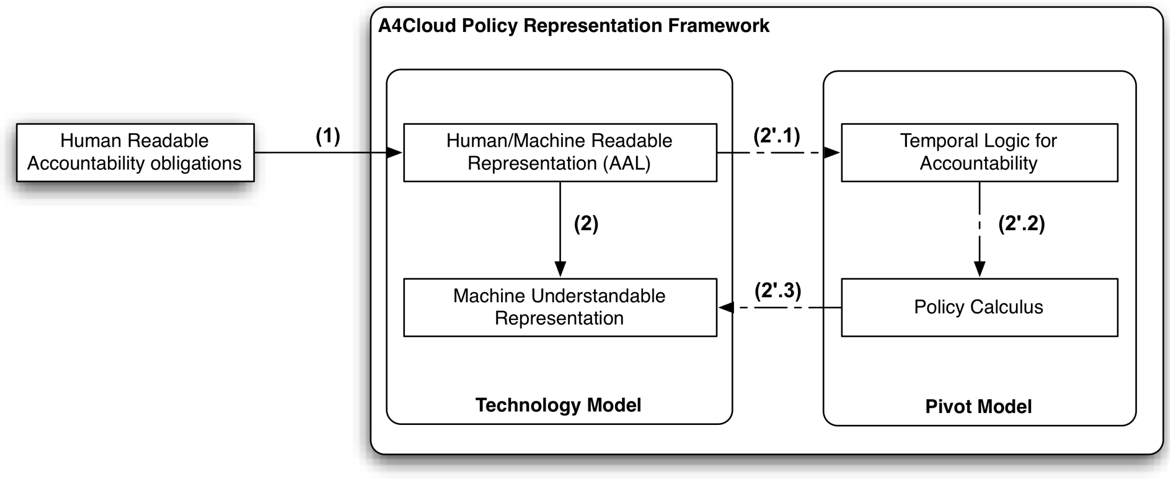
A4Cloud Policy Representation Framework
Human Readable Accountability obligations
(1)
Human/Machine Readable Representation (AAL)
(2'.1)
Temporal Logic for Accountability
(2)
(2'.2)
Machine Understandable Representation
(2'.3)
Policy Calculus
Technology Model
Pivot Model

Figure 1.3: Overview on the machine understandable translation of AAL

The used temporal logic and policy calculus can be seen as the pivot model of the A4Cloud policy representation framework representing the key for automatic translation of AAL to A-PPL. That means that changing the operational target (here A-PPL) does not have much impact on this pivot model. This is also true with the AAL description which could evolve or be changed. This mapping schema decouples the input AAL language from the operational level making our work more general.

Finally, this framework provides an abstract and expressive language for accountability and a flexible system to configure the mapping to a concrete policy language. It will be equipped with some tools but still be open for reusing existing tools like secure logging, auditing, and collection of evidences. All these features are considered important for cloud security measures in the ENISA report on auditing security measures [DKLL13].

Chapter 2

Specifications of the Abstract Accountability Language

In this chapter we first describe the lightweight grammar of AAL and further show some simple accountability properties we expect to express through the description of a simple example. This example also demonstrates that without underlying semantics we cannot really achieve accountability mapping and concrete enforcement. Section 2.2 describes a formal policy calculus and its associated logic. In a second step (Section 2.3) we validate their design against the requirements given in Section 1.1.

2.1 AAL Introduction

The purpose of this section is to informally present a language which is dedicated to accountability expressions as they appear in legal, regulatory and normative texts. This language should be concise and to get an overview of the language a simple example is described. We present a simple grammar to express accountability properties comprising usage control expressions, logging and auditing actions, and possible punishment. Note that we will use the term *rectification* to denote any computer representable action that could be done as a consequence of violating an obligation. Rectification is considered hereby as the principle taking actions aimint at correcting and obligation violation.

2.1.1 Lightweight Grammar

We give here the main rules for a simplified draft syntax of AAL. This is a brute version of AAL, some hints to improve writing and readability are proposed at the end of Chapter 2. We present some simple rules for the AAL grammar where `[ ]` is an option and keywords are uppercase words.

```
Clause ::= [Temporal] (Exp_usage [AUDITING Exp_audit]
                                [IF_VIOLATED_THEN Exp_rectif] )+
Temporal ::= ::= ALWAYS | EVENTUALLY | NEVER
```

An accountability clause has three parts and it is optionally prefixed by a temporal operator. The first part describes data handling rules, the second is an audit action and the last specifies rectifications. This rule defines the general form of accountability clauses. Its informal meaning is to *try to ensure the data handling rules but if the audit reveals certain violations then rectification actions will be taken*. The expressions for auditing and rectification are optional, since in some cases it is convenient to avoid them or to rely on implicit actions. The optional temporal clause means that the property globally, eventually or never stands. AAL clauses are aligned with the analysis of [FJWX12] which identifies five steps in an accountability framework: prevention, detection, evidence, judgment, and punishment. The prevention part corresponds to our usage control expression, detection, evidence and judgment are included in the audit action and the punishment is covered by the more general notion of rectification.

An expression denotes both the properties (access control, usage control, ...) and actions which can be done as remediation or sanction.

```
Exp ::= Modal | Exp OR Exp | Exp AND Exp | Exp ONLYWHEN Exp | Exp THEN Exp
```

Expressions are compound from conjunction and disjunction operators but also with a past and a future operators. The `ONLYWHEN` operator has a past flavor: the first `Exp` can occur if the second one was true in the past. The `THEN` operator is a logical implication but often it is followed by `MUST` which is a future action.

```
Modal ::= MAY Action | MUST Action | MUSTNOT Action
```

The basic expressions are related to permission, obligation and prohibition. `MAY action` means that the action can or cannot occur but other actions are prohibited to occur. `MUST action` means that action should occur while `MUSTNOT` denotes a prohibition. The notion of prohibition is sometimes viewed as an obligation to not doing something (see SecPAL for instance), but in deontic contracts, ENDORSE proposal, PPL, and many other approaches there are explicit constructions for permission, obligation and prohibition.

An action is in fact the precise description of some events an agent could do by using some operations with the necessary parameters.

```
Action ::= Agent . Oper (Parameters)  [BEFORE Time] [AFTER Time]
Time ::= Date | Duration
```

The actions we consider are divided in two sets : basic operations and services provided by some agent (`Agent_provider`). This set can be extended for specific needs. For instance, we are still considering if the distinction between send and receive actions is relevant and if we need a specific action to update a policy.

```
Oper ::= READ | WRITE | LOG | SEND | NOTIFY | ... | SERVICE[Agent_provider]
```

We consider that agents are identified by Uniform Resource Identifiers (URIs).

```
Agent ::= URI
```

Parameters are not described in detail here to simplify the syntax presentation.

```
Parameters ::= Constant | Variable | ....
Constant ::= string
Variable ::= string:string
```

The grammar is not complete. for instance, we postponed the defining of syntactic constraints. This first attempt does not give too much details about data types, services and actions declarations. However, these are not critical issues and we will provide a detailed and complete description of AAL in the next deliverable.

2.1.2 A Simple Example

In this example we consider a Cloud provider that offers a free blog-publishing service to individual end-users. The service requires the users to register before they can start using it, which means that they will provide personal data to the Cloud provider. To make our purpose more precise and to make the policy writing more explicit we consider the following component diagram in Figure 2.1 (in a UML 2.0 style). This scenario is used to illustrate a general situation, and to give some precise examples of obligations written with AAL.

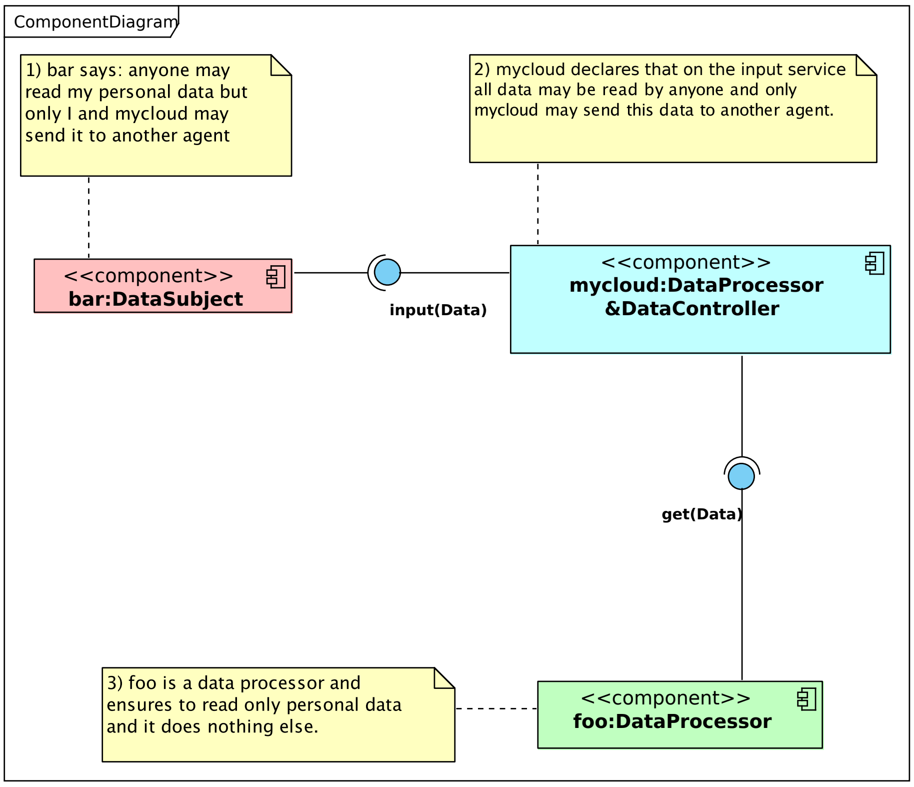
ComponentDiagram
1) bar says: anyone may read my personal data but only I and mycloud may send it to another agent
2) mycloud declares that on the input service all data may be read by anyone and only mycloud may send this data to another agent.
<<component>> bar:DataSubject
input(Data)
<<component>> mycloud:DataProcessor &DataController
get(Data)
3) foo is a data processor and ensures to read only personal data and it does nothing else.
<<component>> foo:DataProcessor

Figure 2.1: A Simple Scenario

In this scenario we have three components representing a data subject (`bar`) a provider (`mycloud`) and a data processor (`foo`). The data subject expresses preferences for his data and the provider defines a provided service and declares an obligation for this service (input). In this simple scenario a data is communicated from bar to the provider via the input service. This data is expected to reach processor foo using the get service. And this last agent declares that he will only read the data. In this example the mycloud data processor is also the data controller. We should note the following points which correspond to the numbering comments in the figure.

1. A data subject writes some preferences regarding the management of the personal data he/she will disclose to a data controller.
2. The data subject's personal data is a parameter of the input service which defines an accountability policy. This is the policy the processor and controller declare to comply with. The data controller should ensure the declared policy rules even in case of redistribution of the data.
3. Another processor can access the data with the get service, this processor should define a policy compliant with the policy of the data from mycloud.

In such a scenario we must express user preferences and provider obligations. Data is sent to an agent using a service (for instance, the input service) only if the user preference policy attached to data matches the service provider obligation. We assume that a data is only readable if the agent gets it first, or if the agent owns this data. In this chapter, we will use some simple notations: words in capital denote actions or services of an agent (examples are READ, SEND), lower case words denote attributes, that is information attached to an agent or a data (for instance, authorizedList). Indentation in AAL clauses has no specific meaning.

2.1.3 AAL Samples

The bar user preferences for its data d could be expressed as follows.

```
MAY ANY:Agent.READ(d:Data) OR MAY bar.INPUT(mycloud, d:Data)
                           OR MAY mycloud.GET(X:Agent, d:Data)
```

Where ANY and X denote any agent and d any personal data of bar. Anyone may read the data d, but due to our assumption above, provided that the agent before receives the data (using some of its provided services). Here "anyone may read d" is not realistic, but we could write something like:

```
(ANY:Agent in mycloud.authorizedList) THEN MAY ANY.READ(d:Data) ...
```

Where THEN is a logical implication (usually written =>), the expected meaning is that any agent in the authorized list may read the data (formally the implicit universal quantification of data and agent is ahead of the clause). The policy from mycloud is:

```
MAY ANY:Agent.READ(d:Data)  OR MAY mycloud.GET(X:Agent, d:Data)
```

The policy declared by the foo component is: MAY foo.READ(d:Data).

Comparing user preferences and service provider policies is a fundamental part of any policy language. Thus one simple case is to compare the obligations declared by mycloud and foo for compliance with the preferences expressed by bar.

The previous expressions only denote access control or usage control rules. AAL should in addition express accountability needs such as specifying logs, auditing and actions taken in case of policy violation that we name as rectifications. For instance, the bar user preferences may be written as follows with an explicit notification (but more complex sanctions could be defined, for instance, compensation owed by the Data Controllers to data subjects, as provided in the Article 23 of the Data Protection Directive).

```
MAY ANY:Agent.READ(d:Data) OR MAY bar.INPUT(mycloud, d:Data)
                           OR MAY mycloud.GET(X:Agent, d:Data)
 IF_VIOLATED_THEN
       (MUST mycloud.NOTIFY(mycloud.controller, "obligation ... violated")
        AND MUST mycloud.NOTIFY(bar, "obligation ... violated"))
```

The expected meaning is, if a violation of the usage is observed then the provider must notify the data controller and the data subject about it. This violation may originate from an audit (which is not explicitly stated here) or result from a complaint or a controller checking. Here, we can note that these actions should be done before a given date (for instance, according to the Article 31 of the proposed Data Protection Regulation[1], the data controller has to notify the data breach to the supervisory authority not later than 24 hours, after becoming aware of it) which implies the need of explicit deadlines.

We can also explicitly declare the logging and auditing steps. In the version below we oblige the provider to log information (`X, d`) when sending the data to someone else.

```
MAY ANY:Agent.READ(d:Data) OR MAY bar.INPUT(mycloud, d:Data) OR
(MAY mycloud.GET(X:Agent, d) THEN MUST mycloud.LOG(X, d)
IF_VIOLATED_THEN
       (MUST mycloud.NOTIFY(mycloud.controller, "obligation ... violated")
        AND MUST mycloud.NOTIFY(bar, "obligation ... violated"))
```

Below is an example with an explicit audit action done by an auditor.

```
MAY ANY:Agent.READ(d:Data) OR MAY bar.INPUT(mycloud, d:Data)
  OR (MAY mycloud.GET(X:Agent, d:Data) THEN MUST mycloud.LOG((X, d)
AUDITING MUST mycloud.auditor.AUDIT(mycloud.logs)
IF_VIOLATED_THEN
      (MUST mycloud.NOTIFY(mycloud.controller, "obligation ... violated")
      AND MUST mycloud.NOTIFY(bar, "obligation ... violated"))
```

2.1.4 Discussion

Simple expressions are easy to read but as far as they are longer or there are several of them, their meaning may become unclear. In the previous examples there is no guarantee that these expressions are meaningful. One can note the use of several operators like `MAY, MUST` which are convenient but sometimes they need to be rephrased for readability. There are also names and

[1] http://www.twobirds.com/en/news/articles/2013/global/libe-committee-of-the-euro-parliament-votes-on-compromise-amendments-to-the-draft

variables which should be properly defined and linked, formally that means we need universal and existential quantifiers and a precise notion of lexical scope. Comparing two policies or verifying if one log set or execution trace is compliant with a policy would need a careful analysis. For instance, does bar's preferences match mycloud's obligation? With our simple example it is quite obvious.

```
MAY ANY:Agent.READ(d:Data) OR MAY mycloud.INPUT(X:Agent, d:Data)
=>
MAY ANY:Agent.READ(d:Data) OR MAY bar.INPUT(mycloud, d:Data)
                           OR MAY mycloud.GET(X:Agent, d:Data)
```

It can be seen as the logical expression `A => (A OR B)`. However, if we add explicit logs, auditing and rectification actions, checking the consistency of an AAL clause becomes more complex. Therefore we will define a clear syntax and rigorous semantics with a formal logical interpretation. The main syntactic elements and some intuitions are described in the next section (Section 2.2). To implement programs interpreting this logic we also provide an operational policy calculus. A program in such a calculus is an abstract but operational way to satisfy the AAL obligations. Such a program can be then translated into languages that deal with policy enforcement like A-PPL. We note that in Section 2.4, we return to AAL to summarise its informal interpretation and discuss some issues related to its design.

2.2 A Policy Calculus and its Logic

The previous examples and discussion lead to the conclusion that formal semantics are needed. But to be properly achieved two levels are required: a logical level and an operational one. Accountability obligations are logical properties of a system (invariant) while policy enforcement is an operational mean to ensure the obligations. We introduce a temporal logic in Section 2.2.4 since it is close to AAL which has a declarative and logical flavor. An interpretation of all AAL expressions is possible with temporal logic. The second level is a policy calculus since we need an operational model to deal with explicit policy enforcement. The calculus uses the classic notions of message-passing models and of reference monitors while being fully executable but still abstract. Essentially, the policy calculus allows reference monitors to be specified. The two levels are strongly related: given a description of the behavior of the agents controlled by the reference monitors, execution traces can be operationally generated. Then a property expressed in the temporal logic can be checked against this set of execution traces: this is a model-checking problem but we need to consider a universal quantification over the behavior of agents. During the specification phase of the policy, there is also a synthesis phase: starting from an AAL expression, first provide an interpretation in the temporal logic, and second synthesize the specification of the reference monitors in the the policy calculus. The next section introduces more precisely the policy calculus.

2.2.1 Introduction to the Policy Calculus

The policy calculus allows policies to be defined. To define a policy, we introduce an operational but abstract model, based on agents exchanging messages. Agents are controlled by reference monitors. A reference monitor is akin to a policy enforcement point in XACML. We adhere to the general standard ”A Framework for Policy-based Admission Control” proposed by IETF[2].

A policy abstractly specifies the behavior of reference monitors controlling agents. It can express accountability concerns as well as security concerns. When the behavior of the agents under the control of the reference monitors is known, it becomes possible to abstractly execute the whole system: the execution generates a trace as a sequence of states. A state of the whole system is defined from the states of all agents and of their associated reference monitors, and from the state of the network connecting the agents. Above this trace semantics, a logical layer can be added: logical properties can be checked over the traces. The logical properties are expressed in a temporal logic, which depends, like any temporal logic as recalled in Section 2.2.4, on another logic used to express properties over states. Therefore we introduce two logics, summed up as follows.

- State Logic: propositional logic used to describe properties of the states of the reference monitors
- Trace Logic: first-order linear-time temporal logic used to describe properties of traces (sequence of states) and based on the state logic to express atomic properties of states.

Note that the state logic only refers to the state of the reference monitors, which compose the trusted computing base. The real agents are encapsulated in trusted reference monitors. The local states of the agents are considered as black boxes since they are under the control of reference monitors. The trace logic can be related to a human-readable logic, following an abstraction process, which essentially correspond to an abstraction of the states: this is essentially the relation with AAL. The policy calculus can also be related to any language for machine understandable representations of policies: by its generality, it can become a pivot language, allowing semantic interpretations and specifications to be given independently of any technology.

2.2.2 Operational Model

We consider a message-passing model at a high level of abstraction. We choose a chemical model [BB90], well-suited to concurrent and distributed systems. In this model, based on a chemical metaphor, agents correspond to sites where chemical reactions can occur. A reaction consumes and produces atoms. Each produced atom then migrates to the site where it can react. If we consider messages as atoms, we effectively get a message-passing model. The global state of a chemical system can be described as multisets of atoms[3], one multiset per agent and a multisetfor the soup where agents float, that is the network.

Agents exchanging Messages. More concretely, agents exchange messages via services (equivalent to channels). Agents and messages have user-defined attributes. We use a dot notation to access attributes: if x is an identifier, then $x.c$ is the value of attribute c of the object associated to identifier x. In order to factorize definitions, classes of objects can be defined: all the objects of the class have the same attributes. We review now some (possible) classes.

[2]See RFC 2753.

[3]A multiset is a set with possible duplicates. It is equipped with an operator, the union, that is commutative and associative: it is the perfect yet simple candidate to interpret parallelism.

Actor	Resource
role	kind
	content
provides services	provides services
required services	required services

Table 2.1: Attributes of Agents

identity
kind
content
source
target
service (called)

Table 2.2: Attributes of Messages

An agent is either an actor or a resource. It has attributes depending on its class. See Table 2.1 for a core definition. According to their roles or kinds, agents may have other attributes. A message also has attributes. An important feature for messages is that new messages can be generated. Thus, we assume that it is possible to produce fresh identifiers for messages. A message has therefore a special attribute that is read-only and initialized at the creation: call it `id`. Table 2.2 gives a first list for message attributes. A message can have three states, according to its status during the communication. Consider some message m. If it just goes out of an agent x and is inside the associated reference monitor, it is in a positive state, and is denoted $+m$. We have therefore for any message m in a positive state inside the reference monitor a wrapping agent x:

$$m.\texttt{source} = x.$$

If it is ready to be delivered to the agent and is still inside the reference monitor, it is in a negative state and is denoted $-m$. We have therefore for any message m in a negative state inside the reference monitor a wrapping agent x:

$$m.\texttt{target} = x.$$

Otherwise, it is simply denoted m: it is either in transit in the network, or inside a reference monitor, ready to be sent or just received from the network. In summary, a message evolves according the following transitions:

$$+m \longrightarrow m \longrightarrow -m.$$

Reference Monitors. To enforce policies, reference monitors are used. Each agent is wrapped by a reference monitor. [And74] defines a secure reference monitor in terms of satisfying three criteria:

- Non-Bypassability Criterion: a reference monitor is always invoked during each communication.
- Tamperproofness Criterion: a reference monitor cannot be modified by unauthorized agents.
- Verifiability Criterion: a reference monitor can be proved correct, with a sufficient level of trust.

In the operational model, the first criterion is satisfied. The second criterion is satisfied by assumption : only reference monitors, which are trusted, are allowed to modify the policies (their own policy or the policy of other monitors). As for the third criterion, we will briefly show how it can be satisfied.

A reference monitor has a state. Precisely, following the chemical model, we choose a state representation as a multiset of atoms. An atom corresponds either to an incoming message (coming from the network), an outgoing message (coming from the agent) or an atomic fact expressing its internal state. An atomic fact is a relation applied to terms. A term is either an identifier (of an agent or a message), or a primitive value (`int`, . . .), the value of an attribute, a record or a queue. The term definition can easily be extended to satisfy particular needs. Currently, we select these constructs as they are useful in the examples presented in Section 2.3.

$$
\begin{array}{llcll}
\texttt{state} & \sigma & ::= & m \mid +m \mid -m & (\texttt{message}) \\
 & & \mid & R(t^*) & (\texttt{atomic fact}) \\
 & & \mid & \&\, \sigma^* & (\texttt{conjunction of atoms}) \\
\texttt{message} & m & ::= & (id, [(c=t)^*]) & (\texttt{identifier}, \texttt{record}) \\
\texttt{relation} & R & ::= & \ldots & (\texttt{identifier}) \\
\texttt{term} & t & ::= & x & (\texttt{identifier}) \\
 & & \mid & v & (\texttt{primitive value}) \\
 & & \mid & x.c & (\texttt{attribute}) \\
 & & \mid & [(c=t)^*] & (\texttt{record}) \\
 & & \mid & ::\ t^* & (\texttt{queue})
\end{array}
$$

The specification of the behavior of a reference monitor is defined as a set of rules, and a policy as a set of specifications.

$$
\begin{array}{llcll}
\texttt{rule} & r & ::= & \sigma_1 \rightarrow \varphi \mathbin{?} \sigma_2 & (\texttt{rule: when } \sigma_1\texttt{, if } \varphi\texttt{, then } \sigma_2) \\
\texttt{specification} & s & ::= & x[r]^* & (\texttt{rules of the reference monitor} \\
 & & & & \ \texttt{wrapping agent } x) \\
\texttt{policy} & p & ::= & s^* & (\texttt{set of specifications})
\end{array}
$$

A rule $\sigma_1 \rightarrow \varphi \mathbin{?} \sigma_2$ expresses a transition of the reference monitor: when the state is σ_1 and if the condition φ holds, then the state becomes σ_2. More precisely, σ_1 is a pattern that binds variables in φ and σ_2 when it matches a concrete state. However, in the following, we omit the instantiation phase, making no distinction between rules and transitions. It remains to specify the logic used to express conditions φ, used as guards in rules. Actually, the guard logic is a restriction of the state logic.

$$
\begin{array}{llll}
\texttt{formula} & \varphi & ::= & \texttt{true} \mid \texttt{false} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi & \\
& & \mid & P(t^*) & (\texttt{equality, comparison, ...}) \\
& & \mid & q \vdash \psi & (\texttt{audit}) \\
& & \mid & x[\sigma \rightarrow \varphi] & (\texttt{introspection})
\end{array}
$$

Table 2.3: State Logic: Syntax

$$
\begin{array}{rcl}
I(\Delta \vDash \texttt{true}) & = & \texttt{true} \\
I(\Delta \vDash \texttt{false}) & = & \texttt{false} \\
I(\Delta \vDash \neg\varphi) & = & \neg I(\Delta \vDash \varphi) \\
I(\Delta \vDash \varphi_1 \vee \varphi_2) & = & I(\Delta \vDash \varphi_1) \vee I(\Delta \vDash \varphi_2) \\
I(\Delta \vDash \varphi_1 \wedge \varphi_2) & = & I(\Delta \vDash \varphi_1) \wedge I(\Delta \vDash \varphi_2) \\
I(\Delta \vDash P(t_i)) & = & P(t_i) \\
I(\Delta \vDash (q \vdash \psi)) & = & (q \vdash \psi) \\
I(\Delta \vDash x[\sigma \rightarrow \varphi]) & = & \exists \sigma'.(x[\sigma'] \in \Delta) \wedge (\sigma \sqsubseteq \sigma') \wedge I(\Delta \vDash \varphi)
\end{array}
$$

Table 2.4: State Logic: Semantics

The State Logic. The state logic is a propositional logic with three types of atomic formulas. It allows the states of the reference monitors to be described.

- $P(t_1, \ldots)$: predicate P between terms $t_1, \ldots$. Equality, comparison and other Boolean operators can be used as a predicate.
- $q \vdash \psi$: queue q satisfies property ψ. The property ψ is expressed in a temporal logic (depending on the domain of the queue and a logic over this domain). The atomic formula is used for audit, after significant events have been stored in the queue.
- $x[\sigma \rightarrow \varphi\,?]$: rule $\sigma \rightarrow \varphi\,?\sigma'$ (for some σ') is able to fire in the state associated to the reference monitor wrapping x. The atomic formula allows introspection, inside the reference monitor.

See Table 2.3 and Table 2.4 for the formal definition of its syntax and of its semantics respectively. We denote by Δ the context, that is a sequence of states $x[\sigma]$, where x is an agent and σ is the state of the reference monitor wrapping x. The semantics defines the Boolean interpretation I of judgments $\Delta \vDash \varphi$, which means that the states of the reference monitors described with Δ satisfy property φ. For introspection, we use the notation $(\sigma \sqsubseteq \sigma')$ to mean that multiset σ is included in multiset σ'. The guard logic used in rules specifying reference monitors is a simple restriction of the state logic: the context is limited to a unique agent, the one wrapped by the reference monitor.

We can now express some interesting properties. For instance, we can express the invariant properties mentioned earlier about messages, by using introspection.

```
∀ m : Message, x : Agent.
    (x[−m → true] ⇒ (m.target = x))
  ∧ (x[+m → true] ⇒ (m.source = x)).
```

In words: for any message and any agent, if the reference monitor is ready to deliver the message to the agent, then the target of the message is the agent, and if the reference has received a message from the agent, then the source of the message is the agent. These invariant properties are clearly required for consistency. As a second example, consider the following property: the reference monitor wrapping agent x contains a log and the queue of events saved in the log satisfies property ψ, which gives the following formula:

```
x[Log(q) → (q ⊢ ψ)].
```

Note that the arrow $\rightarrow$ is not a logical implication in the preceding formula, which expresses that any rule $\texttt{Log(q)} \rightarrow (\texttt{q} \vdash \psi)$? . . . is able to trigger in the reference monitor associated to x.

2.2.3 Operational Semantics

To express the semantics of the calculus, we need to specify the behavior of each agent. The behavior is defined as a (possibly infinite) set of rules:

$$x[\Sigma_1 \xrightarrow{-M_1/+M_2} \Sigma_2].$$

The rule means that if agent x is in state Σ_1 and can consume messages $-M_1$ (a sequence of messages $-m_1$ coming from the reference monitor), then agent x produces new messages $+M_2$ (a sequence of messages $+m_2$ at destination of the reference monitor) and moves to state Σ_2. The identifiers of messages in M_2 are fresh; moreover, each message m in M_2 satisfies $m.\texttt{source} = x$ and each message m in M_1 satisfies $m.\texttt{target} = x$, by consistency with the invariant properties satisfied by messages. As usual, the chemical semantics is defined thanks to inference rules. The transition relation $\Rightarrow$ is defined over a set of states describing the state of each agent and of its associated reference monitor and the state of the network, which contains messages in transit. The state $x[\Sigma][\sigma]$ associated to agent x has two components:

(i) Σ represents the local state of the agent x and is described abstractly, as a black box, without specifying its internal structure,

(ii) σ represents the state of the reference monitor wrapping x and is described as a multiset of atoms.

Semantic states are formally defined as follows:

semantic state	Γ	$::=$	m	(message in transit)
		$\mid$	$x[\Sigma][\sigma]$	(agent x with state Σ and monitor state σ)
		$\mid$	Γ^*	(set)

Finally, the transition relation is defined in Table 2.5. There are two communication rules, [IN] and [OUT], which are generic: they correspond to the communication with the network, in an asynchronous way. The first rule [IN] states that a reference monitor can receive a message towards the

$$\frac{m.\texttt{target} = x}{m, x[\Sigma][\sigma] \Rightarrow x[\Sigma][m \,\&\, \sigma]}\,[\texttt{IN}]$$

$$\frac{m.\texttt{target} \neq x}{x[\Sigma][m \,\&\, \sigma] \Rightarrow m, x[\Sigma][\sigma]}\,[\texttt{OUT}]$$

$$\frac{x[\sigma_1 \to \varphi \,?\, \sigma_2] \in p \quad x[\sigma \,\&\, \sigma_1] \vdash \varphi}{x[\Sigma][\sigma \,\&\, \sigma_1] \Rightarrow x[\Sigma][\sigma \,\&\, \sigma_2]}\,[\texttt{LOCAL}]$$

$$\frac{x[\Sigma_1 \xrightarrow{-M_1/+M_2} \Sigma_2] \quad x[\sigma_1 \,\&\, (-M_1) \to \varphi \,?\, \sigma_2] \in p \quad x[\sigma \,\&\, \sigma_1 \,\&\, (-M_1)] \vdash \varphi}{x[\Sigma_1][\sigma \,\&\, \sigma_1 \,\&\, (-M_1)] \Rightarrow x[\Sigma_2][\sigma \,\&\, \sigma_2 \,\&\, (+M_2)]}\,[\texttt{SYNCHRO}]$$

$$\frac{\Gamma_1 \Rightarrow \Gamma_2}{\Gamma_1, \Gamma \Rightarrow \Gamma_2, \Gamma}\,[\texttt{CHEMICAL}]$$

Table 2.5: Policy Calculus: Transition Relation

agent that it wraps. The second rule [OUT] states that a reference monitor can send a message to a remote agent through the network. The rule [LOCAL] describes an internal transition of the reference monitor, according to a policy p: an internal transition essentially consumes messages from the agent and the network and produces messages to the agent and the network, as explained in the next paragraph. The rule [SYNCHRO] describes the synchronization between the reference monitor and the agent. The messages ready to be delivered are delivered to and consumed by the agent, which also produces new messages. Finally, the rule [CHEMICAL] allows to make a transition in any context.

To ensure determinism (to some extent), consistency and simplification, the rules in a policy satisfy some constraints. They are split into two subsets, the set of local rules, which can be used in transitions [`LOCAL`] and the set of synchronization rules, which can be used in transitions [`SYNCHRO`]. Each local rule $x[r]$ in a policy p is assumed to satisfy the following elementary constraints.

(1) The rule r only consumes messages (i) $+m$ (satisfying $m.\texttt{source} = x$) and (ii) m provided that $m.\texttt{target} = x$.

(2) The rule r only produces messages (i) $-m$ provided that $m.\texttt{target} = x$ and (ii) m unconditionally.

The constraint for produced messages $-m$ is consistent with the invariant properties satisfied by messages. All other constraints allow communication rules to be triggered in a deterministic way. Assuming that a reference monitor initially contains no message.

(i) A rule [LOCAL] can only consume messages that come from the network via rule [IN] (m with $m.\texttt{target} = x$) or from the agent via rule [SYNCHRO] ($+m$);

(ii) A rule [LOCAL] can only produce messages that can be locally consumed (m with $m.\texttt{target} = x$) or go out towards the network via rule [OUT] (m with $m.\texttt{target} \neq x$) or the agent via rule [SYNCHRO] ($-m$ with $m.\texttt{target} = x$).

Invariants		
	$+m$	$m.\texttt{source} = x$
	$-m$	$m.\texttt{target} = x$

Transitions					
Local	$+m$		$\Rightarrow$	$-m$	$m.\texttt{target} = x$
	m	$m.\texttt{target} = x$		m	
Synchronization	$-m$		$\Rightarrow$		

Table 2.6: Messages: Invariants and Constraints (reference monitor associated to agent x)

Each synchronization rule $x[r]$ in a policy p is assumed to satisfy the following elementary constraints.

(i) The rule r only consumes messages $-m$ (satisfying $m.\texttt{target} = x$);

(ii) The rule r produces no message.

The constraints are consistent with the invariant already described. Table 2.6 sums up these properties. For instance, here are two synchronization rules.

```
x[−m → φ ? .]
x[−m & Log(q) → φ ? Log(m::q)]
```

The first one expresses that the agent can freely consume message $-m$ satisfying φ. The second one expresses that each message $-m$ satisfying φ is logged when it is delivered to the agent. A reference monitor that is neutral (or transparent), that is making no control on the messages, can be specified with two local rules and one synchronization rule.

```
x[+m → true ? m]
x[m → (m.target = x) ? −m]
x[−m → true ? .]
```

These rules allows self sending, which corresponds to the composition of the first two rules. To disallow self sending, it suffices to modify the first rule.

```
x[+m → (m.target ≠ x) ? m]
```

2.2.4 Trace Logic

The execution of a system with monitored agents is described as a sequence of states, where each state represents the state of the agents, the state of their wrapping reference monitors and the state of the network, composed of messages in transit. It is quite natural to express properties involving logical time on sequences, hence to use a temporal logic. We first give some background on temporal logic and then show how we use temporal logic in our framework, not only for the trace logic but also for the audit logic.

formula	ψ	$::=$	$\texttt{true} \mid \texttt{false} \mid \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \varphi$	(propositional formulas)
		$\mid$	$\exists x.\psi \mid \forall x.\psi$	(first-order formulas)
		$\mid$	$\texttt{X}\psi \mid \psi\texttt{U}\psi \mid \psi\texttt{R}\psi \mid \texttt{G}\psi \mid \texttt{F}\psi$	(temporal formulas (future))
		$\mid$	$\texttt{X}^{-1}\psi \mid \psi\texttt{U}^{-1}\psi \mid \psi\texttt{R}^{-1}\psi \mid \texttt{G}^{-1}\psi \mid \texttt{F}^{-1}\psi$	(temporal formulas (past))

Table 2.7: Temporal Logic: Syntax

Background on Temporal Logic. Linear temporal logic or more precisely linear-time temporal logic (LTL) is a modal logic with modalities referring to time. In LTL, one can encode formulas about the future or past of sequences of events, for instance execution traces. Given a sequence, it is possible to express that a condition will eventually be true, that a condition will be true until another fact becomes true, that a condition has been true. Precisely, the grammar is defined in Table 2.7. Note that φ ranges over any set of propositions defined over the set of events: this set is assumed to be given, as well as the interpretation of each proposition as a subset of events. Let us describe the temporal operators. We use both versions, for the past and the future, which are dual with respect to time. Our logic is not minimal as many operators can be encoded using a small subset of the operators. However, the encoding can lead to formulas having sizes exponentially larger than the original ones. That is the reason why we choose a rich set of operators.

Assume that a finite sequence of events is given, as well as a position in the sequence. $\texttt{X}\psi$ (read as "next ψ") means that ψ is true in the next position (in the sequence of events). $\texttt{X}^{-1}\psi$ (read as "previous ψ") means that ψ is true in the previous position (in the sequence of events). $\psi_1\texttt{U}\psi_2$ (read as "ψ_1 until ψ_2") means that for next positions, ψ_1 is true until ψ_2 becomes true. Its dual (with respect to negation) $\psi_1\texttt{R}\psi_2$ (read as "ψ_1 releases ψ_2") means that either ψ_2 is always true or ψ_2 is true until ψ_1 becomes true. Symmetrically, $\psi_1\texttt{U}^{-1}\psi_2$ (read as "ψ_1 since ψ_2") means that for previous positions, ψ_1 is true since ψ_2 has become true; its dual (with respect to negation) $\psi_1\texttt{R}^{-1}\psi_2$ (read as "ψ_1 fires ψ_2") means that either ψ_2 has always been true or ψ_2 is true after ψ_1 has become true. $\texttt{G}\psi$ (read as "always ψ") means that for next positions, ψ is always true. Its dual (with respect to negation) $\texttt{F}\psi$ (read as "eventually ψ") means that ψ will be true at some time. Symmetrically, $\texttt{G}^{-1}\psi$ (read as "up to now ψ") means that for previous positions, ψ has always been true; its dual (with respect to negation) $\texttt{F}^{-1}\psi$ (read as "previously ψ") means that ψ has been previously true.

To precisely define the semantics, we first fix the interpretation domains for terms: we consider an Herbrand universe, U, thus a term algebra for each domain. The semantics define the validity of judgments $(\pi, p) \models \psi$, expressing that the sequence π satisfies property ψ at position p (ranging from 0 to $|\pi| - 1$). Table 2.8 gives the formal definition.

Application: Trace and Audit Logics. To express properties for the execution traces generated from transitions described in the operational semantics, we use the full temporal logic described above, based on the state logic previously described. Events are therefore states. For instance, it is possible to express the property that a message m has been delivered to an agent x.

```
x[−m → true] ∧ X(¬ x[−m → true]).
```

$$
\begin{aligned}
(\pi,p) \vDash \texttt{true} &\overset{\text{def}}{\Leftrightarrow} \texttt{true}\\
(\pi,p) \vDash \texttt{false} &\overset{\text{def}}{\Leftrightarrow} \texttt{false}\\
(\pi,p) \vDash \neg\psi &\overset{\text{def}}{\Leftrightarrow} \neg((\pi,p) \vDash \psi)\\
(\pi,p) \vDash (\psi_1 \vee \psi_2) &\overset{\text{def}}{\Leftrightarrow} ((\pi,p) \vDash \psi_1) \vee ((\pi,p) \vDash \psi_2)\\
(\pi,p) \vDash (\psi_1 \wedge \psi_2) &\overset{\text{def}}{\Leftrightarrow} ((\pi,p) \vDash \psi_1) \wedge ((\pi,p) \vDash \psi_2)\\
(\pi,p) \vDash \varphi &\overset{\text{def}}{\Leftrightarrow} \varphi(\pi(p))\\
(\pi,p) \vDash \exists x.\psi &\overset{\text{def}}{\Leftrightarrow} \exists x \in U.((\pi,p) \vDash \psi)\\
(\pi,p) \vDash \forall x.\psi &\overset{\text{def}}{\Leftrightarrow} \forall x \in U.((\pi,p) \vDash \psi)\\
(\pi,p) \vDash \texttt{X}\psi &\overset{\text{def}}{\Leftrightarrow} ((p+1) < |\pi|) \wedge ((\pi,p+1) \vDash \psi)\\
(\pi,p) \vDash \psi_1\texttt{U}\psi_2 &\overset{\text{def}}{\Leftrightarrow} \exists q.(p \le q < |\pi|) \wedge ((\pi,q) \vDash \psi_2) \wedge (\forall q'.(p \le q' < q) \Rightarrow ((\pi,q') \vDash \psi_1))\\
(\pi,p) \vDash \psi_1\texttt{R}\psi_2 &\overset{\text{def}}{\Leftrightarrow} (\forall q.(p \le q < |\pi|) \Rightarrow ((\pi,q) \vDash \psi_2))\\
&\qquad \vee \exists q.(p \le q < |\pi|) \wedge ((\pi,q) \vDash \psi_1) \wedge (\forall q'.(p \le q' < q) \Rightarrow ((\pi,q') \vDash \psi_2))\\
(\pi,p) \vDash \texttt{G}\psi &\overset{\text{def}}{\Leftrightarrow} \forall q.(p \le q < |\pi|) \Rightarrow ((\pi,q) \vDash \psi)\\
(\pi,p) \vDash \texttt{F}\psi &\overset{\text{def}}{\Leftrightarrow} \exists q.(p \le q < |\pi|) \wedge ((\pi,q) \vDash \psi)\\
(\pi,p) \vDash \texttt{X}^{-1}\psi &\overset{\text{def}}{\Leftrightarrow} ((p-1) \ge 0) \wedge (\pi,p-1) \vDash \psi\\
(\pi,p) \vDash \psi_1\texttt{U}^{-1}\psi_2 &\overset{\text{def}}{\Leftrightarrow} \exists q.(p \ge q \ge 0) \wedge ((\pi,q) \vDash \psi_2) \wedge (\forall q'.(p \ge q' > q) \Rightarrow ((\pi,q') \vDash \psi_1))\\
(\pi,p) \vDash \psi_1\texttt{R}^{-1}\psi_2 &\overset{\text{def}}{\Leftrightarrow} (\forall q.(p \ge q \ge 0) \Rightarrow ((\pi,q) \vDash \psi_2))\\
&\qquad \vee \exists q.(p \ge q \ge 0) \wedge ((\pi,q) \vDash \psi_1) \wedge (\forall q'.(p \ge q' > q) \Rightarrow ((\pi,q') \vDash \psi_2))\\
(\pi,p) \vDash \texttt{G}^{-1}\psi &\overset{\text{def}}{\Leftrightarrow} \forall q.(p \ge q \ge 0) \Rightarrow ((\pi,q) \vDash \psi)\\
(\pi,p) \vDash \texttt{F}^{-1}\psi &\overset{\text{def}}{\Leftrightarrow} \exists q.(p \ge q \ge 0) \wedge ((\pi,q) \vDash \psi)
\end{aligned}
$$

Table 2.8: Temporal Logic: Semantics

We also rely on temporal logic for auditing. To express properties for sequences stored in queues, we use the propositional part of the temporal logic, based on some logic for queue elements. For instance, consider a queue `q` containing a sequence of messages. Over the set of messages, we can define as predicate any subset `M` of messages: `M(m)` is true if `m` belongs to `M`. Now we can express that some message `m` occurs in a log `q` at `x` as follows.

$$\texttt{x[Log(q)} \rightarrow \texttt{(q} \vdash \texttt{F}^{-1}\{\texttt{n}\}\texttt{)].}$$

The predicate $\{\texttt{n}\}$ (a singleton) applied to message `m` tests that `m` belongs to $\{\texttt{n}\}$, in other words $\texttt{n} = \texttt{m}$.

2.3 Validation against Requirements

To validate our approach against the A4Cloud policy requirements, we use a simplified variant of the "Health Care Service in the Cloud" business use case from Deliverable B3.1 [BFO+13]. In brief, medical patients use a wireless networked sensor to send their current health status to a hospital for analysis and diagnosis. The hospital collects data and transfers it to an external actor for processing purpose. Once processing is over, collected data is then transferred back to the hospital to elaborate a diagnosis. We assume that data are sensitive personal data and this is denoted by a specific data attribute: `anonymity = ⊤`.

This use case considers five distinct agents:

- The Data Subject (`S`) is the patient with the wireless networked sensor. The subject sends via the sensor his personal health information to the hospital. In addition, the subject and the legal department of the hospital agree to apply a policy governing the usage of subject's data.
- The Resources (`R`) contain data collected from the patient using the sensor.
- The Data Controller (`dc`) represents the legal department of the hospital. It determines how data processor agents process resource according to the subject expectations.
- Data Processor 1 (`dp1`) represents a medical unit of the hospital that collects sensor data to elaborate a diagnosis for the patient. It is important to note the dp1 acts on behalf of the Data Controller. `dp1` also sends data to the external actor.
- Data Processor 2 (`dp2`) is a a Data Sub-processor which acts on behalf of data processors. `dp2` is hence considered as an external actor that processes data.

In addition to these five agents, the system provides reference monitors. A reference monitor wraps an agent in order to control it.

(R1) Capturing user preferences. To capture user preferences, we use the temporal logic, as shown by the following example. We consider a set of services, like `READ`, `WRITE`, `DELETE`, that the data subject wants to allow, oblige or deny. This set of services is user-defined and can be freely extended. To express an obligation, in the sense that a service should be delivered, we use the future operator `F` of temporal logic. For instance, assume that the data processor `dp` will eventually delete all the personal data associated to subject `S`. We can logically express this obligation as follows.

```
∀ Res : Resource. (Res.subject = S) ∧ (Res.anonymity = ⊤)
  ⇒ ∃ m : Message.
    F ( Res[−m → (m.source = dp) ∧ (m.service = DELETE)]
      ∧ X ( ¬ Res[−m → (m.source = dp) ∧ (m.service = DELETE)] ) )
```

It can be read as follows: for all resource `Res` containing personal data with subject `S`, then it will be the case that a `DELETE` message will be delivered from data processor `dp`. It is probably useful to add a deadline requirement to this obligation. Note that this property supposes that the resource (as an agent) is cooperative, and hence trusted, if we want to ensure that the personal data will be effectively deleted. It is also simple to define a prohibition, for instance, that nobody can update the personal data of subject `S`.

```
∀ Res : Resource.
  (Res.subject = S) ∧ (Res.anonymity = ⊤) ⇒ ∀ m : Message.
    ¬ F ( Res[−m → (m.service = WRITE)] )
```

The meaning of a permission is a shorthand for the policy designer to make explicit what is possible (and implicitly prohibiting everything else), rather than writing a long enumeration of prohibitions. To this purpose, we introduced the `MAY` shorthand in AAL. Assuming that the services are only `READ` and `WRITE`, the policy ”anyone may `READ` personal data of subject `S`” should be completed following the principle that anything that is not permitted is forbidden, in other words: ”nobody can `WRITE` personal data of subject `S`”. Generally speaking, obligations, prohibitions and permissions can be combined by usual logical connectives. Thus an implication is needed to express more complex properties relating some past or future actions. For instance the notification policy ”it is always true that if some actor `WRITE` personal data of subject `S`, then the actor must `NOTIFY` the data subject `S`” can be expressed as follows.

```
∀ A : Actor, Res : Resource.
  (Res.subject = S) ∧ (Res.anonymity = ⊤) ⇒ ∀m : Message.
    G ( (Res[−m → (m.source = A) ∧ (m.service = WRITE)]
        ∧ X (¬ Res[−m → (m.source = A) ∧ (m.service = WRITE)]))
      ⇒ ∃n : Message. F (
        S[−n → (n.source = A)
             ∧ (n.service = NOTIFY) ∧ (n.content = m)] ) )
```

(R2) Anonymity/Pseudonymity. We did not address it here, indeed we make the assumption that all data are personal data. The anonymity problem is far from being trivial to solve, we are still studying a solution. Since this requirement was qualified as optional, we postponed an accurate proposal to our next deliverable.

(R3) Data Minimization. This requirement is linked to the previous one and our future work will propose a consistent solution covering both aspects.

(R4) Strong Policy Binding. At the PPL level this aspect is covered by an explicit notion of sticky policy. In the policy calculus, we did not investigate sufficiently this aspect since it is an optional requirement. However, a data segment encapsulated in a reference monitor can be viewed as an abstract notion of sticky policy. This monitor, glued with the data, is in charge to control the usage of the data, this is mainly the role of a sticky policy.

(R5) Data Retention Periods. A monitor can model a local clock by a counter which is regularly incremented (for instance each time a message is consumed or produced). When a message enters the monitor, its data can be stamped with the current value of the counter. When a message must be exported out of the monitor, if the time stamp of its data is too old (a deadline is reached with respect to a validity period), the monitor does not export the message. In other words: once a delay has passed a data cannot be exported. This can be viewed as if the data were deleted.

(R6) Expressing Regulatory Constraints. To express regulatory constraints is not difficult per se but what is generally lacking is the operational model and the mapping from regulations to the model. For instance, the Data Protection Directive provides that personal data may be processed if the data subject has unambiguously given his consent. The consent given to particular processing operations might relate to a service delivered by a data processor carrying out processing operations on behalf of the data controller. Assuming that the protocol (the sequence of interactions) is the following: the data controller informs the data subject about the data processing, then the data subject accepts the condition and sends back its consent to the data controller and finally the data subject uses the service from the data processor. Thus the following expression denotes the right sequence of actions in AAL style:

```
MAY S.SERVICE[dp](d) ONLYWHEN S.ACCEPT[dc]())
AND MAY S.ACCEPT[dc]() ONLYWHEN dc.INFORM[S]()
```

Its interpretation in temporal logic follows.

```
∀ m : Message. G (
  dp[−m → (m.source = S) ∧ (m.service = SERVICE)]
  ⇒ ∃ n1, n2 : Message.
  P( dc[−n2 → (n2.source = ds) ∧ (n2.service = ACCEPT)]
   ∧ P( ds[−n1 → (n1.source = dc) ∧ (n1.service = INFORM)] ) ) )
```

The modality P stands for the past temporal operator $\mathtt{F}^{-1}$.

(R7) Access Control. Access control policies are logically expressed as invariant properties, expressing that no forbidden access has happened and operationally expressed as checks performed by the reference monitors associated to resources. For instance, given two actors dp1 and dp2, and a resource Res with service READ, an access control policy could be stated as follows: "dp1 may read Res". By completion, applying the principle that all operations that are not allowed are forbidden, the policy is extended with the statement "dp2 cannot read Res". Logically, the policy can be expressed as follows.

```
∀ m : Message. ¬ F (
  Res[−m → (m.source = dp2) ∧ m.service = READ])
```

The formula can be read as follows: for all message m, it is never the case that the reference monitor associated to resource Res deliver a READ message from actor dp2.

Operationally, we can add the following local rule to the reference monitor associated to Res.

```
Res[m → ((m.target = Res) ∧
          (¬ Res[Forbidden(m.source, m.service) → true])) ? −m]
```

The rule can be read as follows: for all message m, transform m (message received) into -m (message to be delivered) if it is not forbidden for m.source to request m.service. Then we initialize the state of the reference monitor with the atomic fact Forbidden(dp2, READ), meaning that it is forbidden for dp2 to call READ.

(R8) Delegation of Rights. In a distributed cloud context, where an end-user's data may be processed by different parties, delegations of rights are needed to ensure a trustworthy data transfer. Delegations will allow an end user to grant some right(s), on some particular resource(s), to one or several identified third-parties. AAL supports only a limited form, that is data distribution, but defining dedicated services allow to express delegation of rights.

The policy calculus can express delegation of rights. For instance, consider the simple example of `Alice`, a Data Subject that shares some data through a resource `Res` with `CompanyA` that is acting as a Data Controller and a data Processor. `CompanyA` may need to outsource a part of its data processing to `CompanyB`. To this end, `CompanyA` must provide to `CompanyB` the necessary credentials to access `Alice`'s resource `Res`. The credential transfer is only possible if `Alice` delegates the right to `CompanyA`.

To express this requirement, we use the always operator `G` of temporal logic, to state that for all message `m` from `CompanyB` requesting a read on `Res`, the same credentials as `CompanyA` must be used. Logically, the policy can be expressed as follows.

```
∀ Res : Resource, m : Message, c : Credentials. G(
  (Res.subject = Alice) ∧ (Res.anonymity = ⊤)
  ∧ Res[−m → (m.source = CompanyB) ∧ (m.service = READ)
             ∧ (m.credentials = c)]
  ⇒ ∃ n1, n2 : Message.
  P( CompanyB[−n2 → (n2.source = CompanyA) ∧ (n2.service = DELEGATE)
                 ∧ (n2.credentials = c)]
     ∧ P( CompanyA[−n1 → (n1.source = Alice) ∧ (n1.service = ALLOW)
                        ∧ (n1.credentials = c)]) ) )
```

(R9) Auditability. Auditability checks the enforcement of security policies. In the health care use case, an audit example is to check whether an external actor processes data. To ensure this policy the system uses log.

From an operational point of view, an agent calls the audit service in order to ensure that expected properties are satisfied. This is the reference monitor of the resource which offers the audit service. It uses the log and state logic to check the property.

(R10) Policy Disclosure. Since it is optional we did not yet cover it.

(R11) Reporting and notifications. In case of policy violation or incidents a notification has to be sent to the data subject and third parties. In the health care use case, a notification towards the patient (`S`), the hospital (`dp1`) and the external actor (`dp2`) is required if the property $Prop$ is not satisfied.

$$\begin{array}{ll}
\texttt{Res}_B\ \texttt{[audit(id) \& Log(q)} \rightarrow \neg\ \texttt{(q} \vdash \mathit{Prop}\texttt{) ?} & \texttt{rep-audit(id, notOK)} \\
 & \texttt{\& notify}_{\texttt{S}}\texttt{(notOK)} \\
 & \texttt{\& notify}_{\texttt{dp1}}\texttt{(notOK)} \\
 & \texttt{\& notify}_{\texttt{dp2}}\texttt{(notOK)]}
\end{array}$$

We assume that the different actors (at least their reference monitors) provide a notification service.

(R12) Redress policies. This requirement is about specifying redress through policies that are providing for remedies within the system. In the health care use case, we could consider the following scenario allowing for redress through policies. When the auditor (called Leslie in the deliverable [BFO+13]) receives a notification related to a violation, then not only can it send a warning (related to a sanction for instance) to the data controller but also it can analyze the cause of the violation and provide a remedy. If the first part can be easily formalized, this is not the case for the second part. That is the reason why we let the example of an analysis of a violation for future work. An example of a simple redress policy can be the revocability of a permission: see the next requirement.

(R13) Revocability. Revocability corresponds to the update of a policy with a new access control policy which is more restrictive. In the health care use case, a revocability example is to forbid the processing of data by the external actor (dp2). For instance, the following local rule for resource $\mathtt{Res}_B$ allows the revocability of a permission for data sub-processor dp2.

```
Res_B [ revokePermission(dp2) & Permission(dp2, READ) →
          true ? Forbidden(dp2, READ) ]
```

(R14) Logging. Logging is useful for auditability. In the health care use case, an audit example is to check that an external actor (dp2) processes data. When the reference monitor delivers a message to the resource, it records information into a queue. In other words, the log is a trace usable by the state logic to express audit policies.

(R15) Controlling data locations and transfer. In this case we need the location information which can be captured as an extra attribute location for agents. Expressing data location constraints becomes rather simple under certain circumstances. For example an actor dp only sends messages inside Europe.

```
∀ dp : Actor, m : Message.
    G ( dp[m → (m.source = dp) ∧ (m.target.location ∈ Europe)] )
```

(R16) Usage Control Rules. An access grant to a resource (see R7) may be packaged with some obligations that have to be guaranteed during and after the access. The main difference with access control is that we have a wider set of actions comprising actions related to the data transmission. The temporal logic operators and also explicit sending actions extend our access control mechanisms to express such obligations.

For instance, we can extend the access control rule presented in R7 ("dp1 may read Res") by a simple usage control rule stating that "dp may read Res only if he has accepted a certain EULA[4]". Logically, the policy can be expressed as follows.

```
∀ Res : Resource, m : Message. G(
  (Res.subject = S) ∧ (Res.anonymity = ⊤)
  ∧ Res[−m → (m.source = dp) ∧ (m.service = READ)]
  ⇒ ∃n : Message. P(
    Res[−n → (n.source = Res) ∧ (n.service = ACCEPT_EULA)] ) )
```

[4]EULA: End-user license agreement.

2.4 AAL Validation, Interpretation and Readability

This section presents a first validation by matching the current AAL expressiveness with the Privacy Level Agreement (PLA) proposal from the Cloud Security Alliance (CSA[5]). Another different validation is the application to a use case as illustrated in Section 4.3. More will be done to validate the AAL approach in the next months with the mapping definition and more use case applications. This section also gives an informal discussion about the AAL interpretation with the temporal logic for accountability. Finally, some hints are given to improve writing, usability and readability of this language for end-users.

2.4.1 Matching with PLA

PLA document issued by CSA enumerates - among other things - the data protection issues that need to be addressed for the sale of cloud services in the EU. The objective is to provide a readable and standard template for cloud providers to define their privacy contracts. The adopted point of view is really detailed but without the perspective to make it machine understandable. Reviewing the sixteen attributes we can claim that most of them can be represented and managed with AAL sentences. However, It is worth mentioning that this approach does not always allow machine processing. For example, the technical measures about data security cannot be compared with a computer system. Another reason is that some attributes contain pure information parts which are not processable by a computer. AAL takes a more abstract point of view, abstracting away from technical and computational details. AAL focuses more on abstract accountability properties the cloud actors are concerned with. We review here the attributes expressed in the PLA document.

1. **Identity**: this is information about the Cloud Service Provider (CSP) identity, which can be represented with some agent attributes.

2. **Categories of personal data**: that the customer is prohibited from sending to or processing in the cloud. It can be expressed with the template: `dataCondition(d) THEN MUSTNOT datasubject.service[CSP](d:Data)`.

3. **Ways in which the data will be processed**: this is expressed by a usage control expression. This point is split in three parts: Data location, subcontractors, and customer software installation. The data location and subcontractors are information which can be represented with attributes in AAL. The last part indicates whether the provision of the service requires the installation of software on the cloud customers system. This needs more design information, either more services on both agent sides or to add specific subcontactors representing the required computation. Despite this additional complexity we see no issue to represent it with AAL.

4. **Data transfer**: with the location information and the adequate services corresponding to data migration AAL is able to express it.

5. **Data security measures**: as explained at the beginning of the section only an abstract view is considered with AAL. Thus, integrity, confidentiality, transparency, isolation can be expressed

[5]`https://cloudsecurityalliance.org/research/pla/`, last access October 30, 2013

as properties. The attributes for auditability and portability are discussed in the next points. The intervenability feature is related to the right to update, delete, access personal data, that is taken into account with AAL. The availability feature concerns the processes and measures in place to manage risks. We could express some of them in an abstract way but the exact boundary should be more precisely studied.

6. **Monitoring**: that is possible in various ways, for example, to write usage control expression needed agreement of the data subject before any processing.
7. **Third-party audits**: we introduced a specific keyword in AAL (`AUDITING`) to catch it.
8. **Personal data breach notification**: as soon as an audit has detected some violations, the `IF_VIOLATED_THEN` keywords allows to specify notifications. More generally usage control expressions are possible in case of specific detections.
9. **Data portability, migration, and transfer-back assistance**: whether, cost, and how concerns can be expressed and automatically processed. But this feature seems also to include an information part which is not computer processable.
10. **Data retention, restitution, and deletion**: we have already take this into account. However, the European data protection framework is currently under review and we should get a clear meaning of these actions. But, we do not see an enforcement problem here.
11. **Accountability**: AAL proposes to use three tiers clauses with usage control, audit action and a posteriori actions.
12. **Cooperation**: with the cloud customer to ensure compliance with applicable data protection provisions. It is basically information about compliance, as soon as we have AAL sentences, a matching tool can help in compliance. There is also protocol information between the customers and the provider/controler, but it can be also described abstractly with AAL.
13. **Law enforcement disclosure**: It is initiated by law enforcement agencies (ie. police, tax authorities) and can be expressed by a usage control expression in AAL.
14. **Remedies**: the a posteriori action set can include remedies as long as they are machine understandable.
15. **Complaint and dispute resolution**: only information, not machine understandable.
16. **CSP insurance policy**: only information, not machine understandable.

To conclude, taking an abstract point of view and only considering machine understandable information we can express the PLA attributes with AAL. That matching overview paves the way to an assistance from PLA to AAL translation.

2.4.2 Relation with the Temporal Logic for Accountability

In this section, we give some hints on how to interpret an AAL sentence as a temporal logic expression. The logic framework we can target can be for instance, a first-order temporal logic, like PrivacyLFP in [DGJ$^+$10], but we think that the quantified propositional temporal logic (QPTL) is better from a decidability point of view [Fis08]. This interpretation is expected to be automated as soon as the full syntax of AAL will be definitive, it should appear in the next deliverable. First, not every expression has a meaning, since the user may introduce various errors. We have an identifier for each agent and each resource. Variable are implicitly universally quantified, for instance, `ANY:Agent` means $\forall$ `ANY:Agent` and so on. We keep the usual and intuitive meaning of the logical operators `AND` and `OR`. Data, in the logical model, are in fact encapsulated in specific monitor and called resources, thus it introduces some extra parameters. For instance, data `d` will be minimally translated as $\forall R.[R.contents = d]$. But more complex expressions are possible to take into account at least the data subject and the privacy status: $\forall R.[R.contents = d, R.subject = bar, R.anonymity = \top]$.

The `MAY` operator corresponds to the prohibition of the actions in the set complement, for instance `x MAY READ d` can be viewed as `x MUSTNOT WRITE d` AND `x MUSTNOT COPY d` etc. In other words "everything not explicitly permitted is prohibited". The `MUST` operator is interpreted as the "eventually" (F) temporal operator, that is the event should occur in the future. The `MUSTNOT` operator denotes a prohibition, that can be expressed as the negation of the F operator or with the dual G.

The `ALWAYS, EVENTUALLY` and `NEVER` are interpreted as the LTL operators $G,\ F$ and $\neg G$, respectively. The operator `THEN` is defined as a synonym for => and `ONLYWHEN` is the combination => P or => F^{-1} . However, rules are may be not so simple in the case of a permission and we are studying this case and we will propose a contextual interpretation.

Logging can be viewed as a particular action. One point is to give a logical meaning to expressions with auditing. But the logic is provided with an operator to check the validity of an expression in a log context ($q \vdash \phi$). The `IF_VIOLATED_THEN` clause is related to an audit (which could be implicitly specified). If the result of the audit is false (or if a property violation has been detected) then do the penalty actions.

This is only to give an informal flavor of the interpretation, particularly to convince the non-expert reader that AAL expressions are close to formal temporal logic sentences.

2.4.3 Improving User-Friendliness

We have sketched a view of the brute AAL language which is not yet user friendly and can be improved to be more readable, and better aligned with the vocabulary of a privacy officer. To improve the obligation writing we could use the following ideas:

- We could use keywords, syntactic sugar, simple rewriting and simplification of some expressions. For instance, `MAY x.READ(d) AND MUST x.WRITE(d) AND MUST x.COPY(d)` is equivalent to an empty action if the set of allowed actions for `x` is {`READ, WRITE, COPY`}.

- We could also provide some rephrasing (as in the style of SecPAL [BFG10]) or to allow the writer to use some natural sentences. For instance in AAL we have `MAY ANY.READ(d)` but it could be written as "anyone may read data d". This remark also applies to many other combinations of actions inside a modality.

- We could also propose help in writing and checking accountability expressions. In particular the translation into a temporal logic expression for accountability can provide a means to check well-formdness and consistency.
- We could define pattern to fill with lists of options as in the wizard approaches found in [PPM11, PT12, FJG$^+$13].
- Mapping the policies to the applications can be also an issue for non specialist in software design. One way could be to define simple analyses or rules helping in the context generation (description of services, agents, obligations, etc), in the pattern choice and filling and finally in the placement of the policies.
- We could also use a rich static type system to express data privacy and cloud roles, to denote data anonymity and also to statically capture in a simple readable manner some legal obligations. We have already some experiences with such a type system [ADG$^+$12, CDR$^+$13].
- One useful tool, not too complex to implement and which could provide some value, can be based on the following scenario:
  1. The privacy officer starts from a regulatory sentence.
  2. He writes the corresponding AAL expression (may be by using some checking tools to simplify it).
  3. He translates the result by using an automatic translator into the formal sentences (in some cases assistance of an expert will be required).
  4. Finally, a parser tool can generate textual sentences from the formal ones which are submitted for validation and comparison to the privacy officer. It could be also possible to generate a simplified and correct AAL formula.
- More advanced natural language processing tools could be used to simplify natural sentences or prepare them to translation in AAL.

2.4.4 Issues and Extensions

Issues regarding the language which are not yet solved:

1. Both in AAL and the semantics there is nothing specific to IaaS/PaaP/SaaS. One hint would be to specify this information as a separate view describing the mapping and effective deployment of the agents into a cloud infrastructure.
2. Expressing delegation of rights, but we do not have yet specific example at the upper level. This is a concept which could appear at the design level (distributed system) thus we may add it in the temporal logic but not in AAL. Indeed one part of the delegation is covered, data disclosures are expressed with usage control expressions.
3. Syntactic constraints may avoid complex or useless operator combinations. We also could suggest some patterns to use and writing assistance to end-users. An important concern is the usage of explicit quantifiers, this is needed to write rigorous clauses but can make more difficult both reading and the verification steps.

Chapter 3

Specifications of the Accountable PPL language

In this chapter, we overview the main limitations of the current PPL language and further describe the proposed extensions in order to map A4Cloud obligations.

3.1 PrimeLife Policy Language (PPL): Advantages and Limitations

As already mentioned in Chapter 1, among all the existing policy languages we reviewed, PPL is the language that captures the best most of the accountability obligations. PPL is an XML-based policy language which combines access control and data handling policies. PPL gives service providers an automatic mean to define and manage privacy policies while applications are enabled to compare these service privacy policies with user privacy preferences. It extends XACML and introduces privacy enhancing features. The language is symmetric because according to the definition in [ABDCDV$^+$09], the same language can be used on the data subject's side to express data handling preferences as on the data controller's side to specify data handling policies and the conditions for access to a resource provided by the data controller. Therefore matching policies with preferences becomes easier. The outcome of the matching procedure generates a sticky policy that is bound to the data and travels with the data. The sticky policy specifies statements on:

- **access control** which is inherited from XACML that PPL extends. In particular PPL enables credential-based access control: a access control policy in PPL may specify the credentials that need to be presented by an access requester to be granted access to a resource. Credentials are defined as lists of attribute statements certified by an trusted third-party.
- **authorizations**, that details actions that the data controller is allowed to perform with respect to the purpose of usage of collected data. In addition, authorizations enable to define the conditions of what in PPL specification [ABDCDV$^+$09] is called *downstream usage*[1]. This kind of authorizations are applicable for any third party collecting the data from the service provider under certain conditions specified in the policies.

[1] Downstream usage specifies under which conditions data can be shared.

- **obligations** that PPL defines as ”a promise made by a Data Controller to a Data Subject in relation to the handling of his/her personal data. The Data Controller is expected to fulfill the promise by executing and/or preventing a specific action after a particular event, e.g. time, and optionally under certain conditions” [ABDCDV$^+$09].

The last two items are part of the data handling preference/policy as shown in Figure 3.1. The vocabulary of PPL is left open and additionally it already presents a user-extensible structure for authorizations and obligations. Figure 3.1 presents the structure of a PPL policy. Note that a PPL (Data Subject) preference follows the same structure by replacing the tag `DataHandlingPolicy` by `DataHandlingPreference` (due to the symmetry of PPL).

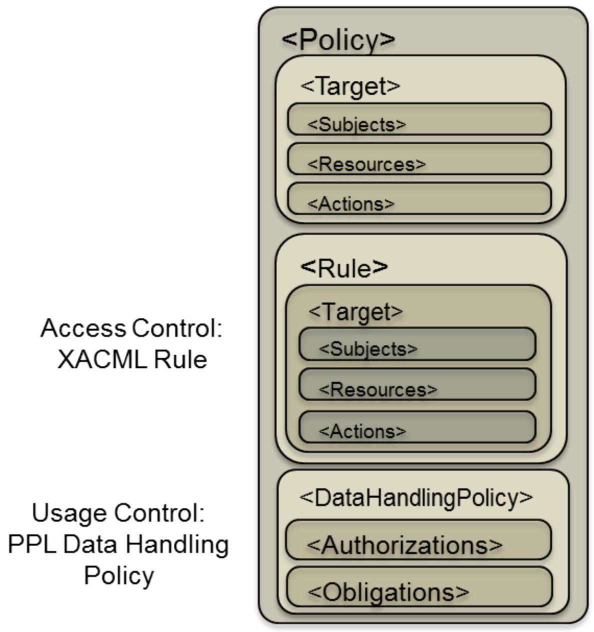

Figure 3.1: General structure of a PPL Policy

In particular, the language provides elements to declare some of the accountability specific obligations such as log or notify. However, these elements need more specification and they may be unpractical when directly used within an accountability policy. For example, in the current version of PPL, the Notify element only allows the Data Controller to notify the Data Subject. In accountability scenarios, notifications are not exclusive to the Data Subject. Instead, notifications may be sent all along the accountability chains to notify the actors within the chain of an occurred event.

As far as logging is concerned, the current specification of PPL allows to declare the action to log, but we cannot specify what information has to be put in the log.

Besides, auditing is not part of the PPL language since the focus of the PrimeLife project was on privacy and not accountability. In the language specification, the way policy violations and security breaches are detected and handled remains unclear. The detection of such violations is out of scope of this document since it focuses on the language itself. The A4Cloud’s policy language aims at providing a way to declare the conditions under which an audit is required, which are not provided in PPL.

An audit may require the provision of evidence to enable the verification of compliance with policies, data subjects' preferences, contracts or regulation. Evidence appears then to be an accountability object of paramount importance.

As audits are not part of PPL, this language fails to capture the concept of an auditor that intrinsically plays a relevant role in an accountable cloud environment. This auditor is responsible of specific tasks, such as requesting evidence or notifying actors in the accountability chain for policy violation.

In the next section, we present an enhanced version of PPL with extensions that address the accountability requirements that we identified in Chapter 1.1. We call this accountability wise policy language **A-PPL**, for Accountable-PPL.

3.2 Extensions

3.2.1 Roles

To address accountability concerns in the cloud environment, it might be necessary to include in the policy a reference to the role of the entity to which the policy is applied to. We suggest to use a standard XACML construct and define the role in the `<Subject>` element of the policy using the `urn:oasis:names:tc:xacml:2.0:example:attribute:role` attribute identifier.

The A4Cloud project identifies several roles and actors in the cloud ecosystem that have to be interpreted in terms of policies. The current version of PPL considers four entities that interact with each other: the Data Subject, the Data Controller, the Downstream Data Controller and the Data Processor.

Data Subject : According to EU Directive 95/46/EC, the Data Subject is an identified or identifiable natural person.

Data Controller : It is defined as the natural or legal person, public authority, agency or any other entity which alone or jointly with others determines the purposes and means of the processing of personal data. The processing may be carried out not only by the data controller but also by the Data Processor performing processing operations on behalf of the Data Controller.

Downstream Data Controller : This term refers to a third-party that receives Data Subject's personal data from the Data Controller and which incurs obligations in respect to the Data Subject.

Data Subjects can also be Data Controllers if they upload data about other data subjects. While the Cloud Provider can be considered as a Data Controller following the A4Cloud taxonomy in MSC-2.2, the other three roles can be mapped to both a Cloud Customer and a Cloud Provider. This is the case for instance of a software as a service provider which processes the data on behalf of the controller, while using the platform of another cloud service provider. For accountability purposes, identifying the Data Controller is required. The problem in PPL is to get information about who the controller is from the policy itself. For the policies written in A-PPL, the Data Controller can be identified thanks to the definition of an attribute for its ID: `dc id`. This attribute can be used to identify the Data Controller for logging or notification purposes for instance.

In addition to the four roles mentioned above, we propose to implement the role of the Auditor that arises from the analysis of accountability concepts within A4Cloud. Note that thanks to the flexibility of XACML, any additional role can easily be implemented if needed.

Auditor : According to the conceptual framework [CFH$^+$13b], the Auditor is trusted third party that can conduct independent assessment of cloud services, information system operations, performance and security of the cloud implementation.

This new role will of course be very useful for accountability specific obligations such as auditability (**R9**), reporting and notification (**R11**) or redress (**R12**).

3.2.2 Actions: Notification, Logging

PPL defines an obligation as Trigger-Action. In the specification of the language, Triggers are events filtered by conditions and related to obligations. These Triggers result in Actions that are executed by the Data Controller according to the Obligation. PPL uses a specific tags for Obligations, Actions and Triggers, see Listing 3.1.

Listing 3.1: Obligation structure in PPL

```
<Obligation>
  <Trigger> ... </Trigger>
  <Action> ... </Action>
</Obligation>
```

The current specification of PPL defines, among others, two Actions: Notify and Log. They cover only partially the obligations we analyzed in the A4Cloud project. For instance, the notification action only enables the notification of the data subject. Therefore we extend them to address some identified shortcomings.

Notification :

The current Action element that enables the notification in PPL is called `ActionNotifyDataSubject`. As its name implies this element specifies the Data Controller to notify the Data Subject. Our language provides a more fine-grained notify action in order to capture the accountability concepts. In particular, the new notify action, `ActionNotify`, is not reserved for notifications to the Data Subject only. Instead, we provide a parameter `recipient` aimed at indicating the recipient of the notification that are all along the accountability chain (be it a Cloud Customer, a Cloud Provider, an Auditor, or the Data Subject). In addition, within the A4Cloud project, notifications can hold several types, such as notifications used to inform about a policy violation, about identified risks, about policy updates, about redress mechanisms, about warning of possibilities for sanctions (**R12**), and about evidence collection. In this perspective, the `ActionNotify` presents another parameter `type` that specifies the type of notification to be sent to the recipient. Table 3.1 describes the notify action.

Logging :

Name	**ActionNotify**		
Description	This action notifies a cloud actor when triggered		
Parameters	Media	media	The media used to notify the user (e-mail, SMS, etc.)
	Address	address	The corresponding address (e-mail address, phone number, etc.)
	Recipient	recipient	The identity of the recipient of the notification
	Notification type	type	The type of notification(policy violation, evidence, redress, etc.)
Examples	Notify the Data Subject a policy violation $\rightarrow$ ActionNotify(e-mail, @,ID, policy violation)		
	Notify Data Controller a sanction $\rightarrow$ ActionNotify(e-mail, @,ID, sanction)		

Table 3.1: Action Notify

Logging is an essential feature for accountability (**R14**). In terms of policies, the Logging action should provide a way to specify not only the details of the event that is logged but also the security properties of the logs (integrity or confidentiality of the logs for instance). Currently, PPL defines an `ActionLog` action. However, this action element fails to capture the concept detailed in requirement **R14**. Therefore, we propose extending `ActionLog` in such a way that it complies with **R14**. In particular, the new `ActionLog` introduces several parameters to make explicit what information on an event needs to be logged. A timestamp is required to log the time of the event. Then the policy must require to log the action (e.g. `SEND`), the identity of the subject who performed the action (e.g. `Cloud x`) and the purpose of the action (e.g. `marketing`). In a data-centric logging perspective, that is in order to trace events based on data, the policy must require the identifier of the data. Other details must also be written in the logs such as some security flags that may state whether the log is encrypted or that specifies that its integrity can be checked. Table 3.2 describes the Log action.

3.2.3 Triggers

This section describes the triggers that the A4Cloud project design for accountability. We recall that in the PPL vocabulary, a trigger is an event filtered by a condition that triggers an action. The current version of PPL [TNR11] already presents a set of triggers that are relevant for accountability such as `TriggerOnPolicyViolation`, `TriggerAtTime` or `TriggerPersonalDataSent`. The proposed set of triggers is extensible and we define a collection of new triggers that catch the accountability obligations. In particular, we identified two triggers that specify events that generate some accountability actions related to evidence collection. As it will be shown in Section 3.2.4, the auditing protocol requires two triggers: one when an evidence request is received by an Auditee and another one when the Auditor receives the evidence. As a consequence, we create the two following triggers: `TriggerOnEvidenceRequestReceived` and `TriggerOnEvidenceReceived`.

Name	**ActionLog**		
Description	This action logs an event based on the details provided within the policy		
Parameters	Timestamp	timestamp	The time of occurrence of the logged event
	Action	action	The action that is logged
	Purpose	purpose	The purpose of the action that is logged
	Subject ID	subject	The identity of the subject that performed the action
	Resource ID	resource	The identifier of the resource the action was made on
	Resource location	location	The location of the resource
	Security Flag	flag	The flag is 1 if the log is confidential, 2 for integrity check, 3 for both
Examples	Log the forwarding of data $\rightarrow$ ActionLog(t, send, marketing, cloud x, record, Germany, 1)		

Table 3.2: Action Log

Similarly, when an update occurs in a policy, the update may incur a set of actions to be performed once the policy is changed. When the user preference is updated by the Data Subject, for example, he or she removes consent on a previously agreed access policy on a piece of data, this may incur the Data Controller to revoke granted access to the data subject's data. If the Data Subject set weaker preferences, this also may incur some actions to be undertaken by the Data Controller. We call this trigger `TriggerOnPreferencesUpdate`. Note that the preferences are considered as personal data cannot be shared with anyone. When the policy is updated by the Data Controller, for instance, it changes its policies, this may require the Data Controller to notify the Data Subject about the update, in order to get informed consent from him/her. In addition the notification provides the Data Subject with a proof of update. We call this trigger `TriggerOnPolicyUpdate`.

Remediability which is considered as a key accountability attribute can be achieved through policies. In particular, a Data Subject can file a complaint regarding the processing of his or her personal data carried out either by a Data Controller or by a Data Processor as provided within, for instance, the company's policy. To handle such complaints, we create a new Trigger, `TriggerOnComplaint` that initiates obligations on the Data Controller, Processor or Auditor side. For example, when an Auditor receives a user complaint, it can send back to the user a notification of receipt of the complaint.

3.2.4 Evidence

Evidences play an important role in accountability. That is the reason why there is a need to create an Evidence object in A-PPL that does not exist in PPL. Evidence are key in auditing processes. The A4Cloud framework identified two types of evidence. The identification of evidence type is indeed a distinct and specific task within the A4Cloud project. From the time of writing this document, two types of evidence have been pointed out: logs and cryptographic proofs such as proofs of retrievability.

We define a new A-PPL action, `ActionAudit`, that is presented in Table 3.3. It enables an actor to initiate an audit protocol to verify compliance of the infrastructure and services with policies and regulations. We envision this protocol as an exchange of XACML Requests and Responses. Since XACML is dedicated to access control, we extend the XACML request-response protocol to audit and evidence gathering. The audit protocol involves two parties: an Auditor (be it a Regulator or a Cloud Customer) and an Auditee (e.g. a Data Controller or a Data Processor).

Name	**ActionAudit**		
Description	This action creates an evidence request		
Parameters	Evidence	evidence type	The type of evidence to generate (logs, crypto proofs, etc)
	Resource	resource id	The ID of the resource the evidence is based on
	Subject	subject id	The ID of the data subject the evidence is based on
	Recipient	recipient id	The ID of the recipient of the evidence request
Examples	Audit the hospital for possession of Kim's medical records → ActionAudit(proofs of retrievability, medical record, Kim, hospital)		

Table 3.3: Action Audit

We identify three types of actions within the audit protocol between these two parties:

Evidence Request : This phase allows the Auditor to request Evidence from an Auditee based on a resource, such as auditing the retrievability of a piece of data or auditing the location of that data, etc. The Evidence request prepared by the Auditor takes the form of an XACML Request where the fields are: the `Type` that specifies the type of evidence requested (such as logs) and `Resource` that specifies the identifier of the resource that is audited. Listing 3.2 shows an example of such an evidence request. In addition to this XACML Evidence Request, the Auditor may send a challenge to the Auditee. For instance, protocols for proofs of retrievability require the auditor to send a cryptographic challenge to obtain the proofs that are verified against this challenge.

Listing 3.2: "XACML Request for Evidence"

```
<Request>
  <Subject>
    <Attribute AttributeId="subject-id" DataType="string">
      <AttributeValue>Kim</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="resource-id" DataType="string">
```

```
        <AttributeValue>medical record</AttributeValue>
      </Attribute>
    </Resource>
    <Evidence>
      <Attribute AttributeId="evidence-type" DataType="string">
         <AttributeValue>proofs of retrievability</AttributeValue>
      </Attribute>
   </Evidence>
  <Recipient>
      <Attribute AttributeId="recipient-id" DataType="string">
         <AttributeValue>hospital</AttributeValue>
      </Attribute>
   </Recipient>
</Request>
```

Evidence Collection : Based on the XACML Evidence request (and possibly the challenge), the Auditee generates the requested evidence. This step consists in gathering the requested evidence: collecting the logs or computing cryptographic proofs. We introduce a new PPL obligation, namely `ActionEvidenceCollection` that based on the Trigger `TriggerOnEvidenceRequestReceived` initiates the Evidence Generation. Table 3.4 describes the Evidence Collection action. Once collected the evidence are sent to the Auditor. In case of logs, the response can take the form of an XACML Response giving access to the requested logs. The XACML Response structure has two attributes: `Decision` that specifies if the access to log is granted or denied and `Obligations` that specifies obligations to adhere to such as delete the evidence after analysis.

Name	**ActionEvidenceCollection**		
Description	This action collects the requested evidence		
Parameters	Evidence	evidence type	The type of evidence to generate (logs, crypto proofs, etc)
	Resource	resource id	The ID of the resource the evidence is based on
	Subject	subject id	The ID of the data subject the evidence is based on
	Recipient	recipient id	The ID of the recipient of the evidence (the Auditor)
Examples	Collect proofs of retrievability for Kim's medical record → ActionEvidenceCollection(proofs of retrievability, medical record, Kim, Auditor)		

Table 3.4: Action Evidence Collection

Evidence Analysis/Verification : The Auditor receives the Evidence and checks whether the Auditee's actions are compliant with policies. This step may require an additional obligation to be enforced by the Auditor such as notify the data subject or the data protection authorities. These actions are trigger with the specific Trigger that we defined in the project: `TriggerOnEvidenceReceived`.

3.2.5 Purpose with Duration

In A-PPL, any declaration of a `Purpose` element can optionally be followed by the definition of an attribute `duration` such as: `<Purpose duration=2Y>admin</Purpose>`. TFor instance, a particular piece of data is used for research purposes for 2 years but has to be kept for legal purposes for 5 years. PPL cannot express this, therefore we define this `duration` attribute for `Purpose`. In addition, this attribute implies that when all durations for each purpose has expired, the data has to be deleted, since the data cannot be used for any purpose anymore.

3.2.6 Attributes Identifiers for Subjects

In XACML, Subject attributes can be separated into multiple categories. In order to correctly determine attributes, policies create `SubjectAttributeDesignators` to specify in which category to consider when searching for a given attribute. Attribute selection is finally achieved via a XPath query, containing an expression and its type, embedded in an `AttributeSelector` to search for a given pair of attribute and value. Note that results can be a set of values. In order to disambiguate among multiple results, the policy author can make a use of `Bag` construct to make unordered collections of values and to process them using functions, for instance to compare with expected values. In the case of our extension to PPL, we can leverage on the existing mechanisms from XACML to handle attribute identifiers for Subjects, in particular, credentials, such as Username/password, SAML assertions, or other authentication tokens.

3.2.7 Overview of the Policy Engine

We give here an overview of the extension of the Policy Engine of PPL to enable the capabilities of A-PPL. This Policy Engine will be in charge of interpreting and matching the A-PPL policy and preferences. The result of the matching generates an accountable sticky policy. The components of the Engine will be similar to the ones of the XACML and PPL frameworks:

Policy Enforcement Point. It formats and forwards the messages between components, such as the XACML requests and responses.

Policy Decision Point. This element is the core of the engine: it performs the matching between preferences of the data subject and the data controller policies. It also makes the decision of the access control requests.

Obligation Handler. This engine handles the triggers and actions specified in the policy.

Event Handler. It monitors the events that are related to the resources stored in the engine and reports them to the Obligation engine to check if some Triggers relate to them.

Logging Handler. It allows auditing the actions taken by the engine.

We aim at extending this framework in order to enforce the accountability specific actions. We may update our language specification according to the design of our policy engine.

3.2.8 Summary of the extended A-PPL

This chapter presented our accountable policy language, A-PPL, that extends PPL in a such way that A-PPL maps the accountability obligations.. We defined the Auditor as a new role in A-PPL. We extended the lists of available PPL actions to catch the accountability actions of logging, notification, audit and evidence collection. Table 3.5 shows the list of available actions. The asterisk indicates the A-PPL extensions. We also extended the list of available PPL triggers for accountability. They are related to evidence, updates on policies or preferences and to the reception of a complaint from the data subject. Table 3.6 shows the list of available triggers. The asterisk indicates the A-PPL extensions.

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

Table 3.5: List of Actions

As evidence are key for accountability we added them in A-PPL. In particular, we defined using actions, triggers and XACML request/response protocol an audit protocol for accountability. We finally gave an overview of the policy engine that is further described in the D43.1 deliverable about the policy engine architectural design.

Name	**Description**
TriggerAtTime	Time-based trigger that occurs only once
TriggerPeriodic	Time-based trigger that occurs periodically
TriggerPersonalDataAccessedForPurpose	Occurs each time the personal data is accessed for a specified purpose
TriggerPersonalDataDeleted	Occurs each time the personal data is deleted
TriggerPersonalDataSent	Occurs each time the personal data is sent
TriggerDataSubjectAccess	Occurs when the Data Subject accesses its own personal data collected by the Data Controller
TriggerDataLost	Occurs when the personal data is lost
TriggerOnPolicyViolation	Occurs when a policy violation is detected
TriggerOnEvidenceRequestReceived*	Occurs when an Auditee received an evidence request
TriggerOnEvidenceReceived*	Occurs when an Auditor received evidence
TriggerOnPolicyUpdate*	Occurs when the Data Controller updates its policies
TriggerOnPreferencesUpdate*	Occurs when the Data Subject updates its preferences
TriggerOnComplaint*	Occurs when the Data Subject files a complaint

Table 3.6: List of Triggers

Chapter 4

Application Examples

In this section we validate the accountability policy representation framework by modeling one of the A4Cloud use cases documented in the A4Cloud public deliverable DB3.1 [BFO$^+$13] and translating the regulations, contracts and privacy policies they must comply with into the policy language.

4.1 Use Case: Health Care Services in the Cloud

The use case that we have analysed in this deliverable concerns the flow of health care information generated by medical sensors, which is sent to and processed in the cloud. The system that is described in the A4Cloud deliverable DB3.1 and illustrated in Figure 4.1, 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.

In this use case the patients will be the data subjects from whom personal data will be collected. The hospital is ultimately responsible for the health care services and since they determine the purpose and the means of processing, they are considered as data controllers with respect to the processing of patients' personal data. In this case the relatives may upload personal data about the patients and can therefore also be seen as data controllers. As can be seen in Figure 4.1, the use case will involve cloud services for sensor data collection and processing, cloud services for data storage and cloud services for information sharing, which will be operated by a collaboration of different providers. Since the primary service provider (the MedNet platform provider), with whom the cloud users will interface, employs two sub-providers, a chain of service delivery will be created. In this particular case, the MedNet platform provider will be the primary service provider and will act as a processor under the instructions of the data controller with respect to the personal data collected from the patients. In addition, the providers of Cloud x and Cloud y will be sub-processors w.r.t. the patients' personal data[1]. The details of the use case is further described in DB3.1 [BFO$^+$13].

[1] In this deliverable the term *sub-processor* is used to mean an entity engaged by the processor to assist it in performing its processing operations assigned by the Data Controller.

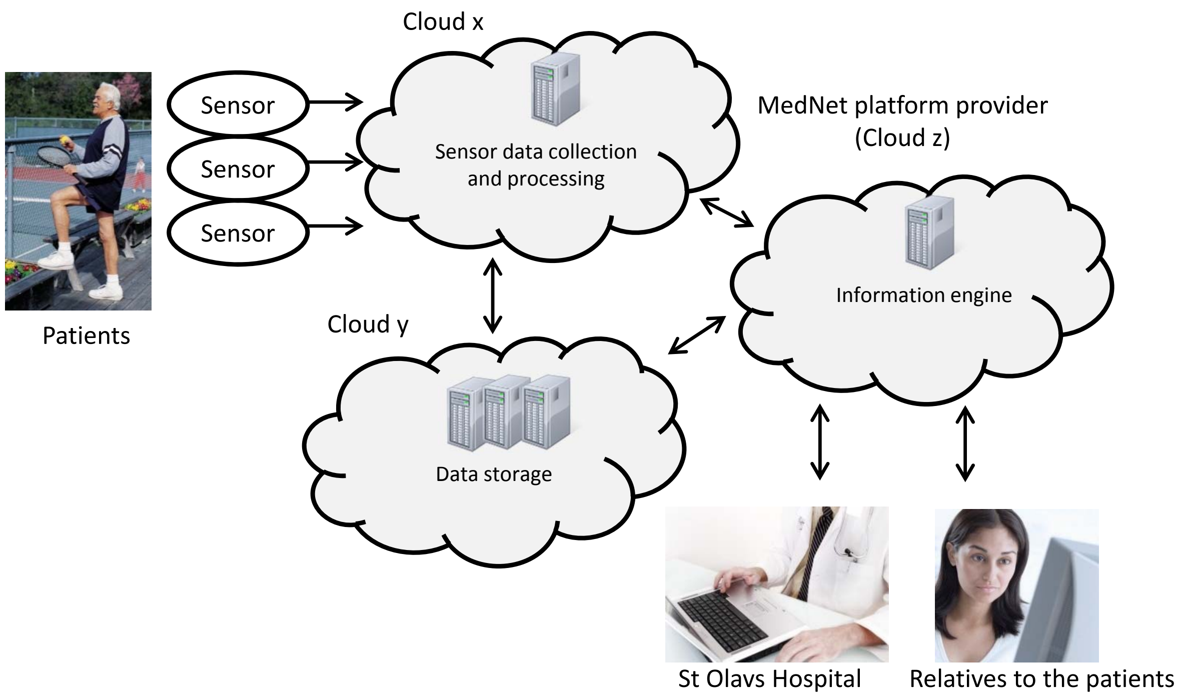
Cloud x
Sensor
Sensor
Sensor
Sensor data collection and processing
MedNet platform provider (Cloud z)
Information engine
Patients
Cloud y
Data storage
St Olavs Hospital
Relatives to the patients

Figure 4.1: An overview over the main actors involved in use case 1

4.2 Obligations for the Use Case

We have identified a number of obligations for the use case, which needs to be handled in the accountability policy framework. Here we list the most important obligations[2]. The list of obligations has been extracted from the list of accountability relationships for use case 1, which is documented in deliverable DB3.1 [BFO+13].

O1 As data subjects, the patients have the right to access, correct and/or delete their personal data.

O2 As a data controller, the hospital needs to provide a policy on what data is collected and for what purposes.

O3 As a data controller, the hospital must ask the data subjects (ie. the patients) explicit consent for collecting and processing personal data.

O4 As joint data controllers, the relatives/friends must ask the data subjects (ie. the patients) explicit consent for collecting and processing personal data.

O5 As a data controller, the hospital must only use personal data for the specified purposes only.

[2]We do not provide the complete list of obligations since it contains many duplicates. For example, all of the involved service providers (the MedNet platform provider, Cloud provider x and Cloud provider y) are responsible for informing the affected parties about any security and/or personal data breaches. We consider it unnecessary to depict all these obligations since they will be represented by almost identical expressions in AAL and A-PPL.

O6 As a data controller, the hospital must, upon request, provide evidence to the data subjects (ie. the patients) on their personal data processing practices.

O7 As a data controller, the hospital must notify the processing of personal data to the competent data protection authority.

O8 As a data controller, the hospital must inform the data subjects (i.e. the patients) of security or personal data breaches.

O9 As data controllers, the relatives must only use personal data for the specified purposes only.

O10 As a data processor, the MedNet platform provider must log all access to personal data.

O11 The data processor (i.e. the MedNet platform provider) must inform the data controller (i.e. the hospital) about the engagement of sub-processors.

O12 As a data processor, the MedNet platform provider must, upon request, provide evidence to the data controller (ie. the hospital) on their personal data processing practices.

O13 As a data processor, the MedNet platform provider must, upon request, provide evidence to the data controller (ie. the hospital) on the correct and timely deletion of personal data.

O14 As a data processor, the MedNet platform provider must notify the data controller (i.e. the hospital) of security or personal data breaches.

O15 As data processors, the MedNet platform provider, the provider of Cloud x and the provider of Cloud y must facilitate for regulators to review evidence of the processing of personal data whenever requested.

The health care use case represents a snapshot of what the A4Cloud conceptual framework is intended to cover. As a result, most of the identified obligations for the health care use case are instances of the requirements for the policy language design that we identified in Chapter 1.1. For example, **O1** and **O2** are examples of **R7** (access control rules), **O3** and **O4** are examples of **R1** (capturing privacy preferences), **O5**, **O6**, **O9**, **O10**, **O12**, **O13** and **O15** are examples of **R9** and **R14** and **O8** and **O14** are examples of **R11** (reporting and notification).

4.3 AAL Obligations for Use Case 1

The method used here to describe usage of AAL to the health care use case assumes several important hypotheses. These assumptions are needed in order to have sufficient enough details to go into an operational, yet abstract description.

- First, we assume here an abstract design of the system described with components and services in a UML V2.0 style.
- Second, original obligations, as described in the previous section, are sometimes rewritten to map the design.

Abstract Component Design

The component diagram (Figure 4.2) describes a part of the health care actors and their interactions. We have the data subject `Kim` and his relative `Sandra`. `Sandra` is a joint data controller. There are three cloud providers (named `cloudX`, `cloudY` and `cloudZ`). The data controller is the `hospital` and `Leslie` is the auditor. These entities are represented as components in the diagram offering and using some services. We consider only one way services to make more explicit the control and the communications. Note that the `AccesRigthInterface` is implemented by the three cloud providers and used by the two data subjects. This specific design can be improved, its purpose is mainly to make more concrete the AAL expressions we expect to write.

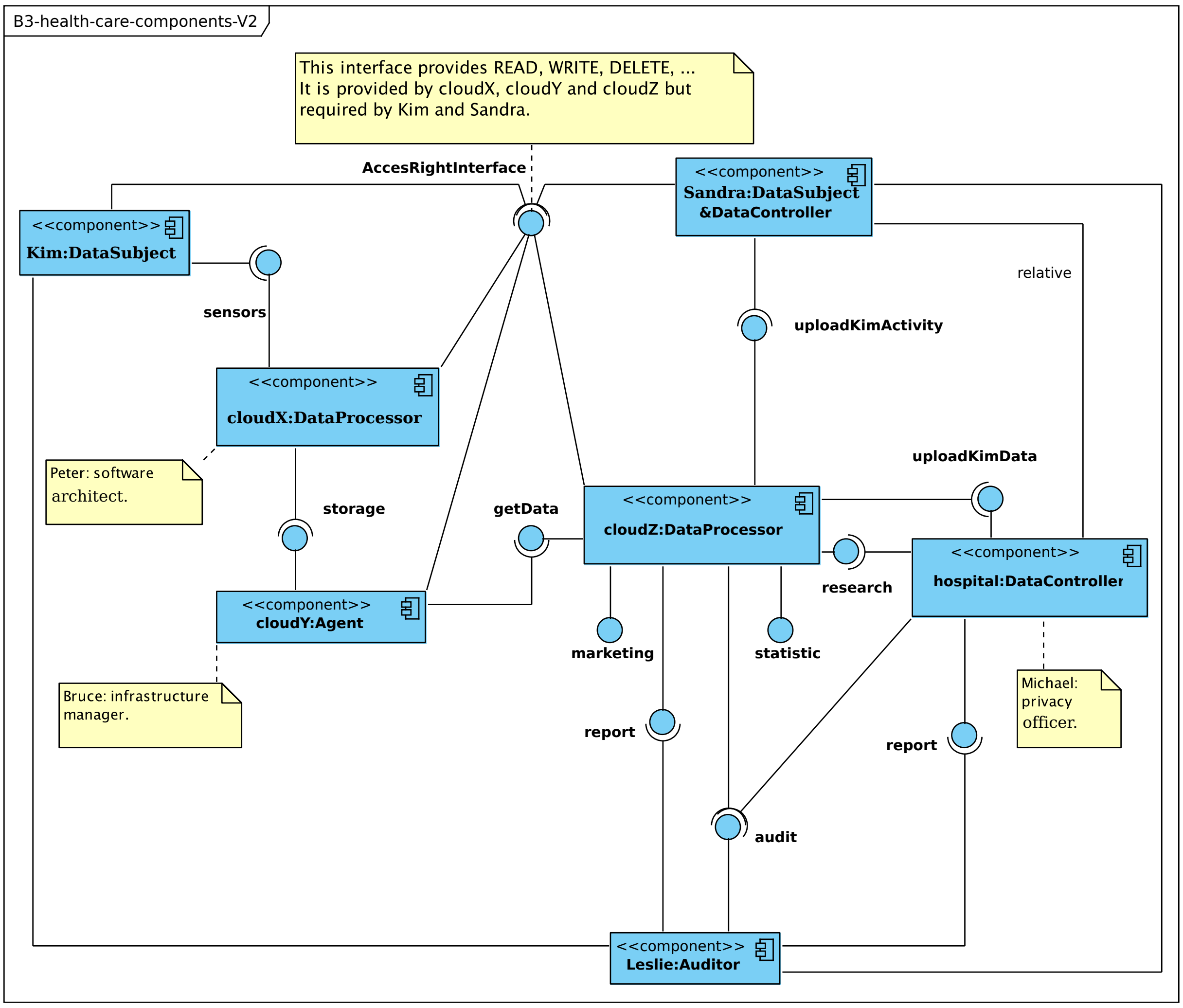
B3-health-care-components-V2
This interface provides READ, WRITE, DELETE, ...
It is provided by cloudX, cloudY and cloudZ but
required by Kim and Sandra.
AccesRightInterface
<<component>> Sandra:DataSubject &DataController
<<component>> Kim:DataSubject
sensors
relative
uploadKimActivity
<<component>> cloudX:DataProcessor
Peter: software architect.
uploadKimData
storage
getData
<<component>> cloudZ:DataProcessor
<<component>> hospital:DataController
research
<<component>> cloudY:Agent
marketing
statistic
Michael: privacy officer.
Bruce: infrastructure manager.
report
report
audit
<<component>> Leslie:Auditor

Figure 4.2: Component diagram for the health care case study.

Expressing the Obligations in AAL

This section reviews the obligations of the previous section and explains how they are taken into account.

O1: As data subjects, the patients have the right to access, correct and/or delete their personal data. We have an interface (`AccessRightInterface`) to provide access for data subjects.

O2: As a data controller, the hospital needs to provide a policy on what data is collected and for which purposes. We have some elements in O3/O4 about the abstract obligation. We need, for the data controller, an expression like:

```
MAY hospital.PROCESS(D:Data) AND ((ANY:Agent in hospital.relatives[D.subject])
    THEN MAY ANY.PROCESS(D:Data))
```

The exact processing action is not described but it can be easily instantiated. Not that this action encodes also the “purpose” textual information familiar in other privacy languages. From now on, to simplify we consider that the purpose information is implicitly associated with actions or services.

O3: As a data controller, the hospital must ask the data subjects (the patients) explicit consent for collecting and processing personal data.

O4: As data controllers, the relatives/friends must ask the data subjects (the patients) explicit consent for collecting and processing personal data. These two obligations are about informed consent. In AAL this is a similar situation as in O1: a static enforcement by interfaces and full accountability with a general obligation taking into account the explicit consent. We need to know component interfaces and interactions, see Figure 4.3.

But more ofen we want to express explicit audit and rectification, for instance: ”if a data controller processes personal data he must have received the informed consent from the data subject else in case of an audit the violation leads to sanctions by the auditor”. Note that in case of sensitive personal data (i.e health data), then the consent must be explicit.

```
ALWAYS
 (MAY cloudX.sensors(D:Data) ONLYWHEN D.subject.informedConsent[hospital]()
  AUDITING Leslie.AUDIT(hospital.logs)
  IF_VIOLATED_THEN MUST Leslie.SANCTION(hospital)
```

Note that our AAL expression only considers the action of `cloudX` but the real policy could include any other actions involving the `D` data, for instance transmission to `cloudY` and `cloudZ`. Strictly speaking `MAY cloudX.sensors(D:Data)` says nothing about the nature of the data. But `D.subject=Kim THEN MAY cloudX.sensors(D:Data)` expresses a notion of personal data for Kim. The ”informed consent” means that the hospital has previously sent its policy (see O2) to Kim before. Thus we need to add the AAL expression below as an additional condition in the usage expression:

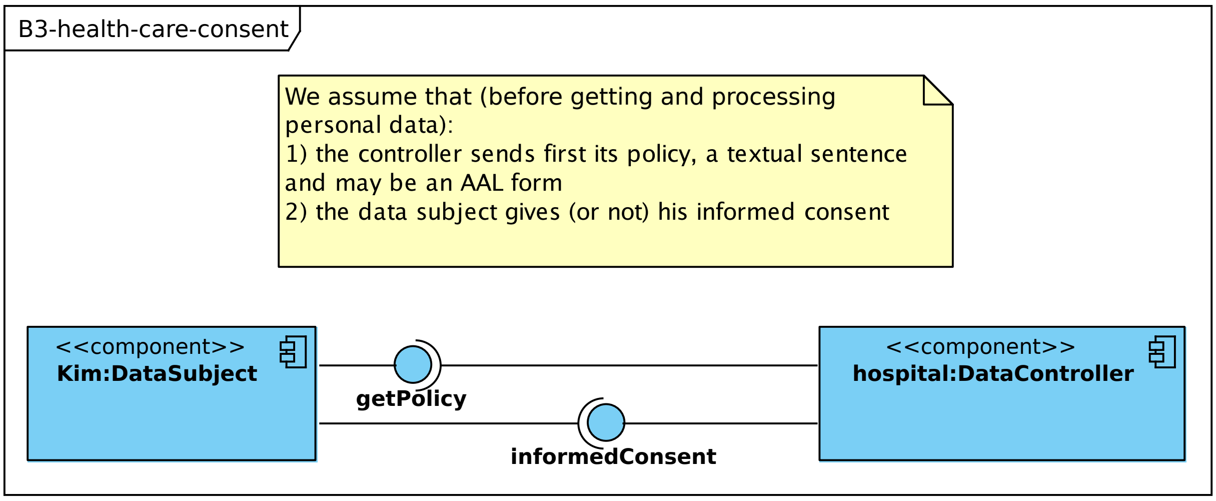
B3-health-care-consent
We assume that (before getting and processing personal data):
1) the controller sends first its policy, a textual sentence and may be an AAL form
2) the data subject gives (or not) his informed consent
<<component>>
Kim:DataSubject
<<component>>
hospital:DataController
getPolicy
informedConsent

Figure 4.3: Interactions for informed consent.

```
 AND (MAY D.subject.informedConsent[hospital]()
        ONLYWHEN hospital.getPolicy[D.subject]("processing policy and purpose"))
```

A related expression with the data subject Kim is: "if data subject Kim uses the sensors service he must have sent before his informed consent to the data controller else in case of an audit the violation leads to sanctions imposed on the data controller by the auditor".

```
ALWAYS
 (MAY Kim.sensors[cloudX](D:Data) ONLYWHEN Kim.informedConsent[hospital]()
  AUDITING Leslie.AUDIT(hospital.logs)
  IF_VIOLATED_THEN MUST Leslie.SANCTION(hospital)
```

The difference between both expressions is related to the point of view the specifier. In the first case, the specifier considers `cloudX` processing some data. This leads to a general expression without an explicit data subject and close to the real obligation. In the second case, we adopt a point of view centered on Kim, a particular data subject.

O5: As a data controller, the hospital must only use personal data for the specified purposes only. This obligation gives more details on the hospital policy.

```
(personal(ANY:Agent, D:Data) THEN MAY hospital.PROCESS(D))
   AND ((ANY in hospital.relatives[D.subject]) THEN MAY ANY.PROCESS(D))
```

We add a predicate `personal(ANY:Agent, D:Data)` to denote a personal data.

O6: As a data controller, the hospital must, upon request, provide evidence to the data subjects (the patients) on their personal data processing practices. Assume that the hospital and its processors securely log their processing actions. Managing evidence and logs will be done by some specific tools, not part of AAL or A-PPL, and that provide assurance about what is logged. There are two kinds of actions, for some agents to use an external service or to do some internal action. In our example, an instance of O6, is: ”if cloudX gets personal data from Kim or if Sandra uploads Kim’s activity to cloudZ, then the controller must ensure the log of these actions”.

```
ALWAYS
  (MAY Kim.sensors[cloudX](D:Data) OR
   MAY Sandra.uploadKimActivity[cloudZ](D1:Data))
     THEN (MUST hospital.LOGS(Kim, sensors[cloudX], D, timestamp)
          AND MUST hospital.LOGS(Sandra,
                                uploadKimActivity[cloudZ], D1, timestamp))
```

Thus the hospital can now provide these logs to the auditor and to the data subjects.

O7: As a data controller, the hospital must inform the processing of personal data to the competent Data Protection Authority. We assume that it occurs once, for instance when defining a new service.

```
MAY hospital.NEWPROCESSINGSERVICE()
     THEN MUST hospital.NOTIFY(hospital.dataauthority, "...")
```

O8: As a data controller, the hospital must notify the data subjects (i.e. the patients) of security or personal data breaches. This obligation is related to a violation and its AAL representation is related to the party who is triggering the violation: the processors, the controllers? But in any case the AAL expression looks like:

```
MAY hospital.VIOLATEPOLICY()
    THEN (MUST hospital.NOTIFY[Kim]("...")
         AND ((ANY:Agent in hospital.relatives[Kim])
                  THEN MUST hospital.NOTIFY[ANY]("...")))
```

O9: As joint data controllers, the relatives must only use personal data for the specified purposes only. The audit should/could detect the purpose usage violation.

```
(ANY:Agent in hospital.relatives) THEN MAY ANY.USEDATA(D:Data)
IF_VIOLATED_THEN ...
```

`USEDATA` should be a more precise usage expression. Furthermore, we should qualify the data, for instance adding the condition `D.subject=Kim AND ANY:Agent in hospital.relatives[Kim]`.

O10: As a data processor, the MedNet platform provider must log all access to personal data. The current diagram represents `cloudZ` but does not include the full details of the `MedNet` platform, we assume here a representation of the `MedNet` agent with an attribute to know its providers. A general template for the policy with explicit logging is below.

```
((OTHER:Agent in MedNet.providers) THEN MAY  ANY:Agent.ACCESS[OTHER](D:Data))
    THEN MUST MedNet.LOGS(ANY, ACCESS[OTHER], D, timestamp)
```

`ACCESS` here is any action to access data, thus should be instantiated with the explicit services allowing an access to `D`.

O11: The data processor (i.e. the MedNet platform provider) must inform the data controller (i.e. the hospital) about the engagement of sub-processors. As soon as ”MedNet.chainofproviders” is known, the platform must notify the controller with this information.

```
MUST MedNet.NOTIFY[hospital](MedNet.chainofproviders)
```

The chain of providers is an attribute of the MedNet platform.

O12: As a data processor, the MedNet platform provider must, upon request, provide evidence to the data controller (ie. the hospital) on their personal data processing practices. It could look like O6, MedNet is using (secure or not) logs and can send them to the data controller.

O13: As a data processor, the MedNet platform provider must, upon request, provide evidence to the data controller (ie. the hospital) on the correct and timely deletion of personal data. The previous O12 is a more general case. But we could express that deleting actions are logged:

```
MAY MedNet.DELETE(D:Data) THEN MUST MedNet.LOGS("deleted", D, currentDate)
```

O14: As a data processor, the MedNet platform provider must notify the data controller (i.e. the hospital) of security or personal data breaches. Similar to O8 with different agents.

O15: As data processors, the MedNet platform provider, the provider of Cloud x and the provider of Cloud y must facilitate for regulators to review evidence of the processing of personal data whenever requested. The problem here is what the exact meaning of “must facilitate”. We can express that processors must log their actions and these logs are provided to an auditor. An evidence tool can help in browsing the logs to find specific information, but it is out of the scope of AAL.

4.4 Expressing the Obligations in A-PPL

In this section we explain how the obligations in the Section 4.2 can be expressed and implemented using A-PPL. First we start mapping which of the A-PPL roles that are applicable in the business use case 1. The star is used to indicate that a particular role is an extension (created in the A4Cloud project) to PPL.

- Patients can be assigned the A-PPL/PPL role **Data Subject**.

- The hospital can be assigned the A-PPL/PPL role **Data Controller**. Also the patients' relatives can be assigned this role when they upload personal data about the patients (they act as joint data controllers).
- The MedNet platform provider, Cloud provider x and Cloud provider y can all be assigned the A-PPL/PPL role **Downstream Data Controller**.
- The Norwegian Data Protection Authority can be assigned the A-PPL role **Auditor**.

In the next step we outline how to use A-PPL to implement the obligations.

O1: As data subjects, the patients have the right to access, correct and/or delete their personal data. Read and write access control is achieved through XACML rules. On the data controller's side, the policy in Listing 4.1 permits read and write access to the data subjects.

Listing 4.1: Read and Write Access granted to the Data Subject

```
<Rule  RuleID="write" Effect="Permit">
   <Target>
      <Subject>
         <SubjectMatch MatchId="string-equal">
         <AttributeValue DataType="string">Patient:Kim</AttributeValue>
         <SubjectAttributeDesignator DataType="string
                 AttributeId="subject-id"/>
         </SubjectMatch>
      </Subject>
      <Resource>
         <ResourceMatch MatchId="string-equal">
         <AttributeValue DataType="string">Patient:Kim </AttributeValue>
         <ResourceAttributeDesignator DataType="string"
                 AttributeId="resource-owner"/>
         </ResourceMatch>
      </Resource>
      <Actions>
         <Action>
            <ActionMatch MatchId="string-equal">
            <AttributeValue DataType="string">read</AttributeValue>
            <ActionAttributeDesignator DataType="string
                    AttributeId="action-id"/>
            </ActionMatch>
         </Action>
         <Action>
            <ActionMatch MatchId="string-equal">
            <AttributeValue DataType="string">write</AttributeValue>
            <ActionAttributeDesignator DataType="string
                    AttributeId="action-id"/>
            </ActionMatch>
         </Action>
      </Actions>
   </Target>
</Rule>
```

As far as the deletion of data is concerned, the data subject can express data handling preferences that specify the obligation that the data controller has to enforce regarding deleting personal data. This obligation can be expressed by the A-PPL obligation action **ActionDeletePersonalData**, which will be used by Kim to delete personal data that has been collected about him. The same action can also be used by Sandra. Listing 4.2 shows how this obligation can be expressed using the trigger **TriggerAtTime**.

Listing 4.2: Delete Personal Data

```
<ppl:DataHandlingPreferences>
   <!-- Data Subject's data handling preferences -->
      <Obligation>
            <TriggersSet>
            <TriggerAtTime>
               <Start><StartNow/></Start>
               <MaxDelay>
               <!-- Delete personal data within 2 years -->
                  <Duration>P2Y0M0DT0H0M0S</Duration>
               </MaxDelay>
            </TriggerAtTime>
            </TriggersSet>
            <ActionDeletePersonalData>
 </ppl:DataHandlingPreferences>
```

O2: As a data controller, the hospital needs to provide a policy on what data is collected and for which purposes. (see Listing 4.12) The second obligation can be fulfilled by creating a **DataHandlingPolicy element**, which describes how the hospital intend to use the personal data that is collected from the patients (and/or the relatives). The policy must contain information on what data will be collected, for what purpose, how long it will be stored and with whom it will be shared.

O3: As a data controller, the hospital must ask the data subjects (ie. the patients) explicit consent for collecting and processing personal data. PPL does not impose any particular technology to collect consent from data subject. However, informed content can be achieved through the PPL protocol: the sticky policy that travels with the data being sent to the data controller is the result of a matching procedure between the user's preferences and the data controller's policy. Releasing this sticky policy constitutes the consent from the data subject. This obligation may also require user interaction on a higher level, for example by allowing the user to click a button to confirm that he accepts the hospital's privacy policy. However, once the data subject has given his consent, the accountability policy trigger **TriggerOnPreferencesUpdate*** can be used to inform the hospital: the data subject's A-PPL engine triggers this and sends a notification to the hospital.

O4: As data controllers, the relatives/friends must ask the data subjects (ie. the patients) explicit consent for collecting and processing personal data. See O3.

O5: As a data controller, the hospital must only use personal data for the specified purposes only. In A-PPL, the Obligation Enforcement Engine (OEE) will make sure that the committed obligations are enforced. Listing 4.3 shows an example of how to express the purposes of usage that are authorized for the patient's personal data.

Listing 4.3: Authorization to use the data for the specified list of purposes

```
<ppl:AuthzUseForPurpose>
<!-- Patient's data authorized to be used for following purposes-->
            <ppl:Purpose>admin</ppl:Purpose>
            <ppl:Purpose>research</ppl:Purpose>
            <ppl:Purpose>marketing</ppl:Purpose>
</ppl:AuthzUseForPurpose>
```

O6: As a data controller, the hospital must, upon request, provide evidence to the data subjects (the patients) on their personal data processing practices. The A-PPL obligation enforcement engine will be extended with audit features, which would allow the hospital to keep track of actions that are executed by the OEE. According to [TNR11], it is possible to enable automatic analysis of traces and check for conformance of data usage and data policies. The obligation action **ActionLog*** can be used to specify how to track the collection and processing of personal data in the cloud. Listing 4.4 shows an example of logging data collection and processing practices. Any time the data is accessed, sent to a third-party or deleted (these events thus define triggers), the data controller logs within 5 minutes the information related to these events.

Listing 4.4: Obligation to log: access and forwarding and deletion of data

```
<Obligation>
            <TriggersSet>
               <TriggerPersonalDataAccessedForPurpose>
                  <MaxDelay>
                  <!-- Delay of 5 minutes -->
                     <Duration>5M</Duration>
                  </MaxDelay>
               </TriggerPersonalDataAccessedForPurpose>
               <TriggerOnPersonalDataDeleted>
                  <MaxDelay>
                     <Duration>5M</Duration>
                  </MaxDelay>
               </TriggerOnPersonalDataDeleted>
               <TriggerOnPersonalDataSent>
                  <MaxDelay>
                     <Duration>5M</Duration>
                  </MaxDelay>
               </TriggerOnPersonalDataSent>
            </TriggersSet>
            <ActionLog>
               <Timestamp>
                  <EnvironmentAttributeSelector
                          AttributeId="current-time"/>
               </Timestamp>
               <Action>
                   <AttributeValue DataType=string>
                      Personal Data accessed, sent or deleted
                   </AttributeValue >
               </Action>
               <Purpose>
                  <AttributeValue DataType=string>admin
                        </AttributeValue>
               </Purpose>
               <Subject>
                   <AttributeValue DataType=string>hospital
                         </AttributeValue >
               </Subject>
               <Resource>
                  <ResourceAttributeDesignator DataType="string"
                       AttributeId="resource:resource-id"/>
               </Resource>
            </ActionLog>
```

```
</Obligation>
```

In addition, A-PPL enables the definition of an audit obligation that requests evidence from the data controller. Listing 4.5 presents an example policy in A-PPL that requires the data controller to collect logs for the patient's medical records on reception of an evidence request.

Listing 4.5: Collection of Evidence

```
<Obligation>
            <TriggersSet>
               <TriggerOnEvidenceRequestReceived>
                  <MaxDelay>
                  <!-- Delay of 5 minutes -->
                     <Duration>5M</Duration>
                  </MaxDelay>
               </TriggerOnEvidenceRequestReceived>
            </TriggersSet>
            <ActionEvidenceCollection>
               <Evidence>
                  <Attribute AttributeId="evidence-type" DataType="string">
                    <AttributeValue>Logs</AttributeValue>
                  </Attribute>
                </Evidence>
                <Resource>
                  <Attribute AttributeId="resource-id" DataType="string">
                    <AttributeValue>Medical Record</AttributeValue>
                  </Attribute>
                </Resource>
                <Subject>
                  <Attribute AttributeId="subject-id" DataType="string">
                    <AttributeValue>Kim</AttributeValue>
                  </Attribute>
                </Subject>
                <Recipient>
                  <Attribute AttributeId="recipient-id" DataType="string">
                    <AttributeValue>
                       Patient:Kim
                   </AttributeValue>
                 </Attribute>
               </Recipient>
            </ActionEvidenceCollection>
</Obligation>
```

O7: As a data controller, the hospital must notify the processing of personal data to the competent Data Protection Authority. The obligation action **ActionNotify*** can be used for this purpose.

Listing 4.6: Notify the Norwegian DPA

```
<Obligation>
            <TriggersSet>
               <TriggerPeriodic>
                  <Start>
                     <DateAndTime>
                        2013-11-01T12:00:00
                     </DateAndTime>
                  </Start>
                  <Period>
                     <Duration>P0Y0M30DT0H0M0S </Duration>
```

```
                </Period>
             </TriggerPeriodic>
          </TriggersSet>
          <ActionNotify>
             <Media>mail</Media>
             <Address>
                 Norwegian Data Protection Authority, Oslo, Norway
              </Address>
             <Recipients>Norwegian DPA</Recipients>
             <Type>Report on data collection</Type>
          </ActionNotify>
</Obligation>
```

O8: As a data controller, the hospital must notify the data subjects (i.e. the patients) of security or personal data breaches. The obligation action `ActionNotify` can be used for this purpose.

Listing 4.7: Notify the Data Subject in case of Policy Violation

```
<Obligation>
<!-- Notification of the Data Subject about Policy violation -->
          <TriggersSet>
             %<TriggerOnPolicyViolation>
             <TriggerOnPolicyViolation/>
             %<TriggerOnDataLost>
             <TriggerOnDataLost/>
          </TriggersSet>
          <ActionNotify>
             <Media>e-mail</Media>
             <Address>Kim@example.com</Address>
             <Recipients>Patient:Kim</Recipients>
             <Type>Policy Violation</Type>
          </ActionNotify>
       </Obligation>
```

O9: As joint data controllers, the relatives must only use personal data for the specified purposes only. The authorization element **AuthzUseForPurpose** makes it possible to restrict the list of purposes for which the joint data controllers could use the collected data. Used within the context of the **AuthzDownstreamUsage** data handling policy element, authorization ”use for purpose” can further be restricted to distinguish the list of the purposes between primary data controller (i.e. the hospital) and the downstream data controllers (joint data controllers). The specification of the lists of authorized purposes is similar to the one in Listing 4.3.

O10: As a data processor, the MedNet platform provider must log all access to personal data. The obligation action **ActionLog*** will allow The MedNet platform provider to specify how to track the collection and processing of personal data in the cloud. Listing 4.4 already showed how to express this obligation in A-PPL.

O11: As a data processor, the MedNet platform provider must inform the data controller (i.e. the hospital) about the engagement of sub-processors (ie. the third party service providers in the service delivery chain). The obligation action **ActionNotify*** can be used to inform the hospital what third party service providers are involved. In addition, the obligation trigger **TriggerPersonalDataSent** can be triggered when personal data have been transferred to a third party. Listing 4.6 already presented the notification obligation. For O11, the policy can specify the data controller ID as the recipient of the notification within the `<Recipients>` tags.

O12: As a data processor, the MedNet platform provider must, upon request, provide evidence to the data controller (ie. the hospital) on their personal data processing practices. As for O10, the obligation action **ActionLog*** will allow The MedNet platform provider to track the collection and processing of personal data. Actually, **ActionLog*** will result in the generation of logs that will form the evidence that auditors will analyze to assess the compliance or the non-compliance of MedNet platform provider with regulations and user preferences.

O13: As a data processor, the MedNet platform provider must, upon request, provide evidence to the data controller (ie. the hospital) on the correct and timely deletion of personal data. To require timely deletion of personal data, the A-PPL obligation trigger **TriggerAtTime** can be set to trigger at the personal data retention time, i.e. when the data are to be deleted. This trigger can be combined with the obligation trigger **TriggerPersonalDataDeleted** that can be used to notify the data subject of the deletion of its data. In addition, if necessary, the obligation action **ActionLog*** will allow The MedNet platform provider to log when and how personal data has been deleted.

Listing 4.8 presents an example of policy on the data processor side that requires the data processor to delete personal data 2 years after its collection, to notify the data subject and to log the deletion action.

Listing 4.8: Evidence of deletion

```
<ObligationsSet>
  <Obligation>
<!-- Delete personal data -->
    <TriggerAtTime>
      <Start><StartNow/></Start>
      <MaxDelay>
<!-- Delete personal data within 2 years -->
        <Duration>P2Y0M0DT0H0M0S</Duration>
      </MaxDelay>
     </TriggerAtTime>
     <ActionDeletePersonalData/>
  </Obligation>
  <Obligation>
    <TriggerPersonalDataDeleted>
      <MaxDelay>
        <Duration>P0Y0M0DT0H5M0S<Duration>
      </MaxDelay>
    </TriggerPersonalDataDeleted>
    <ActionsSet>
    <ActionNotify>
      <Media>e-mail</Media>
      <Address>Kim@example.com</Address>
      <Recipients>Patient:Kim</Recipients>
      <Type>Deletion of personal data</Type>
    </ActionNotify>
    <ActionLog>
      <Timestamp>
        <EnvironmentAttributeSelector
```

```
                          AttributeId="current-time"/>
        </Timestamp>
        <Action>
          <AttributeValue DataType=string>
                       Personal Data deleted
          </AttributeValue >
        </Action>
        <Subject>
          <AttributeValue DataType=string>
                Data Processor
          </AttributeValue >
        </Subject>
        <Resource>
          <ResourceAttributeDesignator  DataType="string"
                       AttributeId="resource:resource-id"/>
          </Resource>
        </ActionLog>
      </ActionsSet>
    </Obligation>
  </ObligationsSet>
```

O14: As a data processor, the MedNet platform provider must notify the data controller (i.e. the hospital) of security or personal data breaches . The obligation action **ActionNotify*** can be used to inform the hospital of any breach that may occur. This policy in A-PPL will look like Listing 4.9.

Listing 4.9: Notify the data controller in case of security breaches

```
<Obligation>
<!-- Notification of the Data Controller of security breaches -->
  <TriggersSet>
    <TriggerOnPolicyViolation>
    </TriggerOnPolicyViolation>
    <TriggerOnDataLost>
    </TriggerOnDataLost>
  </TriggersSet>
  <ActionNotify>
    <Media>e-mail</Media>
    <Address>privacy.officer@example.com</Address>
     <Recipients>Privacy Officer</Recipients>
     <Type>Security Breach</Type>
   </ActionNotify>
</Obligation>
```

O15: As data processors, the MedNet platform provider, the provider of Cloud x and the provider of Cloud y must facilitate for regulators to review evidence of the processing of personal data whenever requested. During an audit the regulators may ask evidence concerning the processing of personal data such as evidence of deletion of data or evidence for correct storage of personal data. A-PPl enables the auditor to specify an obligation for evidence request thanks to the **ActionAudit*** element. Listing 4.10 shows how to express an audit of the log entries in Cloud X's logs that describe the actions performed on patient's personal data.

Listing 4.10: Audit the data processors

```
<Obligation>
```

```
<!-- Audit Cloud x every semester -->
  <TriggersSet>
    <TriggerPeriodic>
      <Start><StartNow/></Start>
      <Period>
        <Duration>P0Y6M0DT0H0M0S</Duration>
      </Period>
    </TriggerPeriodic>
  </TriggersSet>
  <ActionAudit>
    <Evidence>Logs</Media>
    <Resource>Personal Data</Resource>
    <Subject>Patient:Kim</Subject>
    <Recipient>Cloud x</Recipient>
   </ActionAudit>
</Obligation>
```

The following listings show how to translate these obligations into concrete A-PPL policies. Listing 4.11 indicates the data subject's preferences about the processing of his personal data.

Listing 4.11: KData subject's preferences

```
<ppl:Policy>
<!-- Target: condition of application of the policy-->
   <Target>
      <Subjects>
      <!-- Policy applies to Data Subject -->
         <Subject><AnySubject/></Subject>
      </Subjects>
      <Resources>
      <!-- Policy applies to Resource belonging to the data subject -->
         <Resource>
            <ResourceMatch MatchId=function:string-equal>
            <AttributeValue DataType=string>Patient:Kim</AttributeValue >
            <ResourceAttributeDesignator  DataType="string"
                   AttributeId="resource:resource-owner"/>
            </ResourceMatch>
         </Resource>
      </Resources>
      <Actions><AnyAction/><Actions>
   </Target>

   <ppl:DataHandlingPreferences>
<!-- Data subject's data handling preferences -->
   <Obligation>
     <TriggersSet>
       <TriggerAtTime>
         <Start><StartNow/></Start>
         <MaxDelay>
<!-- Delete personal data within 2 years -->
           <Duration>P2Y0M0DT0H0M0S</Duration>
          </MaxDelay>
        </TriggerAtTime>
      </TriggersSet>
    <ActionDeletePersonalData>
  </ppl:DataHandlingPreferences>
 </ppl:Policy>
```

Listing 4.12 presents policies defined by the data controller (i.e. the hospital), recalling all the obligations that we analyzed above.

Listing 4.12: Hospital policy

```
<ppl:Policy>
<!-- Target: condition of application of the policy-->
   <Target>
      <Subjects>
      <!-- Policy applies to Data Subject -->
         <Subject>
            <SubjectMatch MatchId=function:string-equal>
            <AttributeValue DataType=string>Patient:Kim</AttributeValue>
            <SubjectAttributeDesignator  DataType="string"
                   AttributeId="subject:subject-id"/>
            </SubjectMatch>
         </Subject>
      </Subjects>
      <Resources>
      <!-- Policy applies to Resource belonging to data subject -->
         <Resource>
            <ResourceMatch MatchId=function:string-equal>
            <AttributeValue DataType=string>Patient:Kim</AttributeValue >
            <ResourceAttributeDesignator  DataType="string"
                   AttributeId="resource:resource-owner"/>
            </ResourceMatch>
         </Resource>
      </Resources>
      <Actions><AnyAction/><Actions>
   </Target>

   <xacml:Rule Effect="Permit">
      <Target>
         <Subject><AnySubject/> </Subject>
         <Resources><AnyResource/></Resources>
         <Actions>
            <Action>
               <ActionMatch MatchId="string-equal">
               <AttributeValue DataType="string">read</AttributeValue>
               <ActionAttributeDesignator DataType="string"
                        AttributeId="action-id"/>
               </ActionMatch>
            </Action>
            <Action>
               <ActionMatch MatchId="string-equal">
               <AttributeValue DataType="string">write</AttributeValue>
               <ActionAttributeDesignator DataType="string"
                      AttributeId="action-id"/>
               </ActionMatch>
            </Action>
         </Actions>
      </Target>
   </Rule>

   <ppl:DataHandlingPolicy>
   <!-- Hospital's data handling policies -->
      <ppl:AuthorizationsSet>
         <ppl:AuthzUseForPurpose>
<!-- Kim's data authorized to be used for following purposes-->
            <ppl:Purpose>admin</ppl:Purpose>
            <ppl:Purpose>research</ppl:Purpose>
            <ppl:Purpose>marketing</ppl:Purpose>
         </ppl:AuthzUseForPurpose>
```

```
          <ppl:AuthzDownstreamUsage allowed="true" />
          <!-- Kim's data authorized to downstream usage-->
       </ppl:AuthorizationsSet>
       <ObligationsSet>
          <Obligation>
<!-- Obligation to log: access, forwarding and deletion of data-->
             <TriggersSet>
                <TriggerPersonalDataAccessedForPurpose>
                   <MaxDelay>
                   <!-- Delay of 5 minutes -->
                      <Duration>5M</Duration>
                   </MaxDelay>
                </TriggerPersonalDataAccessedForPurpose>
                <TriggerOnPersonalDataDeleted>
                   <MaxDelay>
                      <Duration>5M</Duration>
                   </MaxDelay>
                </TriggerOnPersonalDataDeleted>
                <TriggerOnPersonalDataSent>
                   <MaxDelay>
                      <Duration>5M</Duration>
                   </MaxDelay>
                </TriggerOnPersonalDataSent>
             </TriggersSet>
             <ActionLog>
                <Timestamp>
                   <EnvironmentAttributeSelector
                          AttributeId="current-time"/>
                </Timestamp>
                <Action>
                    <AttributeValue DataType=string>
                       Personal Data accessed, sent or deleted
                    </AttributeValue >
                </Action>
                <Purpose>
                   <AttributeValue DataType=string>admin
                         </AttributeValue>
                </Purpose>
                <Subject>
                    <AttributeValue DataType=string>hospital
                          </AttributeValue >
                </Subject>
                <Resource>
                   <ResourceAttributeDesignator  DataType="string"
                       AttributeId="resource:resource-id"/>
                </Resource>
             </ActionLog>
          </Obligation>
          <Obligation>
<!-- Notification of the Data Subject about Policy violation -->
             <TriggersSet>
                <TriggerOnPolicyViolation>
                </TriggerOnPolicyViolation>
                <TriggerOnDataLost>
                </TriggerOnDataLost>
             </TriggersSet>
             <ActionNotify>
                <Media>e-mail</Media>
                <Address>Kim@example.com</Address>
                <Recipients>Patient:Kim</Recipients>
                <Type>Policy Violation</Type>
             </ActionNotify>
          </Obligation>
          <Obligation>
<!-- Notification of the Norwegian DPA about collection of data -->
             <TriggersSet>
                <TriggerPeriodic>
                   <Start>
                      <DateAndTime>
                         2013-11-01T12:00:00
                       </DateAndTime>
                    </Start>
                   <Period>
                      <Duration>P0Y0M30DT0H0M0S </Duration>
                    </Period>
                 </TriggerPeriodic>
              </TriggersSet>
             <ActionNotify>
                <Media>mail </Media>
                <Address>
                     Norwegian Data Protection Authority , Oslo , Norway
                  </Address>
                <Recipients>Norwegian DPA</Recipients>
                <Type>Report on data collection </Type>
              </ActionNotify>
           </Obligation>
       </ObligationsSet>
       </ppl:DataHandlingPolicy>
</ppl:Policy>
```

Chapter 5

Conclusion

The main objective of this document was to define a policy representation framework allowing to express not only privacy and security concerns but also accountability aspects. In this aim, we first identified the accountability requirements that have to be meet by this framework. These requirements were recognized by analyzing the A4Cloud conceptual framework [CFH$^+$13b] and the outcomes of the first stakeholder workshop for elicitation of requirements [MJH$^+$13]. The second task, we achieved, was to review the state of the art in existing policy representation languages. Thus, we analyzed several existing standards and academic proposals against our requirements to determine their ability to represent accountability concepts. It turned out that none of the existing standards could satisfy all the accountability requirements we defined; while security and access control are generally covered by existing standards, delegation, chain of trust and auditing are only addressed by some academic proposals. Based on this study, we elected the PrimeLife Policy Language (PPL) as the best candidate: It is based on the XACML standard, focuses on privacy and is extensible. PPL gives service providers an automatic mean to define and manage privacy policies while applications are enabled to compare these service privacy policies with user privacy preferences.

During our discussion with other A4Cloud partners involved mainly in the accountability framework design and the contractual and regulatory considerations, it was obvious that a huge gap exists between the formulation of obligations in law and what we could enforce with PPL. To fill this gap we proposed a two-level policy representation framework. The top level provides the means to express legal obligations in a machine readable representation but still human readable. While the bottom level is devoted to concrete enforcement of the obligations, i.e. in a machine understandable representation, based on the PPL language.

The adopted point of view for the top level was to define a domain specific language, the Abstract Accountability Language (AAL), dedicated to accountability obligations. User preferences define privacy and disclosure properties the agents in the system should comply with. Processor obligations express data usage properties the processor ensures to implement. The deliverable provides a formal syntax and semantics in the Chapter 2. The semantics has two steps: A logical view at the top level and an operational view at the concrete level.

The bottom level is the concrete level based on the PPL language and which deals with policy enforcement. However, PPL does not address all the identified accountability requirements and we needed to extend it. Thus, we defined an enhanced version of PPL, A-PPL for accountable PPL, integrating new concerns for logging, notification, and triggers. A-PPL also provides new features as the audit step and the evidence collection.

Finally, we made a first attempt to validate both AAL and A-PPL. We validated AAL and its semantics against the sixteen requirements and with the privacy level agreement from CSA. Furthermore, we analyze the obligations of the health care use case and show how AAL and A-PPL can express these obligations.

Future work will contribute to improve the languages we have defined in our policy representation framework and to describe the mapping from AAL to A-PPL.

Bibliography

[AAB$^+$13] Rehab Alnemr, Monir Azraoui, Karin Bernsmed, Massimo Felici, Simone Fischer-Hbner, Bushra Hasnain, Eleni Kosta, Theofrastos Koulouris, Ronald Leenes, Christopher Millard, Maartje Niezen, David Nuez, Melek nen, Alain Pannetrat, Nick Papanikolaou, Siani Pearson, Daniel Pradelles, Chris Reed, Christoph Reich, Jean-Claude Royer, Anderson Santana de Oliveira, Dimitra Stefanatou, Vasilis Tountopoulos, and Tomasz Wiktor Wlodarczyk. Glossary of terms and definitions. Technical report, Accountability for Cloud and Future Internet Services - A4Cloud Project, November 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.

[ABE$^+$13] Monir Azraoui, Karin Bernsmed, Kaoutar Elkhiyaoui, Hervé Grall, Refik Molva, Melek Önen, Jean-Claude Royer, Anderson Santana de Oliveira, Jakub Sendor, and Mario Südholt. Policy model and language requirements. Technical Report MS:C-4.1, Accountability for Cloud and Future Internet Services - A4Cloud Project, 2013.

[ABP10] Muhammad Ali, Laurent Bussard, and Ulrich Pinsdorf. Obligation language and framework to enable privacy-aware soa. In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, Nora Cuppens-Boulahia, and Yves Roudier, editors, *Data Privacy Management and Autonomous Spontaneous Security*, volume 5939 of *Lecture Notes in Computer Science*, pages 18–32. Springer Berlin Heidelberg, 2010.

[ACD$^+$] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement).

[ADG$^+$12] Diana Allam, Rémi Douence, Hervé Grall, Jean-Claude Royer, and Mario Südholt. Well-Typed Services Cannot Go Wrong. Rapport de recherche RR-7899, INRIA, May 2012.

[AN08] Irem Aktug and Katsiaryna Naliuka. ConSpec – a formal language for policy specification. In *Electronic Notes in Theoretical Computer Science*, volume 197, pages 45–58. 2008.

[And74] J. Anderson. Computer Security Technology Planning Study. Technical Report ESD-TR-7351, Electronic Systems Division, Hanscom Air Force Base, Hanscom, MA, 1974.

[BA05] Travis D. Breaux and Annie I. Anton. Deriving semantic models from privacy policies. In *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '05)*, pages 67–76, June 2005.

[BB90] Gérard Berry and Gérard Boudol. The chemical abstract machine. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '90, pages 81–94. ACM, 1990.

[BBC+06] Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, Glen Daniels, Phillip Hallam-Baker, Maryann Hondo, Chris Kaler, Dave Langworthy, Anthony Nadalin, Nataraj Nagaratnam, Hemma Prafullchandra, Claus von Riegen, Daniel Roth, Jeffrey Schlimmer, Chris Sharp, John Shewchuk, Asir Vedamuthu, Ümit Yalçinalp, and David Orchard. Web Services Policy 1.2 - Framework (WS-Policy). Technical report, W3C, April 2006.

[BFG10] Moritz Y. Becker, Cédric Fournet, and Andrew D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

[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.

[Bib77] Biba. Integrity Considerations for Secure Computer Systems. *MITRE Co., technical report ESD-TR 76-372*, 1977.

[BL96] David Elliott Bell and Leonard J. LaPadula. Secure computer systems: A mathematical model, volume ii. *Journal of Computer Security*, 4(2/3):229–263, 1996.

[BLMR04] A.K. Bandara, E.C. Lupu, J. Moffett, and A. Russo. A goal-based approach to policy refinement. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, pages 229–239, 2004.

[BMB10] Moritz Y Becker, Alexander Malkis, and Laurent Bussard. S4p: A generic language for specifying privacy preferences and policies. *Microsoft Research*, 2010.

[BO12] Alistair Barros and Daniel Oberle. *Handbook of Service Description: USDL and Its Methods*. Springer Publishing Company, Incorporated, 2012.

[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.

[CBMK10] Jorge Cardoso, A. Barros, N. May, and U. Kylau. Towards a unified service description language for the internet of services: Requirements and first developments. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 602–609, 2010.

[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.

[CFH$^+$13a] Daniele Catteddu, Massimo Felici, Giles Hogben, Amy Holcroft, Eleni Kosta, Ronald Leenes, Maartje Niezen, Christopher Millard, David Nu nez, Nick Papanikolaou, Siani Pearson, Daniel Pradelles, Chris Reed, Jean-Claude Royer, Dimitra Stefanatou, Vasilis Tountopoulos, and Tomasz Wiktor Wlodarczyk. Ms:c-2.2. Technical Report MS:C-2.2, Accountability for Cloud and Future Internet Services - A4Cloud Project, 2013.

[CFH$^+$13b] Daniele Catteddu, Massimo Felici, Giles Hogben, Christopher Millard, Nick Papanikolaou, Siani Pearson, Daniel Pradelles, and Chris Reed. Scoping report and initial glossary. Technical Report MS:C-2.1, Accountability for Cloud and Future Internet Services - A4Cloud Project, 2013.

[CLM02] Lorrie Cranor, Marc Langheinrich, and Massimo Marchiori. *A P3P Preference Exchange Language 1.0 (APPEL1.0)*. World Wide Web Consortium, 2002.

[Con99] US Congress. Gramm-leach-bliley act, financial privacy rule. 15 usc 68016809. `http://www.law.cornell.edu/uscode/usc_sup_01_15_10_94_20_I.html`, November 1999.

[Con02] US Congress. Health insurance portability and accountability act of 1996, privacy rule. 45 cfr 16. `http://www.access.gpo.gov/nara/cfr/waisidx_07/45cfr164_07.html`, August 2002.

[DDLS01] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In Morris Sloman, Jorge Lobo, and Emil Lupu, editors, *POLICY*, volume 1995 of *Lecture Notes in Computer Science*, pages 18–38. Springer, 2001.

[DGJ$^+$10] Henry DeYoung, Deepak Garg, Limin Jia, Dilsun Kaynar, and Anupam Datta. Experiences in the logical specification of the HIPAA and GLBA privacy laws. In *9th Annual ACM Workshop on Privacy in the Electronic Society (WPES '10)*, pages 73–82, 2010.

[DKLL13] Marnix Dekker, Christoffer Karsberg, Matina Lakka, and Dimitra Liveri. Schemes for Auditing Security Measures, An Overview. Technical report, European Union Agency for Network and Information Security, Greece, September 2013.

[Eur95] European Parliament and the Council of the European Union. Directive 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 . http://ec.europa.eu/justice/policies/privacy/docs/95-46-ce/dir1995-46_part1_en.pdf, 1995.

[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.

[FJG+13] M Frtunic, F Jovanovic, M Gligorijvic, L Dordevic, S Janicijevic, P.H. Meland, K. Bernsmed, M. Jaatun, H. Castejon, and A. Undheim. CloudSurfer - a Cloud Broker Application for Security Concerns. In *Proceedings of the 3rd International Conference on Cloud Computing and Service Science (CLOSER 2013)*, 2013.

[FJWX12] Joan Feigenbaum, Aaron D. Jaggard, Rebecca N. Wright, and Hongda Xiao. Systematizing accountability in computer science. Technical Report YALEU/DCS/TR-1452, University of Yale, 2012. www.cs.yale.edu/publications/techreports/tr1452.pdf.

[KL03] Shawn Kerrigan and Kincho H. Law. Logic-based regulation compliance-assistance. In *International Conference on Artificial Intelligence and Law*, pages 126–135, 2003.

[LM09] Daniel Le Métayer. A formal privacy management framework. *Formal Aspects in Security and Trust*, pages 1–15, 2009.

[LSE03] D. Davide Lamanna, James Skene, and Wolfgang Emmerich. SLAng: A Language for Defining Service Level Agreements. In *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, FTDCS '03, pages 100–, Washington, DC, USA, 2003. IEEE Computer Society.

[MBJ+13] P.H. Meland, K. Bernsmed, M. Jaatun, H. Castejon, and A. Undheim. Expressing cloud security requirements for SLAs in deontic contract languages for cloud brokers. *International Journal of Cloud Computing (to appear)*, 2013.

[MJH+13] Nils Brede Moe, Martin Gilje Jaatun, Borge Haugset, Maartje Niezen, and Massimo Felici. Stakeholder workshop 1 results (initial requirements). Technical Report D:B-2.1, Accountability for Cloud and Future Internet Services - A4Cloud Project, 2013.

[OAS03] OASIS Standard. A Brief Introduction to XACML. https://www.oasis-open.org/committees/download.php/2713/l, 2003.

[OAS04] OASIS Web Services Reliable Messaging TC. WS-Reliability 1.1. http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf, November 2004.

[OAS06] OASIS Web Service Security (WSS) TC. Web Services Security: SOAP Message Security 1.1. https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, 2006.

[OAS12] OASIS Web Services Secure Exchange (WS-SX) TC. WS-Trust 1.4. http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/errata01/os/ws-trust-1.4-errata01-os-complete.html, 2012.

[OAS13] OASIS Standard. eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, 2013.

[P3P] W3C. Platform for Privacy Preferences. http://www.w3.org/P3P/.

[PAW04] C. Powers, S. Adler, and B. Wishart. Epal translation of the freedom of information and protection of privacy act. Technical Report Version 1.1, Tivoli Software, IBM, 2004.

[PMA$^+$09] Alessandro Perilli, Andrea Manieri, Avner Algom, Craig Balding, and Various. Cloud Computing Risk Assessment ENISA. Technical report, ENISA, Greece, November 2009.

[PPM11] N. Papanikolaou, S. Pearson, and M. Casassa Mont. Towards Natural-Language Understanding and Automated Enforcement of Privacy Rules and Regulations in the Cloud: Survey and Bibliography. *Secure and Trust Computing, Data Management and Applications, Communications in Computer and Information Science*, 187:166–173, 2011.

[Pri11] PrimeLife Consortium. PrimeLife. http://primelife.ercim.eu/, 2011.

[PT12] S. Pearson and P Tsiavos. Taking the Creative Commons beyond Copyright: Developing Smart Notices as User Centric Consent Management Systems for the Cloud. *International Journal of Cloud Computing, Inderscience*, 2012.

[SOA03] Simple Object Access Protocol v1.2. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, 2003.

[Ste09] Steel, Christopher and Lai, Ray and Naggapan, Ramesh. Core Security Patterns: Identity Management Standards and Technologies . http://www.informit.com/articles/article.aspx?p=1398625&seqNum=12, 2009.

[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.

[Tra10] Slim Trabelsi. Second Release of the Policy Engine . http://primelife.ercim.eu/images/stories/deliverables/d5.3.2-second_release_of_the_policy_engine-public.pdf, 2010.

[ZGFvS] Martijn Zuidweg, Jose Goncalves, Pereira Filho, and Marten van Sinderen. Using p3p in a web service-based context-aware application platform. http://www.w3.org/2003/p3p-ws/pp/utwente.pdf.

Appendix A

State of the Art in Policy Languages and Obligation Representations

This appendix is organized in two parts. In the first part, we present some standards and approaches that have been proposed for policy representation and we study their suitability for accountability representation according to the accountability requirements identified in Section 1.1. Although most of the reviewed languages fail at meeting the accountability requirements at first glance, some of them can be extended to support accountability policies. While our policy language review is by no means exhaustive, it has aimed at covering the most relevant standards to our work, namely, XACML, PPL, WS-Security, USDL, SLAng, P3P, SecPal4P, Ponder, and ConSpec.

Moreover, we note that while existing policy standards are suitable for access and usage control, they are not close to natural language, therefore making a direct mapping of regulations and legal texts (which are crucial for accountability) cumbersome. Therefore, the second part of the appendix investigates previous work on formal and semi-formal approaches that can be used to map obligations to a human/machine understandable language. Accordingly, we analyze the approaches of SecPal, SIMPL, Semantic Models from Privacy Policies, Logic-Based Regulation Compliance-Assistance, an application of EPAL, and Logical Specification of the HIPAA and GLBA Privacy Laws. The purpose here is not to compare them with our identified requirements but rather to get some feeling of natural and formal obligation representations. Notice that our review here focuses primarily on formal approaches that only covers privacy obligations. This is due to the fact that most of previous work on formal methods deal with privacy obligations with little attention devoted to accountability. Yet, we believe that by reviewing these approaches, we can motivate the design choices we opted for when devising our AAL.

A.1 XACML

XACML [OAS13] stands for eXtensible Access Control Markup Language and consists of an OASIS standard that defines both a declarative language for expressing access control policies and a request-response message exchange protocol to obtain access control decisions. These are both expressed in eXtensible Markup Language (XML) [BPSM+97] that is both human and machine-readable.

- The XACML policy language describes general access control rules and conditions. It presents extension points for defining new functions, data types, combination logic, etc.
- The XACML request-response message exchange is used to express queries to a decision engine about whether an action should be allowed (request) and describes the respective answers (response).

The XACML standard is widely adopted [OAS03] for expressing access control policies with the properties of interoperability and extensibility. For example, XACML is deployed in authorization solution products for financial industries or health care systems.

General usage scenario

The language model of XACML is shown in the diagram of Figure A.1. XACML provides a standard

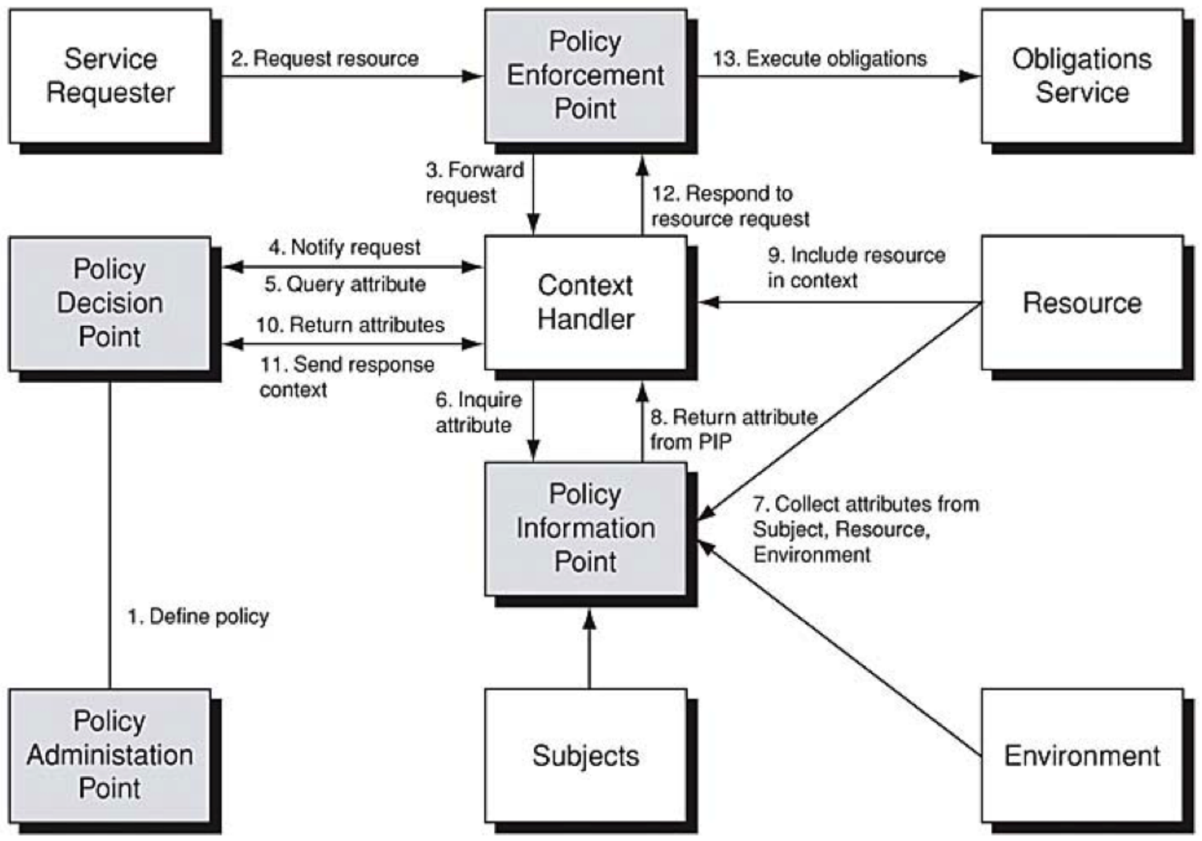

Figure A.1: The language model of XACML [Ste09]

reference architecture to achieve enforcement of XACML policies. When a user wants to take some action on a resource, he makes a request (2) to the Policy Enforcement Point (PEP) that protects the resource. The PEP forms a request based on the attributes of the requester, the resource and the action on that resource that the requester wants to perform. The PEP then sends this request (3, 4) to a Policy Decision Point (PDP). The PDP evaluates the requests and finds policies that apply to that request. Then, the PDP builds a response about whether access to the resource should be granted. The response is sent back to the PEP (11, 12) which can then allow or deny the access to the requester. Both the PEP and the PDP may also need to gather some attributes about the Data Subject or the resource (such as subject, resource, action, environment attributes) that they query to a context handler (5, 10) that forwards the attribute query to a Policy Information Point (PIP) (6-9). In conjunction with the enforcement of an authorization decision, the PEP performs the obligation actions specified in the policy. The Policy Administration Point (PAP) writes and maintains the policies (1).

Terminology

An XACML policy contains one or more rules, policies and/or policy sets.

- A set of rules is encapsulated in a `Policy`. The main components of a `Rule` are:
  - A Target that defines the requests to which the rule is intended to apply. It is expressed in terms of subjects, resources and actions.
  - An Effect that indicates the rule-writer's intended consequence if the evaluation of the rule outputs true. The possible effects are either `Permit` or `Deny`.
  - A Condition that refines the applicability of the rule by putting restrictions on `Target`'s attributes (subjects, resources or actions).
  - Obligations that specifies actions to be performed by the PEP in conjunction with the enforcement of the authorization decision. The XACML specification [OAS13] provides an example of a medical system in which the PEP is obliged to send an e-mail to a patient whose record have been accessed.
  - Advice that gives supplemental information about an authorization decision.
- A Policy is a container for a set of Rules. The main components of a Policy are:
  - A Target that specifies the requests to which the policy is intended to apply.
  - A Rule-combining algorithm that specifies how to combine decisions of different rules in the same policy to obtain a final decision for the policy evaluation.
  - Obligations and Advice, just like in a Rule.
- A Policy Set that encapsulates a set of policies. It comprises the following main components:
  - A Target that specifies the requests to which the policy set is intended to apply.
  - A Policy-combining algorithm that specifies how to combine decisions of different policies in the same policy set to obtain a final decision for the policy set evaluation.
  - Obligations and Advice.

An XACML context defines access requests that might be submitted to a PDP. The structure of a request context comprises one or more attributes that specify:

- the entity making the access request (a subject),
- the resource to which the subject(s) has requested access, identified by its URI, and
- the action that the subject(s) wishes to take on the resource (read, write, etc.).

The PDP processes this request by comparing the attributes in the request with the policy target, and then with the target of rules in the policy. As a result of evaluating the policy, the PDP sends a response context that specifies the access decision taken according to the rule- and policy-combining algorithm. A policy can return `Permit`, `Deny`, `Not Applicable` or `Indeterminate`.

One another optional element of the XACML policy are Obligations. In XACML, Obligations are a set of directives from the PDP that must be performed by the PEP in conjunction with an authorization decision. It must be performed either before or after an access is granted. If it is not feasible for the PEP to comply with this obligation, the access must not be granted.

Analysis according to our requirements

(R1) Capturing Privacy Preferences: XACML does not enable the data subjects to express their privacy preferences about the usage of their data.
(R2) Anonymity/Pseudonymity: XACML does not allow pseudonymous access control. On the other hand, anonymous access control is possible. If the element `<Subject>` does not contain an attribute `subject-id`, this subject is then considered as an anonymous subject. The server constructs an anonymous identity.
(R3) Data Minimization: XACML implements attribute-based access control, meaning that it grants access based on a set of attributes owned by a Data Subject. Upon making the access control decision, the PDP requests attributes and the Data Subject is assumed to provide the PDP with all its attributes. The PDP makes its decision based on its access control policy. Since the access control policy is not known to the Data Subject, there is no direct support for data minimization.
(R4) Strong Policy Binding: XACML does not present a straightforward feature that allows to attach policies to a data.
(R5) Data Retention: The Web services Profile for XACML, a proposed feature for XACML v3.0 defines an attribute describing the data retention period with respect to stored data in Web services.
(R6) Expressing Regulatory Constraints: XACML makes it possible to translate legal policies into machine-readable policies.
(R7) Access Control Rules: XACML supports obligations as actions to be performed by the service provider after an access has been granted and hence supports access control.
(R8) Delegation of Rights: XACML v3.0 introduced the ability to verify delegate rights. It allows an entity to delegate all or parts of its administrative rights to a resource to another entity under certain constraints defined by the delegator in a delegation policy.
(R9) Auditability: XACML does not present straightforward auditability features. We can however envision extending XACML to enable the audit feature by designing an XACML obligation that requests or triggers an audit based on an event. For instance, **on** *sharing data to third party*, **do** *audit*.
(R10) Controlling Policy Disclosure: Access control policy disclosure has been a subject of research for many years especially in the domain of Web services. Although there is no standard for XACML policy disclosure yet, related work for the PrimeLife Policy Language [ABDCDV$^+$09] shows

that XACML can be extended to control the amount of information revealed about the access control policy.
(R11) Reporting and Notifications: XACML does not present straightforward reporting and notification features. However, XACML Obligations element can express a requirement for the PEP to perform report and notification sending. For instance, an obligation can be: *send email notification to data owner of X in case of access granted to resource X*.
(R12) Redress Policies: XACML does not provide any approach to address the issues of remedies and redress.
(R13) Revocability: There is currently no support to address the issue of how to remove or revoke a policy in XACML.
(R14) Logging: XACML does not have a dedicated way to provide a logging feature. However, XACML Obligations enable the logging requirement. For instance, we can specify in a obligation attribute to log all granted access to a resource by keeping the resource ID the access requester name, the time of the access, etc. To do so, an obligation can be defined as a new element in XACML.
(R15) Controlling Data Location and Transfer: XACML does not provide data location in the language specification. However, XACML may be extended to support a data location feature and to describe the allowed locations of data. For instance, a new attribute such as `resource:location` can be considered.
(R16) Usage Control Rules: There is no support for usage control obligations.

XACML provides a standard to express access control policies. An advantage of this language is that it presents many extensible points that can serve to express accountability requirements. However, the definition of XACML obligations lacks support for privacy and usage control obligations.

A.2 The PrimeLife Policy Language

The PrimeLife Policy Language (PPL) [ABDCDV$^+$09] is an XML-based policy language proposed by the PrimeLife project [Pri11] that extends XACML. The language combines access and data handling policies. The goal of PPL is to make service customers aware of the conditions under which their data are handled. Therefore PPL gives service providers an automatic mean to define and manage privacy policies while applications are enabled to compare these service privacy policies with user privacy preferences.

Terminology

The language is symmetric: a similar syntax is used to express privacy policies and preferences. A PPL policy can be defined by a service provider to specify its privacy policies. In particular, the service provider defines in the policy how the collected data will be handled by the data controller and the entities the data is shared with. PPL makes it possible to automatically match these privacy policies with data subjects' privacy preferences. The outcome of the matching procedure generates a sticky policy or an annotated sticky policy that points out the difference between a user's preference and a controller's policy. This sticky policy is bound to the data and travels with the data. The sticky policy specifies statements on:

- **access control** which is inherited from XACML that PPL extends with privacy-friendly credential-based features. Conditions on access control describe how access to which resource under which condition can be granted.
- **authorizations**, that details actions that the data controller is allowed to perform with respect to the purpose of usage of collected data. In addition, authorizations enable to define the conditions of what in PPL specification [ABDCDV$^+$09] is called *downstream usage*[1]. This kind of authorizations are applicable for other data controllers, the downstream usage becomes the sticky policy for the data as it goes downstream.
- **obligations**, that the PPL specification [ABDCDV$^+$09] defines as "a promise made by a Data Controller to a Data Subject in relation to the handling of his/her personal data. The Data Controller is expected to fulfill the promise by executing and/or preventing a specific action after a particular event, e.g. time, and optionally under certain conditions".

PPL extends XACML with privacy-enhancing, credential-based and usage control features. The structure of XACML is preserved: PPL makes use of PolicySet, Policy, Rule, Target and Condition. PPL introduces new elements in XACML in order to enable the description of privacy policies. Such elements are the `CredentialRequirements` element that specifies the credentials that have to be presented in order to be granted access to a resource, the `ProvisionalActions` element that specifies the actions that have to be performed in order to satisfy an access control rule (e.g revealing users' attribute under certain conditions), the `DataHandlingPolicy` element that enable the Data Controller to express how the data received from the Data Subject will be treated, the `DataHandlingPreferences` element that enables the Data Subject to specify how its data has to be treated by the Data Controller, `Obligation` that specifies which specific actions to execute when given events occur (triggers) and `Authorization` that specifies the actions that are allowed to be performed.

Obligations in PPL differ from obligations in XACML. Indeed, XACML obligations specify the actions that the PEP has to perform when an access decision is made for the resource protected by the rule. A more detailed distinction is described in [Tra10].

Analysis according to our requirements

Since PPL extends XACML, PPL inherits the XACML properties. In addition, PPL provides enhanced features satisfying the requirements for an accountability policy language that XACML does not capture.
(R1) Capturing Privacy Preferences: PPL enables users to define their own privacy preferences. The users can declare which authorizations they wish grant to the Data Controller and which obligations he will have to enforce.
(R2) Anonymity/Pseudonymity: PPL integrates the concept of privacy-preserving credential-based access control by use of anonymous credentials. The decision whether access is granted to a requester depends on the possession of credentials that satisfy the requirements expressed in the access control policy. Anonymous credentials provide support for advanced features such as selective disclosure of attributes, thus preserving privacy. In addition, PPL provides an obligation

[1] Downstream usage specifies under which conditions data can be shared with other data controllers.

action `ActionAnonymizePersonalData` that expresses the obligation that personal data should be anonymized.
(R3) Data Minimization: The credential-based access control approach and the generation of a sticky policy at the data subject's side allows selective disclosure of attributes and cryptographic proofs of predicates over attributes with the help of anonymous credentials. These are seen as forms of data minimization
(R4) Strong Policy Binding: In PPL, the user's privacy preferences are evaluated against the service provider's data handling policies. An automated matching procedure checks whether a service provider's proposed data handling policy complies with a user's privacy preferences. In case of a positive match, a sticky policy can be associated with the data. In addition, the result of the automated matching procedure may give feedback to the users about the extent to which their privacy preferences are satisfied in the form of an annotated sticky policy.
(R5) Data Retention: Data retention periods can be expressed in PPL through the obligations that are using time-based triggers. For example the obligation to delete a piece of personal information within 7 days could be expressed using the provided trigger `TriggerAtTime` and action `ActionDeletePersonalData`. However, data retention for different usage purposes cannot be expressed. For example, one cannot express in PPL that a piece of data should be retained for 2 months for a particular purpose and 12 months for a different purpose.
(R6) Expressing Regulatory Constraints: PPL enables to express legal policies. In particular, the principle of purpose limitation (Art. 6.1 (b) of Directive 95/46/EC [Eur95]) can be expressed through data handling policies.
(R7,R16) Access and Usage Control Rules: PPL enhances XACML's obligation and authorization capabilities. The service providers (or the users) can specify access control and data handling policies (respectively preferences). PPL enables to express them in terms of:

- **authorizations**, that specify for which purposes the data can be processed, if the data is allowed to be shared downstream and what data handling policy must be followed by any downstream data controllers.
- **obligations**, actions that the Data Controller promises to carry out (specified in data handling policies) and the actions it is forced to perform (specified in data handling preferences). For instance, the Data Controller promises to delete the data after a certain period of time. This period of time can be specified in a data handling policy. The PPL framework also implements an Obligation Enforcement Engine that is in charge of enforcing obligations and an Obligation Matching Engine that compares obligations within a policy or a user's preference. An audit mechanism can then evaluate whether the obligation is indeed enforced to achieve usage control.

(R8) Delegation of Rights: PPL defines the authorization that makes it possible to specify if Personally Identifiable Information can be shared with the third parties. This type of authorization is mentioned as *authorization for downstream usage* in the PPL specifications. It allows a user to specify the access and usage control policies that the Data Controller has to enforce when forwarding the information to third parties. This procedure deals more with sharing data with third parties than with delegation of rights in access control.
(R9) Auditability: PPL does not present straightforward auditability features. We can however envision extending PPL to enable the audit feature.

(R10) Controlling Policy Disclosure: PPL supports the mechanism of *policy sanitization*: for each condition expressed in a policy, the policy author can specify a disclosure policy for that condition.
(R11) Reporting and Notifications: PPL provides a way to notify the Data Subject, for instance, whenever its data is accessed or shared. There exists an obligation action denoted `ActionNotifyDataSubject` that allows to express such requirement. However, this is very basic in PPL and we aim at extending this object in A-PPL.
(R12) Redress Policies: PPL does not provide any solution to address the issues of remedies and redress.
(R13) Revocability: PPL does not support revocation handling. It can be envisioned to have a PPL obligation `Revoke` that defines the promise to revoke some granted access or authorized usage of data within a certain period of time.
(R14) Logging: It is possible with PPL to log events such as the use of personal data for a specific use (e.g. marketing purpose). PPL provides the obligation action `ActionLog` that allows to specify which events to log and what is the acceptable delay. But this is very basic in PPL. In addition, PPL provides the obligation `ActionSecureLog` that logs an event and ensures integrity and authentication of origin of the event.
(R15) Controlling Data Location and Transfer: PPL does not provide data location restrictions in its vocabulary. However, it can be extended by creating a new attribute such as `resource:location`.

PPL presents a generic and user-extensible structure for authorizations and obligations which are more specific than what is defined in XACML. Indeed, the users are enabled to specify and add their own authorization and obligation vocabularies. Furthermore, the Obligation Enforcement Engine [ABP10] that makes sure that committed obligation as part of a sticky policy are indeed enforced may be extended with audit features to check compliance with policies.
Although the adoption of PPL is still limited today, since it has not been developed until recently (2011), PPL presents features that capture most of the identified requirements (see Section 1.1. Besides, PPL can be extended to provide further components that can address even more specifically those requirements.

A.3 Web Service Security Specifications

When we consider that IaaS, PaaS and SaaS can all be abstracted in terms of Web services, it becomes relevant to analyse what the standardised security specifications for Web services can provide in terms of accountability for Cloud computing. We describe below the major standards of the WS-* security standards family.

WS-Security

WS-Security [OAS06] defines a SOAP (Simple Object Access Protocol, a standard for message exchange for Web services) [SOA03] security header format containing security related information. A SOAP message may include multiple security headers. Each header is targeted at a specific SOAP actor/role that may be either the ultimate recipient of the message or an intermediary. Security headers may encapsulate one or more elements of the following types:

- Security tokens
- Signatures
- Encryption elements
- Timestamps

By providing a common syntax and a flexible processing model for security headers, this specification accommodates a large variety of security models and encryption technologies. Moreover, incorporating security features in the application level ensures end-to-end security.

WS-Trust

This specification provides a framework built on WS-Security for managing security tokens [OAS12]. In the WS-Trust trust model, a requester examines the policy associated with a Web service to identify the claims it needs. If the policy statements require security tokens that the requester does not possess, WS-Trust specifies a way of obtaining them: contacting a Web Service referred to as a Security Token Server (STS). A STS may also be used to renew, cancel and validate security tokens.

WS-Trust defines abstract formats of the messages used to manage security tokens. To each usage pattern corresponds a specific binding providing concrete semantics to the general security token requests and responses. For complex scenarios, WS-Trust describes flexible mechanisms for trust establishment. In fact, different STS may get involved to broker, exchange or delegate security tokens issuance. A general model for negotiation/challenge extensions is specified to support multi-messages exchanges for security tokens management.

The flexibility and extensibility of the specification allows interfacing with a large number of security models, including legacy protocols. In fact, increasing interoperability between trust domains is one of the purposes of this standard.

WS-Policy

The WS-Policy [BBC+06] is a policy expression language for describing the capabilities and requirements of a Web service, i.e. representing whether and how a message must be secured, whether and how a message must be delivered reliably or whether the request must follow a transaction flow. Such requirements are translated into machine-readable policy expressions that are usually provided by the Web service developer for the client component to automatically apply the requirements.

Basically, WS-Policy is a simple language that defines four elements (`Policy`, `All`, `ExactlyOne`, `PolicyReferences`) and two attributes (`Optional`, `Ignorable`) that suffice to create generic policy expressions by combining individual assertions. The policy assertions syntax are outside the scope of WS-Policy specifications. Thus, WS-Policy can be viewed as a meta policy composition language that can express any kind of requirements as long as the policy-aware clients (Web services endpoints and relays) are capable of understanding the specific syntax of the unitary assertions. An individual policy assertion expresses one requirement, behavior or capability related to (i) messaging (how the message must be built), (ii) security (how the message must be secured through authentication or encryption), (iii) reliability (how to ensure that the message has been sent/received) and (iv) transaction (what transaction flow must be followed to ensure that the transaction was committed completely and correctly).

WS-Reliability

WS-Reliability is a SOAP-based specification for reliable messaging requirements [OAS04]. WS-Reliability separates reliable messaging issues into a protocol (wire) aspect which deals with the horizontal contract between sender and receiver (e.g., message headers, choreography) and a quality of service (QoS) aspect which deals with the vertical contract between service providers and service users. The latter define a set of abstract operations on messages (such as `Deliver`, `Submit`, `Respond` and `Notify`). The specification assumes transparency of SOAP intermediaries and support for message integrity (e.g., as in WS-Security).

WS-Agreement

WS-Agreement [ACD+] is an XML based language and protocol designed for advertising the capabilities of Web service offers, creating agreements based on initial offers, and for monitoring agreement compliance at run-time. Agreement templates facilitate discovery of compatible agreement parties. As it is a standard by the Open Grid Forum, its design has mostly been motivated by QoS concerns, especially in relation to load balancing.

The specification is given in three main interconnected sections: (i) a schema for specifying an agreement, (ii) a schema for agreement templates and (iii) a set of port types and operations for managing the life cycle of the agreements (creation, expiration and monitoring of agreement states).

Albeit the fact that the definition of the protocol does not target specific types of Web services (thus potentially addressing cloud computing services) allowing for the negotiation of QoS terms among them, there is no support for assigning accountability in this standard (or the related concerns such as liability, responsibility, etc.).

Analysis according to the accountability requirements

These standards have a low abstraction level with respect to accountability, i.e. they deal with specific technical points about Web service security, but at the same time, they are generic enough to support accountability requirements for cloud services, specially in the management of evidences for non-repudiation and obligation execution. However, none of the presented WS-* security standards supports our identified accountability requirements and only R9, R10, R12 and R14 could be ensured by extending the WS-* security standards.

A.4 Unified Service Description Language

The Unified Service Description Language (USDL) [BO12], is a specification language to describe services from the business, operational and technical perspectives. USDL's ambition is to play a major role in the Internet of Services to describe services advertised in electronic marketplaces.

Some publicly-funded research projects have explored USDL, for instance THESUS, SLA@SOI, and currently FI-WARE [2] is using it as a federating initiative to foster industrial adoption. USDL aims at formalising in a structured manner non-technical aspects of a service description, for instance legal requirements, pricing information, and quality of service.

The language has been tested in the context of service marketplaces as use cases[3] [4] and therefore, since USDL generalizes the notion of "service", this language is being applied to suitably describe Cloud service provisioning chains.

Service specifications in USDL contain three main aspects: business, operational and technical (BOT) perspectives, in contrast with Web services and SOA, where the emphasis is on technical and implementation aspects only. Our analysis of the language model [CBMK10] is summarised below.

Analysis according to our requirements

(R1) Capturing Privacy Preferences: There is no explicit support for this. Perhaps we can assume that the technical module can support these, by determining a standard to cover privacy aspects, but this is unclear.
(R2) Anonymity/Pseudonymity: No explicit support.
(R3) Data Minimization: There is no support.
(R4) Strong Policy Binding: There is no support.
(R5) Data Retention: There is no explicit support, but in general, obligations and other legal requirements can be captured in SLA modules.
(R6) Expressing Regulatory Constraints: Some research effort has been performed in capturing legal aspects by using service descriptions[5], however the focus was on copyright, usage rights, and licensing topics, taking the point of view of the service provider, mostly. We can imagine that this module can be extended to support regulatory constraints and obligations.
(R7, R16) Access and Usage Control Rules: There is no explicit support for this. Perhaps we can assume that the technical module can support these, by determining a standard to cover privacy aspects, but this is unclear.
(R8) Delegation of Rights: There is no explicit support.
(R9, R14) Auditability and Logging: The service level module of USDL can describe monitoring needs agreed between service providers and its consumers. It allows for instance to model service level attributes: constants, variables, metrics and their relationships using arithmetic and logic operands. The language also allows to capture the logging characteristics in the functional module of the service description.
(R10) Controlling Policy Disclosure: There is no support.
(R11) Reporting and Notifications: The language has no explicit constructs for reporting and notifications, but these requirements can be accommodated in the functional service description, and/or in the service level model for the service. Since modeling obligations related to personal data handling is not the focus of USDL, the specification of notifications for security breaches need to be made in an ad-hoc fashion.

[2] `http://www.fi-ware.eu/`, last visited on 21/11/2013
[3] `http://www.internet-of-services.com/index.php?id=288&L=0`, last visited on 21/11/2013
[4] `http://www.linked-usdl.org/node/227`, last visited on 21/11/2013
[5] `http://www.internet-of-services.com/fileadmin/IOS/user_upload/pdf/USDL-3.0-M5-legal.pdf`, last visited on 21/11/2013

(R12) Redress Policies: SLAs in USDL can capture violations and the definition of penalties or compensations. As for most of the state of the art to date, there is no explicit support for a more elaborate notion of redress policy than these penalties.
(R13) Revocability: There is currently no support in USDL.
(R15) Controlling Data Location and Transfer: Among the most fundamental and universal elements (which are independent of, but relevant to the service description) are Time and Location. They are used in the context of service description, for instance, to express temporal and geographical availability, i.e., the time when and location where a service can be requested and delivered. The language does not bind service delivery location to data location. It is possible to describe in USDL many different sorts of non-functional properties and constraints, such as pricing, licensing, temporal availability, geospatial availability, dependent resources with necessary capabilities, and so on. We can then specify data transfer constraints in this way. However, there is no supporting technical mechanism to monitor and enforce these.

USDL may become a relevant standard for Cloud services in the near future. On the other hand, it would demand several extensions to accommodate accountability concepts. These extensions may not be straightforward to implement, as the USDL meta-model is already considerably complex, and that will require consultation with multiple entities taking part in the USDL standardization efforts.

A.5 SLAng

SLAng was developed in the EU project TAPAS [LSE03] and was an early initiative to formalize Service Level Agreements (SLAs). SLAng was designed to capture the mutual responsibilities between a service provider and a service consumer. The responsibilities are expressed in terms of Quality of Service (QoS) attributes, such as availability, throughput and delay. The language was designed to be used as input for automated reasoning systems, providing a format for negotiation of QoS properties that are to be included in contractual agreements. SLAng consists of three general components; service description (e.g., service location and provider information), contract statements (e.g., duration and penalties) and service level specifications (i.e., technical QoS descriptions and associated metrics).

SLAng is an XML-based language and provides a formal language with well-defined syntax and semantics to describe service level specifications. The language was designed for creating specifications at different service levels (application, application services and system resources). The individual agreements could then be composed into an overall service level agreement. More specifically, SLAng supports both horizontal SLAs (between collaborating partners providing the same kind of service) and vertical SLAs (for providers that rely on support from underlying infrastructure). In this respect, the SLAng approach is suitable for Cloud computing ecosystems where services from different service providers can be composed and delivered as a single service to the cloud consumer.

SLAng was designed with five key concepts in mind: parametrization (the language includes a set of parameters that quantitatively describe the service), compositionality (service providers are able to compose SLAs in order to issue new service offerings to customers), validation (the participants are able to validate the syntax and consistency of SLAs), monitoring (the SLA provides the basis for automatic monitoring of the agreed service levels) and enforcement (the SLAs forms the basis for automatic enforcement of the agreed terms). These concepts are highly relevant for A4Cloud; however, since SLAng is designed for agreements on QoS attributes, it is not directly applicable as it is. As shown in [MBJ^{+}13], SLAng can be extended with a domain specific language that describes the security and privacy policies agreed upon by a Cloud provider and its service.

Analysis according to the accountability requirements

Even though SLAng was designed for dealing with machine-readable agreements for non-functional properties of services, it fails to meet almost all identified requirements. It's basic construction only partly supports the auditability requirement (R9), but focus is on QoS attributes rather than security and privacy concerns.

A.6 P3P

The Platform for Privacy Preferences (P3P) [P3P] is an XML-based standard for privacy preferences and data collection policies. P3P provides a standardized format for expressing privacy policies and a protocol that enables web browsers to read and process the policies automatically. By using P3P, service providers can specify data collection policies that include the type of data, the type of use, the user of the data, the purpose of the use and how long the data will be retained. User agents can then fetch the policies, interpret them and present them to the end-user. The purpose is to assist the end-users in determining whether a particular service provider's published privacy policy matches the user's individual privacy preferences. The user agent can also warn the user in the case of a mismatch between the privacy policy and the user's preferences. Such user agents can be built into web browsers, browser plug-ins or proxy servers. They can also be implemented as a part of electronic wallets, form-fillers or any other user data management tool. Users therefore do not need to read the privacy policies at every site they visit or for every service they access.

The goal of the P3P project was twofold. It allowed service providers to formulate and present their privacy policies in a standardized, easily-located and machine-readable manner. It also helped the users understand how their personal data will be collected and used. The work on P3P started in the 1990s, and represented a great step forward in the quest of automating the process of communicating data management practices. However, the standard was also heavily criticized. Large parts of the critique was based on its limited scope; the P3P specification only covers notice/awareness (fetching and displaying electronic policy files) and choice/consent (specifying user preferences and matching them with privacy policies).

Terminology

A P3P policy is written as a sequence of statements (elements) that includes several sub-elements:

- **Purpose** is used to indicate why the data is being collected. P3P includes several predefined values, for example current (to complete and support the activity for which the data was provided) and pseudo-analysis (to infer, e.g. habits and interests of individuals without identifying specific individuals).

- **Recipient** indicates who will use the data. P3P includes six predefined values, for example ours (ourselves) and same (legal entities following our practices).
- **Retention** indicates how long the collected data will be kept. P3P contains five predefined values, for example stated-purpose (discard the data at the earliest time possible) and indefinitely (keep it forever).
- **Data-group** indicates the individual data items that will be collected for the stated purpose

P3P expresses the service providers data collection policies. To complement P3P, the preference language APPEL [CLM02] was designed, which allows the user to express his/her privacy preferences in a machine-readable way. Similarly to P3P, an APPEL preference file consists of a set of machine-readable rules that can be matched against P3P policies. APPEL can also be used to specify actions that will be taken in case the policy does not match the preferences. Meland et al. [MBJ+13] show how the APPEL language can be used to place restrictions on how personal data stored by a service provider will be used, in terms of *purpose*, *recipient* and *retention*.

Analysis according to our requirements

(R1) Capturing Privacy Preferences: This is the core of P3P and hence is supported.
(R2) Anonymity/Pseudonymity: Not supported by P3P.
(R3) Data Minimization: Supported by P3P through the use of the purpose element, which allows data subject to restrict the collection of data depending on how it will be used.
(R4) Strong Policy Binding: Not supported by P3P.
(R5) Data Retention: Data retention can be specified in the policy. **(R6) Expressing Regulatory Constraints:** Not directly supported by P3P, but may be supported by an extension.
(R7) Access Control Rules: Not supported by P3P.
(R8) Delegation of Rights: Not supported by P3P.
(R9) Auditability: P3P partly supports auditing, in terms that it is possible to specify what actions are allowed to be performed on what kind of data. Even though the scope of actions and data types are fairly limited, it can be extended to support further features.
(R10) Controlling Policy Disclosure: Not supported by P3P.
(R11) Reporting and Notifications: P3P allows to specify notifications that will be displayed to the user when there is a mismatch between a privacy policy and the user's preferences, however the notification will only be generated when the policies and preferences are compared (for example during the contract negotiation phase). The language does not provide any detection mechanisms or any way to generate evidence and therefore cannot generate notification of incidents that occur when the service is already running.
(R12) Redress Policies: P3P does not support redress.
(R13) Revocability: Not supported by P3P.
(R14) Logging: P3P does not support logging.
(R15) Controlling Data Location and Transfer: The current version of P3P does not support expressing rules about data location, but the language could be easily extended to cover this part.
(R16) Usage Control Rules: Not supported by P3P.

To summarize, P3P can be used for expressing end users' privacy preferences as well as the service providers' policies for data collection from end users. The language and the protocol are intended to be used for automatic matching of service terms with user preferences and could possibly be used in Cloud procurement phases. As with any otherlanguage, there is no support for run-time monitoring or enforcement of privacy policies. Since the language is designed for the web pages, its current scope is fairly limited. However, the language can be easily extended to cover data collection preferences and policies in other domains such as the Cloud. An example of how to extend P3P is provided in [ZGFvS].

A.7 SecPal4P

Becket et al. [BMB10] introduced the SecPAL4P declarative formal language for specifying user preferences and services' policies on treating collected Personally Identifiable Information (PII), together with a matching algorithm and a protocol for disclosing PII between parties. This is an extension of the SecPal language which is summarized in Section A.10. The goals are similar to those of P3P[6], EPAL[7], (more details about EPAL are given in Section A.14). and PPL, but the approach differs in the following: i) the language and the matching algorithm has formal semantics; ii) the language is more concise and close to natural text and iii) it supports obligations with respect to service providers, third-party information disclosure restrictions and delegation of authority.

During interaction between a user and a service, the service requests some PII from the user and provides him with his privacy handling policies, written in SecPAL4P. The user, instead has a set of privacy preferences, also expressed in SecPAL4P, that are automatically matched to service's policies. In case of success, the PII disclosure is allowed, otherwise denied.

Terminology

User preferences consist of service permissions and obligations; policies, in turn, define possible and promised service behaviors: both of which are expressed in form of assertions and queries in the SecPAL logical language, extended with two modal verbs: *may* and *will*.

For example,

$$\langle Usr\rangle\ says\ \langle Svc\rangle\ may\, use\, FaxNo\ for\ Contact$$

is an assertion expressing a user's $\langle Usr\rangle$ preference on the use of his fax number $FaxNo$ by service $\langle Svc\rangle$. During an encounter the placeholders are automatically instantiated to specific names. Here, $useFaxNoforContact$ is an example of a PII-relevant service behavior expressed using a domain-dependent vocabulary.

$$\langle Usr\rangle\ says\ \langle Svc\rangle\ may\, use\, Email\ for\ Marketing?$$

is a query asking for permission to use the PII.

An obligation is expressed using the *will* verb. e.g.

[6]Platform for Privacy Preferences (P3P) Project, `http://www.w3.org/P3P/`, last visited on 23/12/2013
[7]Enterprise Privacy Authorization Language, `http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/`, last visited on 21/11/2013

$$\exists\ (\langle Svc\rangle\ says\ \langle Svc\rangle\ will\ delete\ Email\ within\ t?\ \wedge\ t \leq 2\,yr)$$

is a query defining the user preference for handling PII (detention period), and the following is the corresponding assertion (obligation) from the service provider

$$\langle Svc\rangle\ says\ \langle Svc\rangle\ will\ delete\ Email\ within\ 1\,yr$$

Analysis according to our requirements

SecPAL4P could be potentially used for expressing privacy preferences (R1), data retention period (R5) and controlling data location (R15). In addition, its language, close to the natural one, seems to be a good option for modeling regulatory and contractual requirements (though it will need some extension).

A.8 Ponder

The Ponder Policy Specification Language [DDLS01] is a language for specifying policies for management and security in distributed systems. Ponder allows for specifying policies over a wide range of applications: network, storage, systems, application and service management. Ponder can also be used for security management activities such as registration of users or logging and auditing events for dealing with access to critical resources or security violations.

In addition to composition and reusability features, Ponder provides flexibility and extensibility while maintaining a structured specification that can be, in large part, checked at compilation time, thanks to its object-oriented features. They support user-defined types of policies to be specified and then instantiated multiple times with different parameters. Meta-policies in Ponder provide a very powerful tool that determines application specific constraints (a kind of attribute based conditions) on sets of policies. The design of Ponder can be of much inspiration for an accountability policy language, mainly because it was conceived for distributed systems and it can easily be extended.

Analysis according to our requirements

Ponder satisfies a number of the accountability requirements identified in Section 1.1, such as delegation of rights (R8), auditability (R9) , reporting (R11), revocability (R13), logging (R14) and partially satisfies access and usage control (R7 and R16). In addition, Ponder offers a filtering mechanism (similar to views in databases) which could be extended to satisfy the anonymity requirement (R2) and scriptable obligation policies allowing to specify actions to be executed that could be extended to satisfy the data retention period requirement (R5).

A.9 ConSpec

Aktug and Naliuka [AN08] introduced the ConSpec language for contract specification. In ConSpec, an application is associated with a formal contract which should be checked for compliance with the user policy. This checking can be either static, at installation or dynamic, and hence a mix of the two is proposed. When the compliance of some application contracts can be statically verified then it is signed by a trusted third party. There are situations where runtime monitoring is needed to enforce the policy. In such a case, the solution is to use reference monitors to control the execution, either with a parallel process or by in-lining control actions in the application. As a matter of fact, the ConSpec language allows to express formal security requirements as well as security on running applications. It is based on automata formalisms, but it is restricted to a deterministic formalism to enable automated verifications. For instance it does not allow arbitrary types and checking containment language is decidable. It also permits to express execution history and its features depend on the lifecycle stage. ConSpec defines a policy language and an event clause syntax which describe security actions. Security events are denoted by method signatures. The policy semantics are based on a class of security automata.

Analysis according to the accountability requirements

The ConSpec language supports a number of the requirements identified in Section 1.1, such as capturing privacy preferences (R1), data minimization (R3), data retention period (R5) and access and usage control rules (R7 and R16),

A.10 Security Policy Assertion Language (SecPAL)

The Security Policy Assertion Language [BFG10] (SecPAL) expresses decentralized authorization policies in a declarative and logic manner. SecPal is not a standard, but provides a rather intuitive way to write authorization queries and delegations of rights. It also provides a decision strategy which is proved terminating and sound. This strategy is based on a translation into Datalog with constraints. SecPAL does not address accountability. Our focus is on usage control and accountability; we expect to provide a domain specific language to provide assistance in writing obligations. One major concern is to express permission, obligation, prohibition as well as past and future constraints. We share with SecPAL the objectives to provide a simple language with an intuitive semantics. The constraints introduced in SecPAL can make less natural the writing of quantified expressions like ”everybody can send data d”.

A.11 A Formal Privacy Management Framework

In [LM09] the authors present a formal privacy framework which is a part of the PRIAM [8] project. The framework proposes a restricted natural language SIMPL (SIMple Privacy Language) to express privacy requirements and commitments. It also defines an event specification language based on communicating agents with a trace semantics. The formal language introduces a really small set of primitives to express data disclosure, request for disclosure, data collection and transfers. The SIMPL language is expected to be manually translated into event specifications. The framework formally defines the notion of an agent with a trace and relies on a notion of sticky policy. This enables to formally define a notion of global correctness that states how data are appropriately protected by the controllers. SIMPL statements are described in purely natural language, we think

[8] Privacy Issues and AMbient intelligence

useful to have an intermediate language less abstract but still human readable and easier to translate than natural language. This is also a formal approach but without a notion of refinement (that is a concrete way to generate operational sentences from formal ones), hence it does not help in the enforcement task. The paper mentions accountability but, except the privacy part, does not propose feature for auditing and rectifications.

A.12 Deriving Semantic Models from Privacy Policies

In [BA05] the authors present a method, a language and a concrete analysis to structure and compare data privacy purposes. Precisely, they describe a general process for developing semantic models from privacy policy goals mined from policy documents. They also provide a method enabling quantitative and qualitative model analysis including the ability to compare policy statements. The considered data set has 1200 goal statements extracted from nearly 100 Internet privacy real natural sentences. The minimal model components are actor, action, and object. The actor has a general capability to perform the action with respect to the object. But the model can nest an auxiliary model attached either to an action or to an object. The semantic model is able to express Boolean constructions (negation), permissions and obligations. The authors succeed in formally modelling 87 of the 100 most frequent privacy goals. The identified limits are about values on continuous domains and temporal relations. The authors quote that, Bandara et al. [BLMR04] have noted the need to derive enforceable policies from high-level goals and that purposes goals are difficult to compare but it is an essential activity. The proposal lacks of complex Boolean constructions and also temporal constructions but gives some constructions a privacy language should provide.

A.13 Logic-Based Regulation Compliance-Assistance

Kerrigan and Law argue, in [KL03], that information technology, if properly designed and developed, has the potential to mitigate and help solving many of the issues raised by the understanding of contracts. They develop an approach where contracts are represented by XML documents enriched with logic metadata. These metadata, first-order logic expressions, are exploited to check the compliance via a theorem prover (the Otter prover[9]). hence, the use of logical tools to check compliance and to solve other related questions for obligations seems not only possible but also desirable.

A.14 EPAL Translation of the The Freedom of Information and Protection of Privacy Act

The Enterprise Privacy Authorization Language (EPAL[10]) is a language defined by IBM. The proposal was submitted to the W3C consortium but not approved so far. The goal of [PAW04] was to test the expressiveness of EPAL against the Ontario Freedom of Information and Protection of Privacy Act (FIPPA[11]). Thus the resulting EPAL policies can be viewed as a validation of EPALs

[9] `http://www.cs.unm.edu/~mccune/otter/`
[10] `http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/`
[11] `http://www.e-laws.gov.on.ca/html/statutes/english/elaws_statutes_90f31_e.htm`

capabilities. EPAL is a purpose based language, extension of XML, and designed to make it easy to translate human readable privacy laws and privacy policies into machine readable policies. It is comparable to XACML but restricted to data privacy. Privacy policies and privacy legislation, as with EPAL, extend the traditional access control rules with three notions: Data subject, purpose, and obligations. Data subject is classic in Data Privacy rules, purpose is the intention of the access and obligations are actions which must be realized after the access. An EPAL policy is a vocabulary and a set of rules. The language allows for hierarchy of policy rules, and access under conditions. The document presents the reformulation of FIPPA rules and a manual translation into EPAL and also a disclosure test which concludes that the formulation is valid. There is also notions of permission and prohibition and the use of a user defined vocabulary on which purposes or usage controls expressions can be built. However, temporal constructions are lacking in EPAL and there is also nothing about the formal semantics and the compliance checking.

A.15 Experiences in the Logical Specification of the HIPAA and GLBA Privacy Laws

In [DGJ$^+$10] the authors provide a formal language (PrivacyLFP) to express privacy laws and a real validation on the HIPAA[12] and GLBA[13] sets. The proposed logic is a first-order logic with fixed point operators expressing past, future and real-time notions with a trace based semantic model. The authors first discuss the structure of privacy laws and emphasize the use of positive and negative clauses. They expose several concepts for privacy laws: structured data and attributes, dynamic roles, purposes of uses and disclosures, principal's beliefs, and past and future obligations. This results in a formal language with predicates, Boolean constructions, universal and existential quantifiers and two fixed point operators. The paper finally discusses some hints for policy enforcement and the main idea is to have an accountable system with logs and audit. The approach is not dedicated to expressing accountability obligations. This is a formal language not sufficiently close to natural languages. Full first-order temporal logic is not decidable (see [Fis08]) and restrictions like linear temporal logic with past and propositional quantification would be a stricter but a better target. Nevertheless, this work demonstrate that a formal specification of real laws are possible with a temporal logic.

[12]Health Insurance Portability and Accountability Act [Con02]
[13]Gramm-Leach-Bliley Act [Con99]