Abstract Accountability Language*

Walid Benghabrit[1], Hervé Grall[1], Jean-Claude Royer[1], Mohamed Sellami[1], Karin Bernsmed[2], and Anderson Santana De Oliveira[3]

[1] Mines Nantes, 5 rue A. Kastler, F-44307 Nantes, France
{firstname.lastname}@mines-nantes.fr
[2] SINTEF ICT, Strindveien 4, NO-7465 Trondheim, Norway
Karin.Bernsmed@sintef.no
[3] SAP Labs France, 805 avenue du Dr Donat Font de l'Orme
F - 06250 Mougins Sophia Antipolis, France
anderson.santana.de.oliveira@sap.com

Abstract. Accountability becomes a necessary principle for future computer systems. This is specially critical for the cloud and Web applications that collect personal and sensitive data from end users. Accountability regards the responsibility and liability for the data handling performed by a computer system on behalf of an organization. In case of misconduct (e.g. security breaches, personal data leaks, etc.), accountability should imply remediation and redress actions. Contrary to data privacy and access control, which is already supported by several concrete languages, there is currently no language supporting accountability clauses representation. In this work, we provide an abstract language for accountability clauses representation with temporal logic semantics.

1 Introduction

On-line processing of personal data requires privacy assurance and transparency on how data protection principles imposed by regulatory frameworks are being implemented by service providers. In addition, data owners need to have means to legally hold service providers accountable for their data processing and usage. Accountability for computer systems, as defined in [1], can be viewed as a general posteriori control to ensure the enforcement of some announced promises. In the following, the term "clauses" refers to theses promises. Setting up an accountable system faces many challenges. First, there is no adequate language for accountability clauses representation; such clauses are generally stated in natural language, making it hard for computer programs to assert whether or not service providers are respecting their clauses. There is also a lack of automated assurance necessary to assert that a service provider's clauses are being carried out (for instance, the collection of events showing who created a piece of data, who modified it and how, and so on). In addition, effective and profitable use of on-line services relies on data transfer and storage across different services.

* This work has been partly funded from the European Commissions 7th Framework Programme (FP7/2007-2013) under grant agreement no: 317550 (A4CLOUD).

J. Zhou et al. (Eds.): IFIPTM 2014, IFIP AICT 430, pp. 229–236, 2014.

These services may be hosted at heterogeneous environments and using different representations for their accountability clauses. This heterogeneity makes it difficult to check clauses compliance in an automated way and the data-owner has no means to verify that her/his preferences are being respected.

Different machine readable representations of privacy obligations have already been proposed [2,3,4]. However they only cover the preventive aspects of accountability and do not offer constructs to represent the other aspects (i.e. auditing and rectification). Despite some formal work, like [5,6,7,8], there is not yet a concrete formal language close to legal or contractual texts for data privacy obligations. To express accountability clauses we propose in this paper an Abstract Accountability Language (AAL), which is devoted to expressing accountability clauses in an unambiguous style and which is close to what end-users need and understand.

The content of this paper is as follows : Section 2 presents background on accountability and related work. The syntax and semantics of AAL are presented in Section 3. We show the expressiveness of AAL in Section 4. Finally, we discuss our results and present directions for future work in Section 5.

2 Background on Accountability

In [1], the authors argue that the usual "hide-it-or-lose-it" perspective on information is dominating but not adequate in a world where information should be communicated to third-parties. Classic privacy means and access control are insufficient to guarantee the protection of privacy since the data can be duplicated on the Web and it is possible to infer accurate details from public information.

In the context of the EU A4Cloud project[1], we consider that accountability concerns data stewardship regimes in which organizations that are entrusted with personal data may be responsible and liable for collecting, sharing, storing and otherwise processing the data according to contractual and other legal requirements from the time the data are collected until when the data are destroyed. Legal accountability is a subset of accountability and covers accountability clauses imposed by laws and regulations. In this paper we use the term *clause* to denote anything an actor is required to do because of a promise coming from a contract or the legislation. This refers to a legal duty that the actor is forced to perform and it implies some sanctions for neglecting it.

The analysis of accountability done in the A4Cloud project identifies four roles related to the creation, storing and processing of data. These roles are already present in current regulations like "Directive 95/46/EC". *Data subject*: this role represents any end-user which has data privacy concerns. Thus the data subject is the original creator of a data and express preferences about the future management of his data. *Data controller*: it is legally responsible to the data subject for any violations of its privacy and to the data protection authority in case of misconduct. *Data processor*: this role is attributed to any computational actor which processes some personal data. It should act under the control of a

[1] `www.a4cloud.eu`

data controller. *Auditor*: it represents data protection authorities which are in charge of the application of laws and directives related to data privacy.

As said before, several languages for specifying privacy preferences and policies. Despite the fact that they provide means for expressing formal and verifiable clauses, they are not readable by lawyers or privacy officers who may be involved in an accountable system. In addition, as far as we know, there are neither tool nor method to assist the design and the analysis of an accountable system proposed with these models. This makes it difficult for end-users to evaluate the compliance of a set of clauses (i.e. policy) with a given data privacy regulation or to compare two policies. In the previously cited research, authors also propose an approach to validate the correctness of their clauses, which is inadequate for non software verification specialists.

This paper presents a language for accountability clause representation that is: *i) Close to natural language:* to be adequate to end-users who do not necessarily have skills in a certain policy language, *ii) Machine understandable:* to be enforceable and to offer means for implementing audit functionalities required for violations detection and evidence collection within an accountable system. *iii) Expressed in a formal language:* to promote its automatic compliance checking and verification, and *iv) Abstract:* to act as a pivot model between different accountability clauses representation models and as such promotes the interoperability of heterogeneous systems.

AAL is only the top part of the design of an accountable system; in [9] we present an end-to-end framework for accountability from natural clauses to concrete policy enforcement relying on our AAL language.

3 Abstract Accountability Language

To represent accountability clauses, we adopt the point of view of [10] with minor modifications. We consider that an accountability clause to be a triplet (uc, aa, rc). The informal meaning of such clause is: "*Do the best to ensure the usage control (`uc`), and if a violation of the usage is observed by an audit (`aa`) then the rectification (`rc`) applies*". Usage control covers classic access control but also data distribution (or data transfer). The audit part covers the detection, judgment and evidences collection steps[2]. Rectification subsumes punishment[2] and includes also sanction, remediation and compensation functionalities.

In contrast to [10] we consider not only clauses violators but also victims and not only punishment but also remediation and compensation for the victims. We focus here on accountability clauses as expressed in legal directives. To give a formal and rigid description of these clauses we define a formal and abstract language `AAL` for Abstract Accountability Language. In our abstract approach we assume that there is an implicit secure logging mechanism. These logs should be sufficient to provide effective detection of breaches and identification of violators. Means to secure logs and auditing is out of the scope of this paper, but the

[2] As stated in Section 2, prevention, detection, judgment, evidence and punishment are the five steps of accountable system.

interested reader can look at [11]. In the following, we introduce the syntax of AAL in Section 3.1 and its associated semantics in Section 3.2.

3.1 Syntax

We present in Listing 1.1 the syntax of a minimal AAL kernel.

```
1   AALprogram     ::= Declaration* Clause*
2   Declaration    ::= AgentDec | ServiceDec | DataDec
3   AgentDec       ::= AGENT Id TYPE'('AgentType*')' RS'('service*')' PS'('service*')'
4   ServiceDec     ::= SERVICE Id TYPE'('Type*')' [Purpose]
5   DataDec        ::= DATA Id TYPE'('Type*')' SUBJECT agent
6   AgentType      ::= "Subject"|"Controller"|"Processor"|"Auditor"
7   Clause         ::= CLAUSE Id ':' Usage [Audit Rectification]
8   Usage          ::= [Quant Variable]* ActionExp
9   Audit          ::= AUDITING [Usage THEN] agent.audit'['agent']' '()'
10  Rectification  ::= IF_VIOLATED_THEN Usage
11  ActionExp      ::= Action | NOT ActionExp | Modality ActionExp |
12                     Cond | ActionExp (AND|OR|THEN|ONLYWHEN) ActionExp
13  Exp            ::= Variable | Constant | Variable.Attribute
14  Cond           ::= [NOT] Exp | Exp ['=' | '!='] Exp | Cond (AND|OR) Cond
15  Action         ::= agent.service ['['agent']'] '('Exp')' [Time] [Purpose]
16  Quant          ::= FORALL | EXISTS
17  Variable       ::= Var ':' Type
18  Modality       ::= MUST | MUSTNOT | MAY | ALWAYS
19  Type, Var, Attribute, Id, agent, Constant, Purpose ::= literal
```

Listing 1.1. AAL Syntax

An AAL program is divided in two parts: declarations and clauses. The declaration part allows users to declare system agents (with theirs types, actions that they can performs called *provided* services `PS()` and actions they can uses called *required* services `RS()` (line 3)), services (defined by a name `Id`, arguments `Types`, and optionally a purpose specifying the context of their usage `Purpose` (line 4)) and data (with their types (line 5)). We use a first-order type system supporting sub-typing for the data types declaration[3]. The clauses part, as mentioned before, an accountability clause (line 7) is a triplet (*uc*, *aa*, *rc*).

- Usage control *uc*, is a combination of actions[4] and conditions applied on variables that can be quantified (line 8). We represent actions in the following form: `agent1.action [agent2](args)` (line 15) where: `agent1` (resp. `agent2`) is the agent using (resp. providing) the `action` and `args` the needed arguments. Optionally we can add a purpose and time[3]. The keyword `THEN` is used in the sense of implication, `exp1 THEN exp2` means that when *exp1* occurs then *exp2* must also occurs. The `exp1 ONLYWHEN exp2` construction means that *exp1* occurs only if *exp2* has been realized in the past.
- Audit actions *aa* are introduced by the `AUDITING` keyword and are expressed as `agent1.audit[agent2]()` (line 9) where: `agent1` is an auditor auditing `agent2`'s logs. Such an audit operation can be more complex therefore we extends it by adding usage control `[Usage THEN]`.
- Rectification *rc* is introduced by the keyword `IF_VIOLATED_THEN` (line 10) followed by an usage expression, that represents the actions to perform when the audit detects that the usage control *uc* has been violated.

[3] Due to space limitations, we do not detail it in this paper.

[4] Actions are prefixed by modalities : MUST for obligation, MUSTNOT for prohibition and MAY for permission.

3.2 Semantics

In order to interpret the language AAL, we resort to temporal logic [12]. Linear-time temporal logic (LTL) is a modal logic with modalities referring to time. Indeed we focus on an extension with quantified data, known as first-order temporal logic [13]. Precisely, the grammar is defined in Table 1. 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.

Table 1. Temporal Logic: Syntax

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

We will now describe the temporal operators. Assume that a finite sequence of events is given, as well as a position in the sequence. $\texttt{X}\psi : \psi$ is true in the next position (in the sequence of events), $\texttt{X}^{-1}\psi : \psi$ is true in the previous position, $\texttt{G}\psi$: for next positions, ψ is always true. Its dual (with respect to negation) $\texttt{F}\psi : \psi$ will be true at some time. Symmetrically, $\texttt{G}^{-1}\psi$ means that for previous positions, ψ has always been true; its dual (with respect to negation) $\texttt{F}^{-1}\psi$ means that ψ has been previously true.

Instantiation and Extension for Accountability. In order to instantiate the general framework presented in the preceding paragraph, we need to define the set of events and the set of atomic predicates. The set of events is defined as the set of messages exchanged. A message is a structure with four components: source (emitter), target (receiver), the service name and a data (the content). Such a message is written as: $source.service[target](data)$. The atomic predicates over the set of events are defined with patterns (terms with free variables) or with logical predicates (equalities, comparisons). Given a pattern φ, a message e and a valuation σ assigning closed terms to the free variables in φ, $\varphi(e)$ is satisfied if the term $\varphi[\sigma]$ is equal to message e. A predicate φ equal to a disjunction $\vee\varphi_i$ of patterns φ_i can be extended to a sequence of message in order to define a projection: if π is a sequence of messages, then $\varphi(\pi)$ is the sub-sequence of π containing all the messages e in π such that e satisfies some φ_i for some valuation, and only these messages. We add a new modality for accountability :

formula	$\psi ::= \ldots$	
	$\mid \texttt{Acc}(\varphi : \psi)(\varphi : \psi)$	(Accountability)

Intuitively, a sequence π satisfies proposition $\texttt{Acc}(\varphi_1 : \psi_1)(\varphi_2 : \psi_2)$ at position p if the sequence before p satisfies $\neg\psi_1$ when projected with φ_1 and the sequence

after p satisfies ψ_2 when projected with φ_2. More formally, given a sequence π and two positions p and q such that $(0 \leq p \leq q < |\pi|)$, we denote the subsequence of π starting at p and terminating at $(q-1)$ by $\pi^{p,q}$.

$$(\pi, p) \vDash \texttt{Acc}(\varphi_1 : \psi_1)(\varphi_2 : \psi_2) \overset{\text{def}}{\Leftrightarrow}$$
$$\big((\varphi_1(\pi^{0,p+1}), p) \vDash \neg\psi_1\big) \Rightarrow \big((\varphi_2(\pi^{p+1,|\pi|}), p) \vDash \psi_2\big)$$

Interpretation. We give the main elements for the interpretation of AAL accountability clauses. Let $(uc,\ aa,\ rc)$ an AAL clause with its three parts, the accountability clause is translated using the `Acc` modality $G(aa \implies \texttt{Acc}(\varphi_1 : uc)(\varphi_2 : rc))$. Actions are represented by messages : $source.service[target](data)$. The Boolean operators (`NOT,AND,OR`) and quantification (`FORALL,EXISTS`) are translated in a straightforward manner : `NOT` $\neg$, `AND` $\wedge$, ... The operator `MUST` is translated in F, `ALWAYS` in G and `MUSTNOT` translated in $G\neg$. The operator `THEN` is translated as an implies $\implies$. The `ONLYWHEN` operator is used for past and translated in $=> F^{-1}$. The `MAY` operator is interpreted as a conjunction of `MUSTNOT` with the idea that what is not permitted is forbidden. For instance, if $\mathcal{R}equired$ is the set of required actions for agent a, and $act \in \mathcal{R}equired$ then `MAY` $a.act$ is translated as $\bigwedge_{b \in \mathcal{R}equired \wedge \neg b = act}\ G\ \neg b$.

4 Validation : The Health Care Use Case

To validate the expressiveness and adequacy of AAL for accountability clauses representation, we extract clauses from a realistic use case documented in [14] and illustrate their representation in AAL. This use case concerns the flow of health care information generated by medical sensors in the cloud. Patients will be connected to wireless sensors that monitor their vital signs (e.g., movement, temperature, etc.). The sensor data will be transmitted to the cloud where they will be further processed and stored. Figure. 1 represents a component diagram for the use-case. In this design, involved actors are represented as interconnected components. The interactions between the components are made via interfaces representing the different services offered and used by the actors. In the following we present three accountability clauses and their representation in AAL.

Clause 1: The data subject's right to access, correct and delete personal data. This clause is statically ensured by the `AccesRightInterface` (noted `ARI*` in Figure. 1) interface. However, this preventive means cannot be sufficient since for instance the actor Y might not implement actions properly. Thus the hospital clause should be written in AAL, making explicit the audit step and the rectification that applies in case of violation.

```
EXISTS p:Patient EXISTS d:Data
  (d.subject = p) THEN (MAY p.read(d) OR MAY p.write(d) OR MAY p.delete(d))
AUDITING DPA.audit[hospital]()
IF_VIOLATED_THEN MUST DPA.sanction[hospital](...)
```

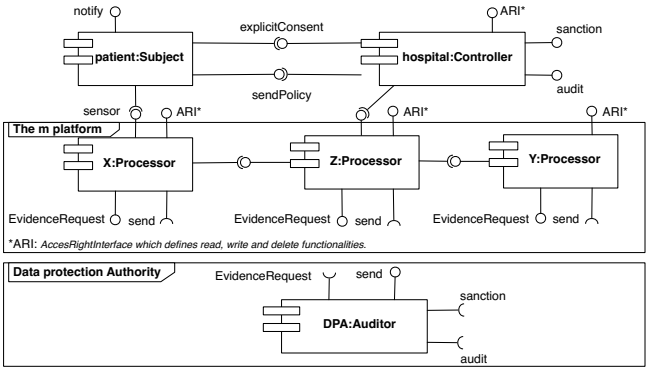

Fig. 1. Component diagram for the health care use case

Clause 2: The data subject's informed consent. Data subjects must consent to the processing of their personal data, before any personal data is collected about them. We assume that the hospital defines a specific protocol to get the explicit consent. In a first time the hospital sends its policy to the patient (`sendPolicy`) which then replies to the hospital with its consent or not (`explicitConsent`).

```
EXISTS p:Patient EXISTS d:Data
 ((p = d.subject) THEN MAY p.sensor[X](d)
    ONLYWHEN d.subject.explicitConsent[hospital]("true"))
 AND (MAY p.explicitConsent[hospital](b:Boolean)
    ONLYWHEN hospital.sendPolicy[p]("processing␣policy␣and␣purpose"))
AUDITING DPA.audit[hospital]()
IF_VIOLATED_THEN MUST DPA.sanction[hospital](...)
```

Clause 3: The data controller must notify the data subjects of security or personal data breaches. In case of a security or privacy incident that is related to the patients' personal data, Cloud providers X and Y must notify m platform, m must notify the hospital and the hospital must notify the patients. We assume that the hospital has been informed (or has detected) a violation with the `violation` action. The usage clause is below.

```
EXISTS p:Patient EXISTS d:Data
(MUST hospital.violation(d) AND (p = d.subject))
                THEN MUST hospital.notify[p]("data␣breach")
```

The three clauses presented above shows that AAL is capable of expressing the accountability requirements for all actors involved in the use case.

5 Conclusion

Accountability makes clear the responsibilities of data controllers, in particular in the case of data breaches, reinforcing trustworthiness in the cloud. We propose for the first time a domain specific language (DSL) to express rules close to sentences in laws, data directives and contracts. This language is equipped with a formal logical background and is the first stone towards clauses enforcement trough accountable design and verification. We demonstrate the expressiveness of AAL on a use case that includes real examples of clauses from data protection legislation. Our proposal defines conditions and event sequences which

are relevant to express complex chains of actions. The syntax of AAL is simple and abstract enough making it human readable and such it can easily be used by non specialist users. At the same time, AAL is based on a temporal logic semantics making it machine readable. Such semantics allows early expression simplifications, well-formedness checking and verification of expected properties. We start such a work in [15] using the mCRL2 model-checker. We are currently developing a Web based framework called `AccLab` [5], to support these concepts.

Accountability clauses written in AAL are quite close to natural language. However, there is still work to fill the gap with data protection legislation. The exact shape of AAL is not definitive since more experiments will be needed. We have started thinking on design and verification but one important aspect is to develop techniques for manual and automatic clauses enforcement.

References

1. Weitzner, D.J., Abelson, H., Berners-Lee, T., Feigenbaum, J., Hendler, J., Sussman, G.J.: Information accountability. Commun. ACM 51(6), 82–87 (2008)
2. DeYoung, H., Garg, D., Jia, L., Kaynar, D., Datta, A.: Experiences in the logical specification of the HIPAA and GLBA privacy laws. In: WPES 2010, pp. 73–82 (2010)
3. Le Métayer, D.: A formal privacy management framework. In: Degano, P., Guttman, J., Martinelli, F. (eds.) FAST 2008. LNCS, vol. 5491, pp. 162–176. Springer, Heidelberg (2009)
4. Piolle, G., Demazeau, Y.: Representing privacy regulations with deontico-temporal operators. Web Intelligence and Agent Systems 9(3), 209–226 (2011)
5. Etalle, S., Winsborough, W.H.: A posteriori compliance control. In: Lotz, V., Thuraisingham, B.M. (eds.) SACMAT 2007, pp. 11–20. ACM (2007)
6. Jagadeesan, R., Jeffrey, A., Pitcher, C., Riely, J.: Towards a theory of accountability and audit. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 152–167. Springer, Heidelberg (2009)
7. Feigenbaum, J., Jaggard, A.D., Wright, R.N.: Towards a formal model of accountability. In: NSPW, pp. 45–56. ACM (2011)
8. Zou, J., Wang, Y., Lin, K.-J.: A formal service contract model for accountable saas and cloud services. In: SCC 2010, pp. 73–80 (2010)
9. Benghabrit, W., Grall, H., Royer, J.-C., Sellami, M., Önen, M., Oliveira, A.S.D., Bernsmed, K.: A cloud accountability obligations representation framework. In: CLOSER (2014)
10. Feigenbaum, J., Jaggard, A.D., Wright, R.N., Xiao, H.: Systematizing "accountability" in computer science. Technical Report TR-1452, University of Yale (2012)
11. Vaughan, J.A., Jia, L., Mazurak, K., Zdancewic, S.: Evidence-based audit. In: IEEE 25th Computer Security Foundations Symposium, pp. 177–191 (2008)
12. Fisher, M.: Temporal representation and reasoning. In: Handbook of Knowledge Representation, pp. 513–550. Elsevier, Amsterdam (2008)
13. Hodkinson, I.M., Wolter, F., Zakharyaschev, M.: Decidable fragment of first-order temporal logics. Ann. Pure Appl. Logic 106(1-3), 85–134 (2000)
14. Bernsmed, K., Felici, M., Oliveira, A.S.D., Sendor, J., Moe, N.B., Rübsamen, T., Tountopoulos, V., Hasnain, B.: Use case descriptions. Deliverable, A4Cloud (2013)
15. Benghabrit, W., Grall, H., Royer, J.-C., Sellami, M.: Accountability for Abstract Component Design. In: EUROMICRO DSD/SEAA 2014, Verona, Italy (August 2014)

[5] `http://www.emn.fr/z-info/acclab/`