Towards the Specification of Natural Language Accountability Policies with AccLab: The Laptop Policy Use Case

Walid Benghabrit[1] and Jean-Claude Royer[1] and Anderson Santana De Oliveira[2]

[1] IMT Atlantique, site de Nantes, 5 rue A. Kastler, F-44307 Nantes, France
{firstname.lastname}@imt-atlantique.fr
[2] SAP Labs France, 805 avenue du Dr Donat Font de l'Orme, France - 06250, Mougins, Sophia Antipolis
anderson.santana.de.oliveira@sap.com

1 Introduction

Accountability allows to assign legal responsibility to an entity. It is the basis for many contracts, obligations or regulations, either for digital services or not. In our previous work we studied the Abstract Accountability Language for expressing accountability [BGRS15,RSDO16] with a logical focus.

We demonstrate the AccLab tool support over a set of policies from the Hope University in Liverpool governing the use of their IT systems and computer resources. These policies are representative of terms of use and other agreements, being really part of the University management, under control of the University Council. Moreover, this application context is more familiar to the general public than other domains, such as financial or health care services.

One important task is to analyze and interpret these policies which are written in a natural, sometimes legal style. Of course, during this analysis we found many ambiguities, omissions, inconsistencies and other problems, but our purpose is to demonstrate that a part of it can be formalized and automatically verified. This paper presents the laptop user agreement policy and discussed its specification with our tool support.

2 Use Case Introduction

The policy of interest is a set of seven texts from [Uni17] related to IT concerns and data protection. We will focus here on the laptop agreement. The laptop agreement is rather short (10 clauses), included below.

In accepting the use of a University laptop, I agree to the following conditions:

1. *I understand that I am solely responsible for the laptop whilst in my possession*
2. *I shall only use the laptop for University related purposes.*

3. *I shall keep the laptop in good working order and will notify I.T. Services of any defect or malfunction during my use.*
4. *I shall not install and / or download any unauthorized software and / or applications*
5. *I shall not allow the laptop to be used by an unknown or unauthorized person. I assume the responsibility for the actions of others while using the laptop.*
6. *I shall abide by the University Acceptable Use, Information Security and Portable Data Device security policies as published on the I.T. Services Website*
7. *Any work saved on the laptop will be deleted prior to the machine's return*
8. *If the laptop is lost, stolen or damaged, the incident must be reported to the University Secretary's Office within 24 hours*
9. *If the lost, stolen or damaged laptop and / or accessories is determined to be caused by negligence or intentional misuse, I shall assume the full financial responsibility for repair costs or fair market value of the laptop.*
10. *I am aware that any breach of these policies may render me liable to disciplinary action under the University's procedures*

In the remainder of this paper we will go through the exercise of formalizing and verifying this policy.

2.1 AAL

Our policies are written with the Abstract Accountability Language (AAL) and we will comment the main constructions. In our approach, we consider that an accountability clause should express three things: a *usage*, an *audit* and a *rectification*. The usage expression describes access control, obligations, privacy concerns, usage controls, and more generally an expected behavior. In the course of the design of AAL we reviewed several related languages, and our requirements are aligned with several points raised by [BMB10]. Expressiveness is ensured by negation, unrestricted set of actions, type hierarchies, conditions and policy templates. We also advocate for a readable language with succinct unambiguous semantics. In this context we introduce notions of permission, interdiction and obligation, but we do not have the exact concepts of deontic logic since we want to get our approach free from paradoxes and we aim at reusing classic logic with its tool support. Point to point communications with messages passing, used by many related work, is a good abstraction, simple and flexible. The audit and rectification expressions are similar to usage expression but dedicated to auditing, punishment and remediation. The audit expression defines a specific audit event which triggers the auditing steps. The rectification expression denotes actions that are done in case of usage violations. For instance, to punish the guilty party and to compensate the victim agent. We follow [Sch99,Mul00] which argue that punishments and sanctions are parts of accountability. AAL allows to define types with union, intersection, inclusion and negation to help in modeling data and roles. An action or a service call is represented as a message `sender.action[receiver](parameters)`. The language enables predicates (prefixed by @)

and if `Type` is a type, `@Type` denotes its associated unary predicate. Authorizations are denoted by the `PERMIT` and `DENY` keywords prefixing an action or an expression. The language provides Boolean operators, first-order quantifiers and linear temporal operators. We need to define policies and to reuse them thus we implement a notion of template (introduced by `TEMPLATE`) which improves readability and structuration. A template enables to name a policy with parameters (called with `@template`) but it also supports higher-order definitions helpful in defining common schemas of accountability or usage.

2.2 The Laptop Policy

As we can see in the policy from Section 2, sentences are often vague but sometimes they contain more precise information. It is the case in other policies like the data protection, data portable or information security policies. Thus we did a first reading of all the policies to extract some elements about the information system. From that, we conclude that the "I" is referring to a student or a staff of the university (named here `resp`) which is explicitly defined as of type `EligibleUser` in the IT usage policy. In fact another kind of person (`AllPerson`) appears in clause 5 which obliged us to reconsider clause 2 with a different meaning as other persons may use the laptop assigned to `resp`.

As a first simple example the clause 10 above states that "disciplinary action under the University's procedures" may take place in case of breach. This defines the policy rectification and we represent it as an abstract action in Listing 1.1 with a simple policy with a typed parameter. `LHU` is a constant denoting the University representative lawyer, it may be a fictitious or a real person.

Listing 1.1. Simple Rectification in AAL

```
TEMPLATE LHURectificationPolicy (resp:AllPerson)
       (IF (@EligibleUser(@arg(resp))) THEN {LHU.disciplinaryAction[@arg(resp)]()})
```

In the laptop policy there is no more information about rectification and nothing about audit. This last can be reduced to `auditor.audit[LHU]()`. As in the case of rectification once we have more details, the template construction allows to refine these definitions. A large section of the laptop policy describes in fact information about permissions, prohibitions, obligations and some conditions. Most of this usage policy will be described by a template named `laptopUA`, see Listing 1.2. Only our interpretation of few clauses is given but this policy covers clauses 1 to 9. The first line says that if a person is permitted to use a laptop then he is an eligible user and this laptop was assigned to him. We also assume that the eligible user should bring back his assigned laptop in the future to the university secretary.

Listing 1.2. Laptop Policy Agreement in AAL

```
TEMPLATE laptopUA(resp:AllPerson)(
(FORALL laptop:Laptop FORALL p:Purpose (IF (PERMIT @arg(resp).use[laptop](p))
         THEN {@EligibleUser(@arg(resp)) AND @assigned(@arg(resp), laptop)})) AND
(FORALL laptop:Laptop FORALL p:Purpose
       (IF (@EligibleUser(@arg(resp)) AND @assigned(@arg(resp), laptop))
            THEN {SOMETIME (@arg(resp).bringBack[LHUsecretary]())})) AND ...)
```

They are many things which are left implicit, are wrong or skipped in natural language policies. One important benefit is that problems and omissions appear during the specification phase or will be detected by the verification tools.

The laptop clause 2 is simple, however, it interacts with the clause 5 which leaves open the use of the laptop by a known and authorized person, maybe a colleague. But in fact we do not have information about which kind of person is permitted and for what kind of purposes. Other technical problems occur with the clause 5, 7 and 8: they need explanations, as we will see later. Finally, the overall structure of the laptop accountability policy is as in Listing 1.3. The accountability policy has two parts, the first is related to all clauses except clause 7 which is related to the second part.

Listing 1.3. Laptop Accountability Policy in AAL

```
TEMPLATE LaptopAccountabilityPolicy (resp:AllPerson) (
// from 1 to 10 but clause 7
@template(ACCOUNT, @template(laptopUA, @arg(resp)),
                   @template(LHURectificationPolicy, @arg(resp)))  AND
// from clause 7 since it is another schema
@template(ACCUNTIL, @template(condition7, @arg(resp)),
  @template(achievement7, @arg(resp)), @template(LHUWeakRectificationAndPay, @arg(resp)))})
```

The ACCOUNT and ACCUNTIL templates define accountability schemas. Let us describe the most classic which is quite similar to the one used in our previous papers, see Listing 1.4.

Listing 1.4. Basic Accountability Template in AAL

```
TEMPLATE ACCOUNT(UE:Template, RE:Template) (
 ALWAYS (auditor.audit[LHU]() AND (IF  (NOT(@arg(UE)))
                 THEN {ALWAYS (IF (auditor.audit[LHU]()) THEN {@arg(RE)})})))
```

This template assumes that the audit is simple and at each instant (in linear time) if the usage (`@arg(UE)`) is not satisfied then rectification (`@arg(RE)`) applies in case of an audit. The ACCUNTIL will be discussed later. Note that for clause 7 we diverge from the original text and we choose a weaker rectification to illustrate the language flexibility.

3 AccLab

Formal specifications are beneficial but they are really more effective if we have tool support and preferably some automated verification means. Thus we develop AccLab for Accountability Laboratory to experiment the specification, verification and monitoring of accountability policies. AccLab is compound from a set of tools which are: The component editor, the AAL editor and its verification, and the monitoring tools. The last release of AccLab is version 2.2 which was released on July 23, 2017 on github (`https://github.com/hkff/AccLab`) under GPL3 license. The AccLab IDE is a web interface that provides a component diagram editor and tools to work with the AAL language. The back-end is written in Python3 and the front-end in JavaScript based on `dockspawn` (`http://www.dockspawn.com`) which is a web based dock layout engine released

under MIT license. For verification purposes AccLab is interacting with the TSPASS tool ([LH10]), a prover for first-order linear temporal logic ($\mathbb{FOTL}$). The implementation is still in progress we will give an overview of its main current features. Some features like the full type constructions and the template are not fully operational thus we mix the use of AccLab and the TSPASS prover with some manual manipulations to process our example.

To manage more easily the AAL language a dedicated editor has been implemented. This editor is directed by the syntax and highlights the language keywords. There are syntactic checking but also semantic controls for type checking and the consistency of the declared services. A panel in the editor arranges a set of tools providing assistance in writing by the use of dedicated templates, for instance generating type declarations, accountability clauses or specific privacy expressions. This panel also contains few verification tools mainly the conflict checking with localization and the compliance checking. AccLab translates the AAL language into $\mathbb{FOTL}$ and the checking tools use the connection with TSPASS and its satisfiability algorithm. In case of unsatisfiability we implemented a mean to isolate the minimal core unsat. The tool provides macro calls which are useful in automating some complex tasks related to the translation to $\mathbb{FOTL}$ and the interaction with TSPASS.

One idea behind AccLab is to see accountability in action, and one way to achieve that is to be able to run simulations. AccLab includes a simulation module that allows it to monitor agents in a system and to observe accountability in action. We also proposed a tool called AccMon which reuses the above monitoring principles and provides means to monitor accountability policies in the context of a real system.

4 Lessons Learned and Discussion

This is an ongoing work but from now on we formalize several parts. One important task was to built and partly invent the information system and to get it consistent.

4.1 Accountability Schemas

The basic accountability scheme for a given usage is expressed simply as `(UE OR RE)`, or equivalently `IF (NOT UE) THEN RE`, where `UE` is the usage expression and `RE` the rectification expression. However, often we need quantifiers, for instance to identify the responsible user. We also consider time and our choice was to consider linear time as it seems sufficient in many cases. Thus we write formulas like `ALWAYS FORALL resp:Any IF (NOT UE(resp)) THEN RE(resp)`, or `FORALL resp:Any ALWAYS IF (NOT UE(resp)) THEN RE(resp)`. These are two distinct schemas, the first implies the second but the reverse is false (it is related to the Barcan formula).

Note that the above scheme allows to define first order accountability, that is the user is responsible and will be subject to rectification in case of a violation. But it is possible to define second order accountability, that is the processor or

implementer is responsible to enforce IF (NOT UE) THEN RE and in case of violation it is rectified with RE2. In this case, the schema will be (UE OR RE OR RE2), and more generally higher-order responsibility is defined as (UE OR RE OR ... OR REn). Templates are useful here to capture relevant accountability schemas.

4.2 Laptop User Agreement

We consider to have a full specification of the laptop agreement but it evolved since our first specification. We prove the satisfiability with most of the types and actions declarations and we also prove that violation of every clause will result in a rectification of the responsible person. As a general comment it is always possible to represent any kind of information but without semantics it remains purely syntactic. Of course aligning the syntax between different policies is a non obvious requirement. In our work we elaborate a rich information model resulting from the analyzing of most of the 7 policies. We have a type hierarchy with nearly 50 types and 50 relations, a set of 40 actions and more than 40 predicates and constants. This is a critical task because there is more or less nothing in the texts and sometimes they have some flaws. An important part is to add some behaviours to link together actions and predicates. Except type relations the action and predicate the behaviour is still poor but may be enriched later with the analysis of other policies.

The overall structure of the laptop policy has been given in Listing 1.3. We succeed in proving the satisfiability of the clauses and taking into account some user behaviours. The $\mathbb{FOTL}$ formula generation takes around 3s, the prover generates a total of around 1000 conjunctive normal forms in less than one second. We express several different user behaviours, for instance (see Listing 1.5) once a user signs the policy he gets an assigned laptop and accept the policy until he leaves.

Listing 1.5. A user behaviour

```
FORALL res:AllPerson FORALL laptop:Laptop
   ALWAYS (IF (resp.signed[LaptopAccountabilityPolicy]())
          THEN {(@assigned(resp,  laptop) AND @template(LaptopAccountabilityPolicy, resp))
                  UNTIL (resp.leave[LHU]())})
```

Indeed, the interaction between the proper user behaviour and the policy is a critical point for semantic reasons but it also brings some difficulties about quantifiers. The above formula is not monodic (this is a constraint for decidability of satisfiability) and we reformulate it with the laptop quantifier inside the accountability policy rather than in the user behaviour.

With the laptop clause 7 there is a technical point to discuss. The simplest formulation is to use an UNTIL operator, however this is more tricky to align with our ACCOUNT accountability clause which is based on a ALWAYS pattern. One solution is to use the principles of the separated normal form, for instance from [Fis11]. We can rewrite this expression as an ALWAYS clause without the need of the UNTIL operator. However, the result is not intuitive and only understandable by specialists of linear logic. We implement another solution that is to define a separate

accountability clause based on a different pattern. This pattern is `ACCUNTIL = (A UNTIL B) | (A AND NOT B) UNTIL NOT (A OR B)`. The second term of this pattern represents a part of the negation of the first, and then the case where rectification should happen. This pattern is not equivalent to the `A UNTIL B OR NOT (A UNTIL B)` scheme, but the difference is on the negative part. This difference is an infinite trace which is not really monitorable thus it can be discarded. Nevertheless, this behaviour introduces new quantifiers and the satisfiability process does not terminate. We reconsider it and succeed with a weaker specification (see Listing 1.6) and additional behaviour for the delete action.

Listing 1.6. Clause 7 specification

```
FORALL laptop:Laptop IF (@assigned(resp, laptop) AND resp.bringBack[LHUsecretary](laptop))
                     THEN {@deletedSavedWork(resp)}
```

The clause 8 introduces a notion of real time with "before 1 day", while we have a construction for that we choose to replace it with a `NEXT` construction. In fact there are only three such references in all the 7 policies. But during the verification step we realize that it is not sufficient because: It is acceptable that the user reports immediately and this is not acceptable that he reports after two states. Furthermore the `ALWAYS` pattern is not correct since the rectification may occur before the violation will be effectively realized. Thus, as for clause 7, a separate accountability pattern can be used, see Listing 2.

Listing 1.7. Clause 8 specification

```
FORALL laptop:Laptop
   (IF (@EligibleUser(resp) AND @assigned(resp,  laptop)
                        AND (@lost(laptop) OR @damaged(laptop) OR @stolen(laptop)))
      THEN {(resp.report[LHUsecretary](problem)
             OR (NEXT (resp.report[LHUsecretary](problem))))})
```

Another important point to note is related to clause 5 which states that "I assume the responsibility for the actions of others while using the laptop". The "I" is represented by `resp` in our usage policy and appears at the level of the accountability policy, see Listing 1.3. From that we can verify that if another user did a breach with the laptop assigned to `resp` then rectification is effectively applied to the responsible person. These verifications where successfully done, however, not without some technical difficulties.

4.3 Discussion

This is an ongoing work, as we need to complete our information model, to formalize other policies and at least to verify their consistency. The current natural language policies have many drawbacks: lack of precision, redundancies, ambiguities etc, unsurprisingly. Regarding accountability there are some information about monitoring and really few details about the remediation, compensation and punishment parts. One important task before any formalization of the policies is to build a consistent information model with sufficient relevant details about data and behaviour.

There are ambiguities about the status of some statements and our language's strength in clarifying them. For instance, in several policies (laptop, IT usage) there are sentences related to the fact that a person will sign and accept a policy. If we consider it as a part of the accountable usage it will say that to not sign is a breach which is not sensible. Thus it should be part of the proper user behavior, it is free to sign, and if he signs he will accept the responsibilities included in the policy. As we have seen, this impacts the structure of the specification but also brings some difficult points regarding the interaction between quantifiers and modal operators.

The use of $\mathbb{FOTL}$ or linear temporal logic often simplifies the specification and there is only few references to precise dates or dense time in the policies. However, there are several subtleties in writing formulas with both quantifiers and temporal operators. $\mathbb{FOTL}$ is quite expressive but the bottleneck is the tool support, there is only one, TSPASS, which is now not maintained. Another problem is that it relies on the monodic constraint which is constraining for the behaviour specification. Improvements are possible here but need important theoretical and implementation efforts. Another point is the use of the standard semantics based on infinite traces which is not suitable for real monitoring. The alternative is a translation into pure $\mathbb{FOL}$, the drawback is the explicit management of time parameters. These additional parameters may compromise the decidability of satisfiability but this logic has been intensively studied and a complete map of the decidable fragments exists. An adaptation of our language and its tool support is perfectly possible and it seems a sensible future perspective.

References

[BGRS15] Walid Benghabrit, Hervé Grall, Jean-Claude Royer, and Mohamed Sellami. Abstract accountability language: Translation, compliance and application. In *APSEC*, New Delhi, India, 2015. IEEE Computer Society.

[BMB10] Moritz Y. Becker, Alexander Malkis, and Laurent Bussard. A practical generic privacy language. volume 6503 of *ICISS 2010*, pages 125–139. Springer, 2010.

[Fis11] Michael Fisher. *An Introduction to Practical Formal Methods using Temporal Logic*. Wiley, 2011.

[LH10] Michel Ludwig and Ullrich Hustadt. Implementing a fair monodic temporal logic prover. *AI Commun*, 23(2-3):69–96, 2010.

[Mul00] Richard Mulgan. 'accountability': An ever-expanding concept? *Public Administration*, 78(3):555–573, 2000.

[RSDO16] Jean-Claude Royer and Anderson Santana De Oliveira. AAL and static conflict detection in policy. In *CANS, 15th International Conference on Cryptology and Network Security*, LNCS, pages 367–382. Springer, November 2016.

[Sch99] Andreas Schedler. *Self-Restraining State: Power and Accountability in New Democracies*, chapter Conceptualiazing Accountability, pages 13–28. Lynne Reiner, 1999.

[Uni17] Liverpool Hope University. IT services policies, 2017. https://www.hope.ac.uk/aboutus/itservices/policies/.