

Framework pour la responsabilisation

Walid Benghabrit

Mines Nantes, 5 rue A. Kastler, F-44307 Nantes, France
walid.benghabrit@mines-nantes.fr

Abstract

Nous assistons à la démocratisation des services du cloud et de plus en plus d'utilisateurs (individuels ou entreprises) utilisent ces services dans la vie de tous les jours. Dans ces scénarios les données personnelles transitent généralement entre plusieurs entités. L'utilisateur final se doit d'être informé de la collecte, du traitement et de la rétention de ses données personnelles, mais aussi ce dernier doit également avoir les moyens nécessaires pour tenir responsable le fournisseur de service en cas d'atteinte à sa vie privée. La responsabilisation (ou *accountability*) est la propriété d'un système ou d'une personne à être responsable de ses actes et de leurs conséquences. Dans cet article, je présente une vue générale d'un framework pour la responsabilisation permettant d'assister le développement d'un système responsable de la phase de conception jusqu'à l'implémentation.

Mots clés : Responsabilisation, vie privée, DSL, service, cloud, sécurité, génie logiciel.

Abstract

Nowadays we assist to the democratization of the cloud services, more and more users (individual or business) use these services in everyday life. In such scenarios our personal data transit generally between many actors. The end user must be informed on collecting, processing and retention of his personal data, but also he must be able to designate a service provider as a responsible in case of privacy violations. Accountability is the property of a system/person to be responsible for its actions and their consequences. In this paper, we present an overview of an accountability framework which assists the development of an accountable system starting from the design time to the implementation.

Keywords : Accountability, privacy, DSL, service, cloud, security, software engineering.

1 Introduction

Les mécanismes de sécurité classiques (ou préventifs) ont montré leurs limites face aux situations non gérées ou non définies dans la politique de sécurité. Ce risque augmente de façon considérable dans un environnement connecté et hétérogène comme le cloud, composé d'entités avec différentes technologies et leurs failles techniques. L'*accountability* (qui se traduit par responsabilisation ou obligation de rendre compte) permet de combler les failles des mécanismes de sécurité préventifs en offrant la possibilité d'appliquer des corrections a posteriori.

Selon [12], l'*accountability* concerne le régime de gestion de données adopté par des organismes chargés de manipuler les données personnelles. Ces organismes sont responsables du traitement, du partage et du stockage de ces données, et cela sous des contraintes et des règles juridiques. Les obligations associées à ces responsabilités concernant la gestion de données peuvent être exprimées dans une politique de responsabilisation, qui est un ensemble de règles qui définit les conditions dans lesquelles une entité responsable doit fonctionner. Actuellement il n'y a ni standards pour exprimer ces politiques, ni méthode pour concevoir un système responsable. Dans [9], les auteurs considèrent que l'*accountability* peut être assurée en 5 étapes temporelles : (1) Prévention : l'application des mécanismes préventifs de sécurité. (2) Détection : la détection des violations des politiques de sécurité. (3) Évidence : la collecte de preuves en

se basant sur les logs du système. (4) Jugement : l'identification des victimes et des coupables. (5) Sanction : punition des coupables par le système.

La problématique de protection des données personnelles et de la vie privée prend une très grande ampleur à l'échelle nationale et internationale. De plus en plus d'utilisateurs prennent conscience de ce fléau. Plusieurs lois ont vu le jour comme la directive 95/46/EC en Europe, la réglementation HIPAA et GLBA aux États Unis, FIPPA au Canada, ainsi que plusieurs projets internationaux comme P3P et européens comme PPL, PRIAM et A4CLOUD ¹ dans lequel s'inscrit cette thèse. Dans ce travail je présente un framework permettant de prendre en compte la responsabilisation dans les logiciels dès la phase de conception.

2 Framework pour la responsabilisation

2.1 Présentation générale

Dans l'optique de prendre en compte la responsabilisation dans les logiciels dès la conception, nous avons proposé d'enrichir le diagramme de composants [3] en utilisant un langage abstrait pour la responsabilisation [4]. La figure 1 représente l'architecture globale du framework appelé AccLab (Accountability laboratory).

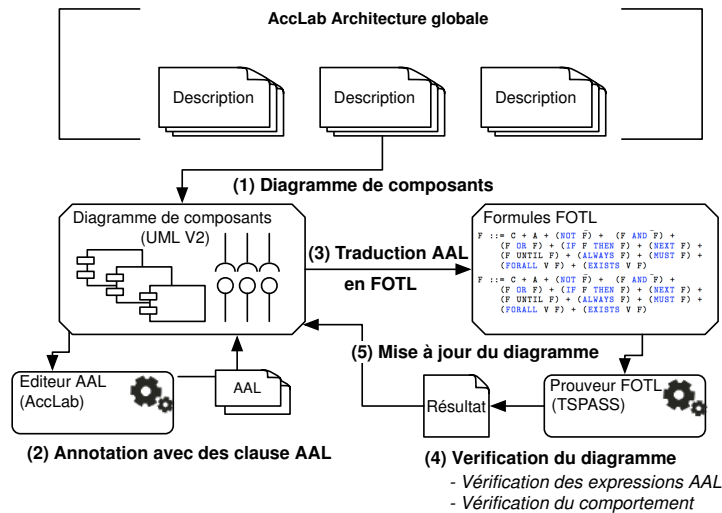


Figure 1: Vue globale du framework

1. On représente l'architecture du système à l'aide d'un diagramme de composants;
2. On annote le diagramme de composants avec les politiques de chaque acteur en utilisant le langage dédié ² nommé AAL;
3. Le diagramme de composants ainsi que les annotations de politique AAL sont traduits en logique temporelle du premier ordre;
4. On vérifie la cohérence du diagramme, la consistance des politiques ainsi que certaines propriétés à l'aide d'un prouveur logique;

¹Cloud Accountability Project. <http://www.a4cloud.eu/>

²DSL : Domain specific language

5. Suivant les résultats de la preuve, on corrige le diagramme et les politiques associées aux composants.

2.2 Traduction des obligations en AAL :

Afin de représenter les obligations qui sont en générale décrites en langage naturel, nous utilisons le langage AAL (Abstract Accountability Language) [4], qui est une sorte de pivot entre le langage naturel et le langage machine. La grammaire simplifiée de AAL est présenté dans le listing 1.

Listing 1: Grammaire du langage AAL

```

1  AALprogram    ::= (Declaration | Clause)*
2  Declaration  ::= literal : Type
3  Clause       ::= CLAUSE Id ( Usage [Audit Rectification])
4  Usage        ::= ActionExp
5  Audit        ::= AUDITING ActionExp
6  Rectification ::= IF_VIOLATED_THEN ActionExp
7  ActionExp    ::= Action | NOT ActionExp | Modality ActionExp | Condition | Author | Quant*
8                | ActionExp (AND|OR|ONLYWHEN|UNTIL) ActionExp | IF ActionExp THEN ActionExp
9  Exp          ::= Variable | Constant | Var.attribute
10 Condition    ::= [NOT] Exp | Exp [== | !=] Exp | Condition (AND|OR) Condition
11 Author       ::= (PERMIT | DENY) Action
12 Action       ::= agent.service['agent'](Exp) [Time] [Purpose]
13 Quant        ::= (FORALL | EXISTS) Var
14 Modality     ::= ALWAYS | NEVER | SOMETIME | NEXT
15 Time         ::= (AFTER | BEFORE) Date | Time (AND|OR) Time
16 Date, Type, Var, attribute, Id, agent, Purpose ::= literal

```

On considère une clause de responsabilisation comme étant un triplet (ue, ae, re) (ligne 3, Listing 1) avec la sémantique informelle suivante : *A chaque état d'exécution, assurer l'usage (ue) au mieux, si une violation de ce dernier est observée par un audit (ae) alors on applique la rectification (re).* L'usage (ue) contient un ensemble d'autorisations (ligne 11) et des actions (ligne 12). La forme générale d'une action est `agent1.service[agent2](exp)` qui signifie que `agent1` utilise le service fourni par `agent2` avec des données spécifiques `exp`. Les parties audit et rectification contiennent aussi un ensemble d'autorisations et d'actions mais ces derniers sont délimitées par les mot clés `AUDITING` pour l'audit (ligne 5) et `IF_VIOLATED_THEN` pour la rectification (ligne 6).

Ci-dessous un exemple d'un programme en AAL :

```

AGENT Kim TYPE(DataSubject) REQUIRED(write, read, notify) PROVIDED(sensor)
AGENT cloudX TYPE(DataProcessor) REQUIRED(sensor, send) PROVIDED(read, write)
AGENT auditor TYPE(Auditor) PROVIDED(audit)

CLAUSE kim_policy (
  FORALL d:data IF (d.subject==Kim) THEN (
    PERMIT Kim.write[cloudX](d) AND
    PERMIT Kim.read[cloudX](d) AND
    FORALL a:agent DENY cloudX.send[a](d))
  AUDITING ALWAYS IF Kim.notify[auditor]()
    THEN auditor.audit[cloudX]()
  IF_VIOLATED_THEN auditor.sanction[cloudX]()
)

```

Ce programme est composé de déclarations (les agents `kim`, `cloudX` et `auditor`) avec leurs services fournis et requis ainsi qu'une clause `kim_policy` représentant la politique de l'agent `kim`. Cette clause définit : (1) une expression d'usage autorisant à `kim` d'écrire et de lire ses données stockées chez `cloudX` et interdisant à `cloudX` d'envoyer les données de `kim` aux autres agents; (2) une expression d'audit qui déclenche un audit quand `kim` notifie `auditor`; (3) une expression de rectification sanctionnant `cloudX` en cas de violation.

2.3 Traduction en FOTL et vérification

On utilise une logique temporelle de premier ordre (FOTL) [10] pour interpréter notre langage. FOTL étend la logique du premier ordre (permettant les quantificateurs et les prédicats) et la logique temporelle linéaire qui permet les modalités temporelles.

formula	$\psi ::= \text{true} \mid \text{false} \mid \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \varphi$	(formules propositionnelles)
	$\mid \exists x.\psi \mid \forall x.\psi$	(formules du premier ordre)
	$\mid \mathbf{X} \psi \mid \psi \mathbf{U} \psi \mid \psi \mathbf{R} \psi \mid \mathbf{G} \psi \mid \mathbf{F} \psi$	(formules temporelles future)

Table 1: FOTL : Syntaxe

$\mathbf{G} \psi$ pour **always** ψ signifie que la formule ψ doit être vraie à chaque instant; $\mathbf{F} \psi$ pour **sometime** ψ signifie que la formule ψ doit être vraie au moins une fois dans le futur; $\mathbf{X} \psi$ pour **next** ψ signifie que la formule ψ doit être vraie au prochain instant; $\psi_1 \mathbf{R} \psi_2$ pour **release** ψ signifie que la formule ψ_2 doit être vraie jusqu'à ce que ψ_1 le soit; $\psi_1 \mathbf{U} \psi_2$ pour **until** ψ signifie que la formule ψ_1 doit être vraie jusqu'à ce que ψ_2 le soit.

Principe de préférences Une des plus importantes propriétés est de vérifier si les politiques définies par l'utilisateur (ses préférences) sont bien respectées par les fournisseurs de services. Le principe est le suivant : Soient H et F deux clauses AAL représentant respectivement les préférences d'un utilisateur et la politique d'un fournisseur de service. H est assurée par F se traduit en logique par $H \Rightarrow F$. On doit donc prouver la validité de cette formule. Une formule ψ est dite valide si ψ est satisfiable et $\neg\psi$ est non satisfiable.

1. La formule $H \wedge F$ doit être satisfiable (H et F sont consistantes)
2. La formule $H \Rightarrow F$ doit être satisfiable
3. La formule $H \Rightarrow F$ doit être non satisfiable

2.4 Outils

Afin de vérifier les propriétés nous avons besoin d'un prouveur logique avec une syntaxe concrète et un système de décisions. Nous utilisons TSPASS³ qui est un prouveur pour la logique temporelle du premier ordre. Nous avons développé un outil portant le nom du framework AccLab⁴ qui couvre les différentes étapes du framework. L'outil fournit : (1) un éditeur de diagramme de composants (qui traduit les composants du diagramme en déclarations AAL); (2) un éditeur de politique AAL; (3) détection des erreurs syntaxiques et sémantiques; (4) une traduction automatique d'un programme AAL en FOTL; (5) vérification du principe de préférences.

3 Conclusion

Il est nécessaire de considérer *l'accountability* dès la phase de conception des systèmes [5], mais peu de travaux prennent cela en compte. Plusieurs langages ont été proposés pour représenter les obligations de protection de vie privée, comme PPL (*Prime Policy Language*) [1] qui est basé sur XML, le langage déclaratif SecPal4P [2] qui permet de spécifier à la fois les politiques et les

³<http://lat.inf.tu-dresden.de/~michel/software/tspass/>

⁴<http://www.emn.fr/z-info/acclab/>

préférences des utilisateurs exprimées par des assertions et des requêtes, SIMPL (*SIMple Privacy Language*) [11] qui est très proche du langage naturel. Cependant ces derniers ne couvrent que l'aspect préventif de *l'accountability* (i.e le contrôle d'accès et d'usage). Il existe aussi plusieurs modèles théoriques [6, 7, 8] permettant d'exprimer les obligations de façon formelle et vérifiable, cependant ils sont difficilement voire non utilisables par des non spécialistes. De plus, il n'existe pas d'outils ou de méthodes à notre connaissance accompagnant ces modèles afin d'assister la conception et l'analyse d'un système dit responsable. Dans ce contexte-là on propose un framework permettant de prendre en compte la responsabilisation dans un système informatique dès la phase de conception. Nos travaux futurs porteront sur la mise en application des politiques AAL et la traduction vers un langage de politiques compréhensible par la machine.

References

- [1] 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, 2009.
- [2] Moritz Y Becker, Alexander Malkis, and Laurent Bussard. S4p: A generic language for specifying privacy preferences and policies. *Microsoft Research*, 2010.
- [3] Walid Benghabrit, Hervé Grall, Jean-Claude Royer, and Mohamed Sellami. Accountability for abstract component design. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, Verona, Italy, August 27-29, 2014*, pages 213–220, 2014.
- [4] Walid Benghabrit, Hervé Grall, Jean-Claude Royer, Mohamed Sellami, Karin Bernsmed, and Anderson Santana de Oliveira. Abstract accountability language. In *Trust Management VIII - 8th IFIP WG 11.11 International Conference, IFIPTM 2014, Singapore, July 7-10, 2014. Proceedings*, pages 229–236, 2014.
- [5] Denis Butin, Marcos Chicote, and Daniel Le Métayer. Log design for accountability. In *IEEE Symposium on Security and Privacy Workshops*, pages 1–7. IEEE Computer Society, 2013.
- [6] 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.
- [7] Sandro Etalle and William H. Winsborough. A posteriori compliance control. In Volkmar Lotz and Bhavani M. Thuraisingham, editors, *SACMAT 2007, 12th ACM Symposium on Access Control Models and Technologies, Sophia Antipolis, France, June 20-22, 2007, Proceedings*, pages 11–20. ACM, 2007.
- [8] Joan Feigenbaum, Aaron D. Jaggard, and Rebecca N. Wright. Towards a formal model of accountability. In Sean Peisert, Richard Ford, Carrie Gates, and Cormac Herley, editors, *NSPW*, pages 45–56. ACM, 2011.
- [9] 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.
- [10] Michael Fisher. A normal form for temporal logics and its applications in theorem-proving and execution. *Journal of Logic and Computation*, 7(4):429–456, August 1997.
- [11] Daniel Le Métayer. A formal privacy management framework. *Formal Aspects in Security and Trust*, pages 1–15, 2009.
- [12] Siani Pearson, Vasilis Tountopoulos, Daniele Catteddu, Mario Südholt, Refik Molva, Christoph Reich, Simone Fischer-Hübner, Christopher Millard, Volkmar Lotz, Martin Gilje Jaatun, Ronald Leenes, Chunming Rong, and Javier Lopez. Accountability for cloud and other future internet services. In *CloudCom*, pages 629–632, 2012.