On the Specification and Enforcement of Privacy-Preserving Contractual Agreements*

Gerardo Schneider

Department of Computer Science and Engineering,
Chalmers | University of Gothenburg, Sweden
gerardo@cse.gu.se

Abstract. We are here concerned with the enforcement at runtime of contractual agreements (e.g., Terms of Service) that respect users' privacy policies. We do not provide a technical solution to the problem but rather give an overview of a framework for such an enforcement, and briefly discuss related work and ideas on how to address part of the framework.

1 Introduction

Each time we download an app in our smart phone or access certain webpages we need to deal with *contractual agreements* (e.g. *Terms of Service* —ToS) and *privacy policies*. These are usually written in *legalese*, a somehow obscure language, motivating people to click the *I agree* button without reading the text, eventually having unexpected consequences. The situation is even more complex given that in many cases the applications may interact with each other in complex ways, most of the time without our knowledge nor consent.

As an example, let us consider the following scenario. Facebook and Spotify have their own ToS which users must agree on before getting the right to install and use their services. It could happen that when listening to a song in Spotify, a message in your wall is posted showing what the user is listening to. This might no be wanted by most users, compromising their privacy. It could be desirable to have the possibility to statically check whether the ToS conforms with the user's privacy policies. In case a potential breach of privacy is found statically, if the user still want to install and use the application, it could be desirable to have a monitor that warns the user when Facebook is going to post about what you are listening to (even better, the monitor could prevent Spotify and Facebook to make the post public if this is not allowed by the user's privacy policy).

Two crucial observations: i) Privacy is an important concern as personal information may be collected without our (explicit, or informed) consent from online social networks, search engines, mobile devices, etc., and used and shared

* Published in *7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation - ISoLA'16 (2); Track: Runtime Verification and Enforcement, the (industrial) application perspective*, volume 9953 of LNCS, pages 413-419. Springer, Oct 2016. DOI: 10.1007/978-3-319-47169-3_34

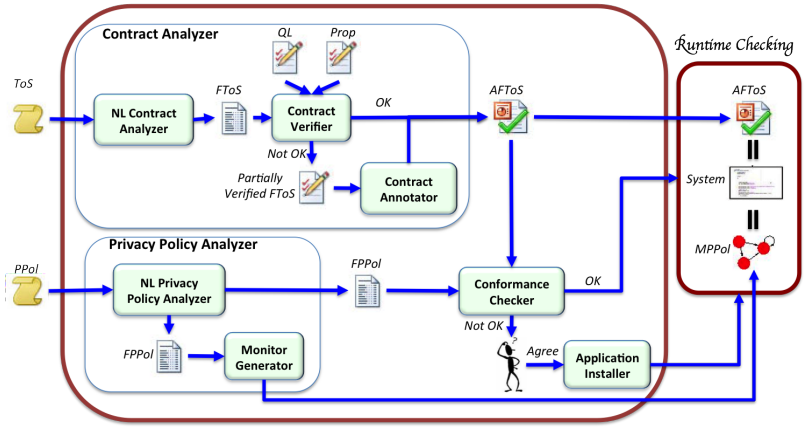

Fig. 1. Conceptual view of our framework

by private and governmental agencies for different purposes (e.g., see the series of articles in the Wall Street Journal[1]); ii) Normal citizens do not have the knowledge (nor the time) to go through the details of contractual agreements (e.g., ToS), accepting them with the naive expectation that they are to be trusted.

Our ultimate goal is to make the hard task of analyzing contractual agreements easier for normal users. Besides, we want to empower users with the possibility to be in control of *what* may be seen by *whom*, and *when*. For that we envision a framework allowing users to: i) Define their own privacy policies, at different levels and for general or specific applications, using a user-friendly environment; ii) Perform some simple queries on contractual agreements to ensure they are satisfied with the terms, before accepting them; iii) Check the contractual agreements do not violate the user's privacy policies (offline, static checking); iv) Get a warning in case the application is going to perform an action not allowed by the contractual agreement, or that might violate their privacy policies (online, at runtime).

In order to achieve the above, there is a need for a multi-disciplinary approach combining expertise from natural language processing (NLP), machine learning (ML), formal methods (formal semantics, specification and verification), logics, security, and legal analysis and formalization of law. More concretely, we aim at a framework (*cf.* Fig. 1) where policies are written in natural language and translated into a formal language (*cf. NL Privacy Policy Analyzer*), and a monitor is automatically generated to ensure that the policies are not violated at runtime (*cf. Monitor Generator*). Contractual agreements written in natural language, e.g. ToS, are also transformed into a formal language (*cf. NL Contract Analyzer*) so they are verified against properties (*cf. Contract Verifier*) and statically checked for conformance with the privacy policies (*cf. Conformance Checker*).

[1] "What they know", *Wall Street Journal*. Accessed on March 10, 2014 (`http://online.wsj.com/public/page/what-they-know-digital-privacy.html`).

These formal languages will be used during the execution of the application to ensure that the privacy policies are respected (*cf. Runtime Checking*).

We give an overview of the framework in §2, and discuss some of its components in more detail in §3 putting them into context w.r.t. the state-of-the-art.

2 Description of the PPCA Framework

We describe below a prototypical scenario of our framework (*cf.* Fig. 1).

1. The user will define her privacy policies interacting with a user-friendly interface. The policies will be translated by the module *NL Privacy Policy Analyzer* into the formal language *FPPol*.
2. The *Monitor Generator* will generate a monitor for the privacy policies.
3. When an application is uploaded, the user will be asked to approve the ToS (*ToS*). Before agreeing to upload the application, the ToS will be translated into a formal contract language (*FToS*) by the *NL Contract Analyzer*.
4. *FToS* will be analyzed by using the *Contract Verifier* module, allowing the verification of the contract against user-defined properties (*Prop*), or queries defined in the query language *QL*. The results of the the analysis (and queries) will be shown to the user, who will decide whether to accept it or not. If not accepted, *FToS* will be annotated with additional information by the *Contract Annotator* module.
5. The *Conformance Checker* module will statically check whether the contract *might* violate the policies specified by the user. If so, the user will get to know what clauses are to be analyzed in detail. In case of uncertainty the ambiguous sentences will be highlighted to be further analyzed interactively. The user will decide whether to agree to upload the application or not.
6. The user may decide to modify her privacy policies, by relaxing or constraining the policies for this particular application. This is done by the module *Privacy Policy Updater* (*not shown in the picture*). This is fed into the corresponding modules to obtain a runtime monitor for the new privacy policies.
7. If the user agrees to upload the application (or automatically done by the *Conformance Checker*), the monitor *MPPol* will run in parallel with the underlying system, checking that the privacy polices are respected at runtime.
8. The *Runtime Violation Analyzer* (*not shown in the picture*) detects when the contract is violated and uses the history of the transaction to analyze it at runtime (and eventually acting accordingly by canceling the transaction, or giving a warning). In case of non-conformance with respect to the privacy policies, the user is notified and the application temporary blocked.
9. The result provided by the *Runtime Violation Analyzer* (a *Log* file) will be passed to the *Static Violation Analyzer* so the end-user (or a specialist in case of litigation) can analyze it off-line to further determine what where the causes and who was responsible for the contract violation (*not depicted*).

3 On the Specification and Enforcement of Contractual Agreements and Privacy Policies

We only focus here on three main parts of the framework: i) The formal specification of contractual agreements; ii) The formal specification of privacy policies; iii) The static and runtime conformance checking.

Formalization and analysis of contractual agreements Since its modern conception in the 1950's *deontic logic* [19] has been the base of almost all the attempts to formalize normative systems. Most of the work have been dedicated to the study of properties of normative operators, e.g., obligations, permissions and prohibitions, and how to handle their violations (e.g. [13]), but few attempts have been made in order to obtain a usable formal language for the specification and analysis of contractual agreements. One of such languages is $\mathcal{CL}$ [16, 17], based on deontic, temporal and dynamic logic. $\mathcal{CL}$ has been shown to be insufficient to handle rich contractual documents; besides it does not have many of the desirable properties of contract specification languages [12]: does not allow to determine liabilities nor causalities, it is not compositional, and it cannot express real-time constraints. Some of these issues have been partially solved by *C-O Diagrams* [10] but the language is still not expressive enough as to capture real contractual agreements. Another language, FLAVOR [18], allows to distinguish between different "instances" of a contract, but reasoning about permissions is not possible. Besides, there are no analysis algorithms/tools associated with the language. In what concerns the formal analysis of contractual agreements, there is not much work. For instance for $\mathcal{CL}$ a complex translation into an existing model checker has been provided [11] by abstracting away some of the features of the language losing expressivity in what can be proven. The translation of C-O Diagrams into timed automata is promising as it opens the possibility to use verification tools like UPPAAL [4].

In order to ensure the fulfillment of the contractual clauses, the contract needs to be monitored at runtime. An interesting work along these lines is based on the event calculus [8]. However, this and other up-to-date approaches can only partially monitor a contract fulfillment, and more powerful monitoring techniques are needed. A complete automatic generation of a monitor from a formal contract is in general impossible, because most contracts contain prescriptions and descriptions at a high level, not specifying the underlying (algorithmic) procedures. This is the case for instance if the contract talks about average or percentages within a given period of time. We are not aware of any work providing (semi-)automatic monitor extraction techniques for such complex contracts.

Formalization and analysis of privacy policies As for contractual agreements, privacy policies would need to be expressed in a formal language, for instance based on real-time, epistemic and deontic logics. Epistemic logic is needed to specify and reason about *who* knows *what*, deontic operators are needed to describe for instance who is (not) *allowed* to perform certain actions, and real-time

is obviously needed as policies may have deadlines or durations, and to describe and reason about the evolution of the system and the policies themselves.

One approach to formally define privacy policies is to use some variant of *epistemic logic* [7], where it is possible to express the knowledge of *multi-agent systems*. Other approach for privacy, not based on epistemic logic, is *Relationship-based access control* (REBAC) [9], where the reasoning is focused on the resources owned by the agents. This approach is highly suitable for a practical implementation of a policy checking algorithm. On the other hand, it is mostly suitable for controlling access to resources, not for detecting certain kinds of implicit knowledge flow. Datta *et al.* present in [6] the logic PrivacyLFP for defining privacy policies based on a restricted version of first-order logic (the restriction concerns quantification over infinite values is avoided by considering only relevant instances of variables). The logic is quite expressive though it is not clear how it could be used for the kind of polices we are aiming here.

Though not directly concerned with privacy policies, the work by Basin and colleagues on different aspects of *usage control* is quite relevant. These include works on the definition of formal models for mechanisms to enforce usage control policies on the consumer side [15], and the use of temporal logics to express usage policies and runtime monitoring to check system compliance e.g. [3, 2].

A starting point for defining a rich formal language for privacy policies could be the recent work on the definition of the $\mathcal{PPF}$ framework [14].

Static and runtime conformance checking Our aim is to develop static verification techniques and algorithms for determining whether a given contract might compromise the parties' privacy policies. The static verification would be done with respect to a compliance relation between the formal term representing the contract and the formal privacy policy. The techniques behind this kind of proofs are quite standard, the challenge being in specializing them into our particular formal languages. One challenge is to obtain algorithms to extract a monitor from the formal policy language in order to check that the contracts do not violate the parties' privacy policies at runtime. An idea would be to apply techniques based on the *sub-formula construction*. However, this approach cannot capture clauses having *algorithmic content*. A possible solution to that would be to enhance the monitor with a library computing such clauses, and combine it with a *rule-based* approach to consider in more detail the specific events that might violate the policy. Besides, there might be a need to provide a full operational semantics of the system under test in order to consider what are the side effects of every event. We are not aware of any work checking conformance between contractual agreements and privacy policies.

An interesting promising approach would be to combine static and runtime verification as done in the StaRVOOrS framework [1, 5].

4 Conclusion

We believe the benefits of the PPCA framework are many, as well as the challenges to achieve it. A specific challenge for the runtime verification community

is how to automatically obtain and deploy runtime monitors from formal contractual agreements and privacy policies, in order to enforce their satisfaction as well as the compliance of the former w.r.t. the latter.

Acknowledgements Partially supported by: the Swedish Research Council (*Vetenskapsrådet*) under grants Nr. 2015-04154 (*PolUser: Rich User-Controlled Privacy Policies*) and Nr. 2012-5746 (*Remu: Reliable Multilingual Digital Communication*), and the European ICT COST Action IC1402 (*ARVI: Runtime Verification beyond Monitoring*).

References

1. W. Ahrendt, M. Chimento, G. Pace, and G. Schneider. A specification language for static and runtime verification of data and control properties. In *FM'15*, volume 9109 of *LNCS*, pages 108–125. Springer, 2015.
2. D. Basin, M. Harvan, F. Klaedtke, and E. Zalinescu. Monitoring data usage in distributed systems. *IEEE Trans. on Soft. Eng.*, 39(10):1403–1426, 2013.
3. D. A. Basin, M. Harvan, F. Klaedtke, and E. Zalinescu. Monpoly: Monitoring usage-control policies. In *RV*, volume 7186 of *LNCS*, pages 360–364, 2011.
4. J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal – a Tool Suite for Automatic Verification of Real-Time Systems. In *Hybrid Systems III*, number 1066 in LNCS, pages 232–243, 1995.
5. J. M. Chimento, W. Ahrendt, G. Pace, and G. Schneider. STARVOORS: A Tool for Combined Static and Runtime Verification of Java. In *RV'15*, volume 9333 of *LNCS*, pages 297–305. Springer, 2015.
6. A. Datta, J. Blocki, N. Christin, H. DeYoung, D. Garg, L. Jia, D. K. Kaynar, and A. Sinha. Understanding and protecting privacy: Formal semantics and principled audit mechanisms. In *ICISS*, volume 7093 of *LNCS*, pages 1–27. Springer, 2011.
7. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about knowledge*, volume 4. MIT press Cambridge, 1995.
8. A. Farrell, M. Sergot, M. Sallé, and C. Bartolini. Using the event calculus for tracking the normative state of contracts. *Int. J. Cooperative Inf. Syst.*, 14(2-3):99–129, 2005.
9. P. W. Fong. Relationship-based access control: Protection model and policy language. In *CODASPY'11*, pages 191–202. ACM, 2011.
10. E. Martínez, E. Cambronero, G. Diaz, and G. Schneider. A Model for Visual Specification of e-Contracts. In *IEEE SCC'10*, pages 1–8. IEEE Comp. Soc., 2010.
11. G. Pace, C. Prisacariu, and G. Schneider. Model checking contracts –a case study. In *ATVA'07*, volume 4762 of *LNCS*, pages 82–97. Springer, 2007.
12. G. J. Pace and G. Schneider. Challenges in the specification of full contracts. In *iFM'09*, volume 5423 of *LNCS*, pages 292–306, 2009.
13. M. Palmirani, G. Governatori, A. Rotolo, S. Tabet, H. Boley, and A. Paschke. Legalruleml: Xml-based rules and norms. In *RuleML'11*, volume 7018 of *LNCS*, pages 298–312. Springer, 2011.
14. R. Pardo and G. Schneider. A formal privacy policy framework for social networks. In *SEFM'14*, volume 8702 of *LNCS*, pages 378–392. Springer, 2014.
15. A. Pretschner, M. Hilty, D. A. Basin, C. Schaefer, and T. Walter. Mechanisms for usage control. In *ASIACCS*, pages 240–244. ACM, 2008.

16. C. Prisacariu and G. Schneider. A formal language for electronic contracts. In *FMOODS'07*, volume 4468 of *LNCS*, pages 174–189. Springer, 2007.
17. C. Prisacariu and G. Schneider. CL: An Action-based Logic for Reasoning about Contracts. In *WOLLIC'09*, volume 5514 of *LNCS*, pages 335–349. Springer, 2009.
18. R. Thion and D. L. Métayer. Flavor: A formal language for a posteriori verification of legal rules. In *POLICY'11*, pages 1–8. IEEE Comp. Soc., 2011.
19. G. H. V. Wright. Deontic logic. *Mind*, 60:1–15, 1951.