Ensuring Privacy Conformance in Inter-Domain Systems

Carl A. Gunter
University of Illinois Urbana-Champaign
September 2006
W3C Workshop on Languages for Privacy Policy Negotiation and Semantics-Driven Enforcement

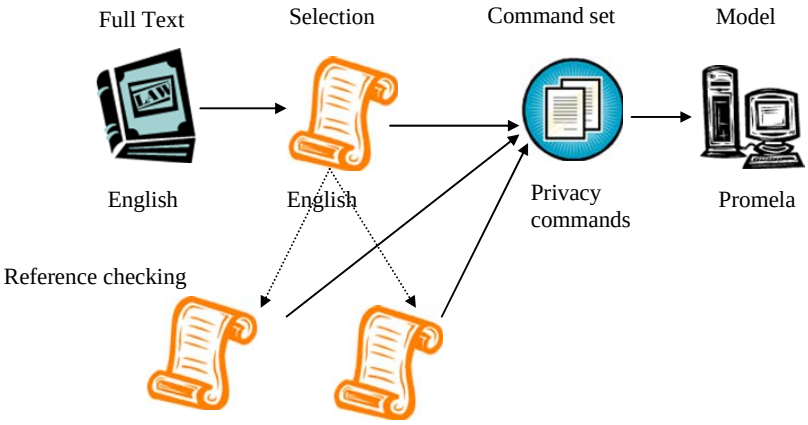
Introduction

Conformance to legal rules for handling private data is one of the most important new challenges to enterprise IT. Citizens of many countries are concerned by the scope of sensitive information that is being collected about them and the scope of losses that can occur when this privacy is compromised. They have often reacted by asking their governments to establish rules for handling their data. This has a profound impact on the information systems of companies, which are now faced with a fresh collection of requirements imposed on their information processing infrastructure and workflow. Even in the absence of such rules, there is a need for taking care to prevent the creation of burdensome rules, to promote good business practice, and to do the right thing. This comes in an environment in which strong business opportunities lie in working with private data. Examples include context aware systems, which track the locations of people using their vehicles or cell phones, and assisted living, which can exploit data networking to report medical information about people from their homes. To assure conformance and maintain the trust of subjects whose private data is at stake, enterprises need to develop techniques that are more formal, automated, and rigorous.

The aim of this white paper is to speculate on the challenges posed in taking techniques for encoding and analyzing privacy rules and combining these with technologies for inter-domain policy management. To this end I will sketch some progress on formal encoding of legal rules using "auditable privacy APIs", which were introduced in [MayGL06] and consider their use in the context of inter-domain policy adaptation based on the semantic web concepts discussed in [AfandiZG06]. I will then cull out a few of the challenges in bringing these technologies together and illustrate this with an assisted living application.

Formal Privacy

Legal privacy rules are important to satisfy, but not easily translated into formal statements that express the requirements for computer software. However, it is important to accept the ambiguity of the legal rules without being frozen by it. When showing that a workflow or software system conforms to legal rules I would rather be in the position of saying "we interpreted these rules as follows and here's how we know we comply to our interpretation" rather than saying "the rules might be ambiguous so we ignored them". In [MayGL06], we developed a framework for the analysis of legal rules that enabled them to be reduced to a family of interfaces that describe what can be done in managing information in accordance with the rules. The target rules are precise enough to admit formal analysis using a model checker. The following figure illustrates the main ideas.



The analysis starts with a natural language version of the complete set of rules; in our study these were written in English (or American to be a little more precise). These are reduced to the target rules for encoding, which, in our study, were the consent rules in two revisions of the U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA). The consent rules themselves reference a number of other parts of the full HIPAA text, so reference checking incorporates all of the necessary natural language materials in to a self-contained extract of the full text. These are then converted in to a collection of formal expressions called *privacy commands* in our system. Here is an example of one such, drawn from our encoding of the 2003 version:

Disclose506c1 (a, s, r, p, f, evidence)
if AllowedAsIn506c1 (a, s, r, p, f, evidence)
and own **in** (a, f)
then CopyObject (a, s, f, f')
and insert own **in** (r, f')
and EnterDisclose (a, p, f)
end

This encodes a portion of 164.506(c)(1), which says that "A covered entity may use or disclose protected health information for its own treatment, payment, or health care operations". Note the difference in style between this type of encoding and what one would obtain from EPAL or XACML, which do not focus on the explicit manipulation of state. Privacy commands in our system are basic access control matrix language expressions augmented by what is required to encode privacy rules, like evidence and obligations). A full discussion of the related work can be found in [MayGL06].

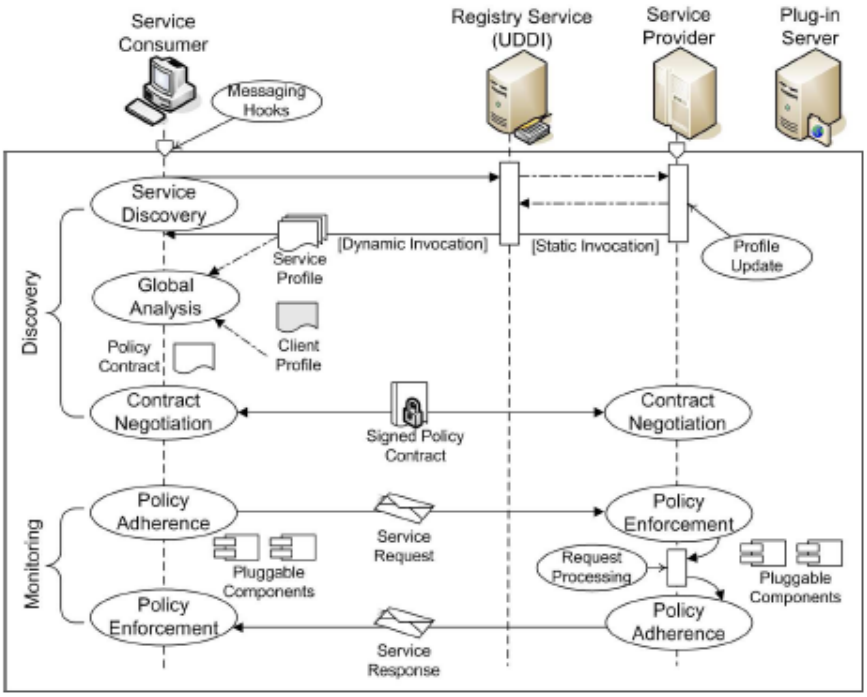
We encoded the 2000 and 2003 versions of the consent rules: there were 65 rules in the 2000 version and 59 rules in the 2003 version. The resulting sets of privacy commands gave us a pair of what we call *Auditable Privacy Interfaces (Privacy APIs)*. These were analyzed in the SPIN model checker to examine properties of the 2000 version that were viewed as problematic in discussions that led to the 2003 version of the rules. For example, one such "theorem" says: *Ambulance workers must obtain consent for services they did for unconscious patients after the fact*. This feature of the 2000 rules was disruptive to the workflow of ambulance workers, and it was agreed to change it in the 2003 version. Our analysis confirmed this property in the new version, but not some other similar ones. In the end, we did not find anything new to attorneys and privacy officers who specialize in HIPAA, but we did find things *we* were not aware of that that they recognized as important issues. This suggests benefits can be derived for other privacy rules sets, like U.S. GLB and COPPA, EU Privacy Directive 95/46/EC, and the Australian Privacy Act of 1998, which are areas of future application we would like to explore. The main objective of this work lies not so much in pointing out issues to law makers as it does in helping

enterprises to express, implement, and prove conformance of their systems to legal privacy rules. This can, in principal, be accomplished by implementing software that manipulates private data on top of a privacy API. Of course, one must close the gap between the privacy API formalism we encoded in SPIN and the implementation of a real system, which is probably written in a language like C++, C#, or Java, but encoding legal rules in a language that stands between the natural language and the system implementation is reasonable and feasible step.

Adaptive Middleware Policy

Let us now turn to the question of inter-domain policy conformance. It would make things easier in many respects if it were possible to adopt a global collection of policies that all enterprises were willing to accept and enforce. In practice, this is far from feasible, since different organizations have different goals and, even if their goals are all the same, universal recognition and encoding of the common policy would probably be impossible. Consequently, the next best option is to enable principals to express their policies and prepare a collection of mechanisms for principals to conform to the policies they encounter. I find it helpful to use Internet messaging, *viz.* email, as a case study for thinking about this problem. It is universally adopted on the wide scale, but there are many variations of policy, such as the sizes of messages a relay will accept, the types of file extensions that will be permitted, the rules that will be used to determine whether a message is SPAM, and so on. These policies are not typically well known. In some cases, this is deliberate (like most SPAM filtering rules), but in other cases it is just inconvenient. How many times have you sent a message that did not arrive because it did not conform to some policy of the recipient that you would have liked to know before hand?

This email case study helps to illustrate the key elements that would be required of an *adaptive middleware* for distributed system that is able to perform policy conformance steps more automatically in a scalable diverse universe. In [AfandiZG06] we explored a design for this type of middleware that supports, in particular, QoS features in a Service Oriented Architecture (SOA). QoS features are non-functional properties such as reliability, response-time, security constraints, and privacy. Our underlying SOA is based on semantic models and the use of languages such as OWL and SWRL. These are used to express the three main components that such systems need: description, discovery, and monitoring. An overview of the architecture, which is named Adaptive Middleware Policy to Support QoS (AMPol-Q) is given in the following figure:



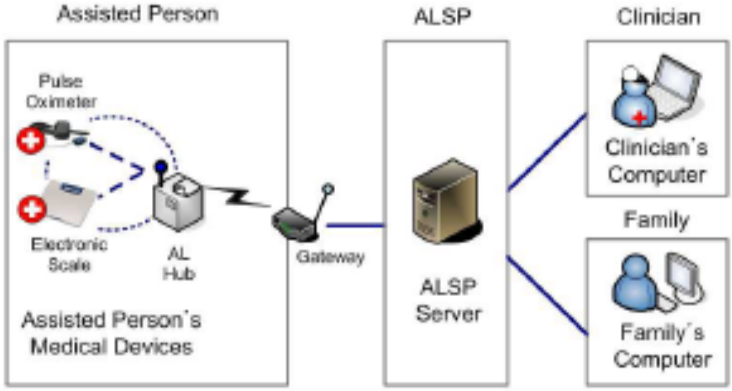
This can be understood "end-to-end" by moving through the steps that a service consumer follows to get a service that meets his QoS constraints. In the first step, the principals describe their policies statically. This description process exploits a semantic QoS ontology model (for basic concepts), a policy model (for rules the principals wish to enforce), and entity profile models (which say which QoS features require a policy). The next step is to initiate service discovery for the specific circumstances of the communication. This comprises finding the right service chain, carrying out global QoS analysis, and reaching policy agreement based on contract negotiation. The service chain is discovered based on the static policies considered in an Input, Output, Preconditions, Effects (IOPE) approach. The negotiation process forms the policies into a suitable dynamic policy for the global interaction the consumer seeks. Once this is settled, the final step provides extensions necessary to adhere to policies, which enables the service consumer to satisfy policy enforcement globally to achieve the desired request processing. One feature of AMPol-Q is agile and automated extension capabilities derived form pluggable components that the service consumer can obtain from a plug-in server.

Adapting to Privacy Rules

Privacy is a critical QoS feature that refers to the willingness of an enterprise to obey privacy rules in supplying its functional requirements. It notably includes features like encrypting network communications, but the story is much more complex that that. We can see this complexity from the HIPAA consent rules, which concern many circumstances in which record data can be released. In addition to such "declassification" rules, there may in general be rules about when data must be released (like information about infectious diseases), when data must be destroyed, and many others. An interesting challenge, therefore, is what it would take to add such privacy parts of an architecture like AMPol-Q and do so in a way that is amenable to a level of formal assurance. Our work on privacy APIs provides some hint about how this may proceed. In particular, this work shows a link between the QoS feature of privacy and the functional properties of the clients and servers in a distributed system by stipulating that all of the entities across all administrative domains must aggregately use no more than the permitted privacy API

commands. How the entities determine and enforce such a global guarantee is not clear from the current state of the art, but such assurances must be made in actual deployments.

As a brief illustration of this issue let us consider some work in [WangSLZ0ACGGHK06] and [MaySGL06] on developing an open system to support assisted living. The core idea is to provide an *Assisted Living Service Provider (ALSP)* who is capable of mediating the exchange of medical data between clinicians and *Assisted Persons (APs)*. Here is an illustration



In this architecture, a collection of devices from different vendors can communicate to a collection of diverse clinicians and third parties such as family members, insurers, etc. in a way that accepts the independence of these parties. The ALSP is an independent monitoring party that glues this all together. Overall we implemented this using web services, but not using AMPol-Q, which envisions a new strata of flexibility and dynamic operation. The open challenge is how to use such advanced features and provide the types of assurances that a system like privacy APIs could enable. [MaySGL06] discusses how we are able to show regulatory-relevant security features for the protocols using Microsoft's TulaFale system, but this falls far shy of showing full HIPAA compliance for the whole workflow that the system requires. Advanced types of inter-domain distributed systems that must comply to privacy rules in a global way will require more formal and automated techniques than we have yet achieved and studies to push the envelope with new SOA and semantic web concepts will be an interesting direction of exploration.

References

[AfandiZG06] Raja Afandi, Jianqing Zhang, and Carl A. Gunter, *AMPol-Q: Adaptive Middleware Policy to Support QoS*. To appear in: **International Conference on Service Oriented Computing (ICSOC '06)**, Chicago, IL, December, 2006.

[MayGL06] Michael J. May, Carl A. Gunter, and Insup Lee, *Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies*. In: **Computer Security Foundations Workshop (CSFW '06)**, Venice, Italy, July 2006.

[MaySGL06] Michael J. May, Wook Shin, Carl A. Gunter, and Insup Lee, *Securing the Drop-Box Architecture for Assisted Living*. In **Formal Methods in Software Engineering (FMSE '06)**,
Alexandria, VA, November 2006.

[WangSLZ0ACGGHK06] Qixin Wang, Wook Shin, Xue Liu, Zheng Zeng, Cham Oh, Bedoor K. AlShebli, Marco Caccamo, Carl A. Gunter, Elsa Gunter, Jennifer Hou, Karrie Karahalios, and Lui Sha, *I-Living: An Open System Architecture for Assisted Living*. In **International Conference on Systems, Man and Cybernetics (SMC '06)**, Taipei, Taiwan, October 2006.