Usage Control in Service-Oriented Architectures*

Alexander Pretschner[1], Fabio Massacci[2], and Manuel Hilty[1]

[1] Information Security, ETH Zürich, Switzerland
[2] Dept. of Information and Communication Technology, Università degli Studi di Trento, Italy

Abstract. Usage control governs the handling of sensitive data after it has been given away. The enforcement of usage control requirements is a challenge because the service requester in general has no control over the service provider's information processing devices. We analyze applicable trust models, conclude that observation-based enforcement is often more appropriate than enforcement by direct control over the service provider's actions, and present a logical architecture that blends both forms of enforcement with the business logics of service-oriented architectures.

1 Introduction

The past few years have seen major technological and business trends that are reshaping software technology and business processes. These trends have a profound impact on the trust models, security policies, security procedures, and security infrastructures that companies need to develop and maintain [1, 2]. From a technological perspective, service-oriented architectures (SOA) and business process management platforms have emerged as the architectures and technologies of choice for structuring and integrating applications within and across enterprises. From a business perspective, companies and institutions have increasingly outsourced the non-core parts of their business processes. Outsourcing is the ongoing administration, management, and possibly subcontracting of specific IT processes by external parties to enhance efficiency and effectiveness of those processes (cited after [2]). Outsourcing is sometimes iterated, so that the service provider (SP) itself outsources some functions to third parties [3]. In this way a company can concentrate on its core business rather than on peripheral tasks. Outsourcing often involves sensitive data such as trade secrets or the personal data of customers. Data owners (i.e., the companies who own the trade secrets) and data subjects (i.e., the customers to whom (personal) data is related) are interested in governing how that data may be used by the SPs. Further, regulatory frameworks such as the Sarbanes-Oxley act also impose technical and governance restrictions on how business-relevant data may or must be processed.

There is a long history of security research concerned with the protection of data. *Access control* (AC for short) addresses the question of who may access which data under which circumstances. Those decisions are taken with information that relates to

*This work was done while A. Pretschner was on leave at the universities of Trento and Innsbruck—support by the Bolzano Innsbruck Trento Joint School for Information Technology is gratefully acknowledged. F. Massacci was supported by the EU-funded S3MS project.

the present and the past. More recent work has extended the concept to *usage control* (UC for short; [4–7]) that is concerned with what may happen with data once a *data provider* has given it to a *data consumer*. In service-oriented settings, the data consumers are typically the SPs. Requirements about the future handling of data are called *obligations*. Examples include "delete the document within thirty days", "notify the data owner whenever the data is accessed", "log every access to the document", and "the data must be stored in an encrypted manner" [8]. Obligations are also studied in the areas of privacy (e.g., [9–11]) and digital rights management (DRM, e.g., [12, 13]).

In this paper, we tackle the problem of enforcing obligations in SOAs. Because of the special trust relationships in service-based business processes, it might be sufficient not to *ensure* the adherence to obligations—which is what is needed in DRM contexts—but rather to *observe and react* to violations in hindsight. Based on these considerations, we present a logical architecture that identifies the necessary core functionalities for enforcing UC. We show how to connect the UC logic to the business logic(s) of a SOA. We make the assumptions explicit, discuss the crucial aspect of trust, and demonstrate the limitations of the architecture. Implementing the architecture is a next step.

Problem Statement and Contribution. To summarize, the *problem* that we study is whether and how UC requirements can be enforced in SOAs and which enforcement strategies are appropriate in which scenarios. Our *solution* is the analysis of different trust models in usage control scenarios and the specification of functional components that can enforce UC requirements. The *contribution* of this paper is, to our knowledge, the first explicit conceptual treatment of security and trust requirements for UC in outsourced SOAs and the first logical architecture addressing it.

Overview. In §2, we discuss the fundamentally different UC-related trust models in the areas of DRM and service-based business processes. This analysis leads us to two different kinds of enforcement of UC requirements in §3, detective and preventive enforcement. We present the logical architecture in §4 and sketch two different deployment schemes in §5. Finally, in §6, we put our work in context and conclude.

2 Usage Control and Trust in Different Domains

Different stakeholders have different interests in UC. Human data subjects are interested in their privacy being respected. To keep their competitive advantage, companies want to prevent their trade secrets from falling into the hands of competitors. Similarly, artists and distributors of artworks and software are interested in receiving royalty payments for their intellectual property. Finally, shareholders and other parties are interested in compliance with governance rules such as the Sarbanes-Oxley act. In contrast to this perspective of the *data provider*, data consumers (or SPs, respectively) have different interests with regard to whether and how obligations are enforced.

In the DRM (B2C) area, the data consumer has in general no interest in adapting its computing infrastructure to meet the more or less prying needs of a data provider. The data consumers' well-being does not depend on whether or not they receive a movie. Furthermore, limitations imposed by DRM are often seen as a nuisance by the data consumers because DRM may also prevent playing data on several devices or making backup copies. Legal restrictions are often not taken seriously by the data consumers

because the enforcement of the law in this area is nearly impossible. In terms of UC enforcement mechanisms, the data consumers have an interest in data protection as many DRM mechanisms send information back to central servers that may be of privacy-sensitive nature.

This is in contrast to the relationship between a company and an outsourced SP (B2B). SPs are interested in adhering to the stipulated terms and conditions for two reasons. Firstly, maintaining a high reputation is important in competitive markets. Loosing one customer translates into considerable losses (compare the value of an outsourcing contract and the value of an mp3 song). Depending on the level of customization, a company using, e.g., the SAP R/3 business suite, might quickly move from one SP to another; and SPs with a low reputation have difficulties of finding new customers. Secondly, the legal implications of not adhering to the terms and conditions may be severe. The penalties stipulated in the outsourcing contracts act as a deterrence for the SP to handle the customer's data in unintended ways.

Finally, public administrations, seen as SPs, have in general no direct economic relationship with businesses (A2B) or citizens (A2C) who send them sensitive data. However, most processes in administrations are strictly governed and public administrations usually have no particular interest in breaking the respective laws and regulations. We may conclude that these administrations are inherently honest and that violations of usage control requirements tend to happen unintentionally rather than deliberately.

The relationship between SPs and service requesters often is subject to regulatory demands. These may include the requirement that a company provide evidence to regulators, auditors and finally its customers that it is delivering a secure, privacy-respecting, trustworthy service. Yet, regulators might not consider sophisticated outsourcing structures and may only hold one of the parties accountable to the end user. Similarly, even if contractual protection and deterrence can help avoid problems with regulators and law enforcement, they are not sufficient to mitigate the rage of customers whose pressure might force a global brand to take responsibility for outsourced services.

In sum, service requesters (data providers) must trust the SPs (data consumers) to handle the received sensitive data in accordance with stipulated terms. To achieve this trust, secure service-oriented infrastructures must be developed so that data providers can *specify* security policies for services and the infrastructure can *enforce* such policies, *monitor and detect* violations, and diagnose the root causes for violations in order to take appropriate actions. However, SPs may be reluctant to give the service requesters too much control over their IT infrastructures. Even giving away information about the internal behavior may be critical to the SP; but it is generally more acceptable than giving control to the service requester.

3 Enforcement of Usage Control

In DRM, data providers are interested in gaining enough control over the data consumers' IT infrastructures so that they can make sure that UC requirements are adhered to. This is a consequence of the trust relationship described in §2. In contrast, in the domain of business IT it may be impossible, not practical, too costly, or simply not necessary to fully control the IT infrastructure of the service that receives sensitive data.

This is equally a consequence of the respective trust relationship discussed in §2. SPs may, however, agree to present some (trustworthy—cf. §5) information about their actions. The original senders of the data can then, in hindsight, decide whether previously stipulated obligations have been adhered to. If not, they can penalize the receiver, e.g., by lowering trust ratings. The idea is that of deterrence: potential delinquents are aware that their wrongdoings may be detected and that they may be held accountable. In the following, we will refer to the first kind of enforcement as *preventive enforcement* and to the second kind as *detective enforcement*.

The functionality of mechanisms for *preventive enforcement* can be broken down into the fundamental strategies of inhibition, modification, execution, and finite delay [8, 14]. Mechanisms for preventive enforcement are mostly developed in the DRM area and usually perform enforcement by inhibition, with a few exceptions that support enforcement by modification [15]. *Detective enforcement* does not require direct influence on the actions performed by the SP but relies on signaling mechanisms that *inform* about actions of the SP. As a consequence, the original requirement (e.g., "delete the data item within thirty days") is transformed into a combined statement that consists of an observable requirement ("the execution of the deletion command within thirty days must be confirmed") and a compensating action that is executed in case of violation (e.g., "lower the SP's trust rating") [7]. We require that a violation of the observable requirement implies a violation of the original requirement. Ideally, one would like the opposite direction to hold as well, but this is in general not possible—this is the cost for using the weaker observation-based kind of enforcement. Detective enforcement involves both *signaling* and *monitoring* components. Typically, signalers reside at the SP's side (or at some distributed parts of the service requester infrastructure such as SAP R/3 clients). They send (partial) information about the provider's internal state or actions to the service requester. Monitors predominantly reside at the requester's side. They receive signals from the signalers and verify if these signals conform with applicable UC policies. Obviously, a monitor must trust the information that is sent by signaling components; the latter must be correct and complete: notifications are sent whenever necessary, and there must not be any "spurious" notifications (§5).

4 A Logical Architecture for Usage Control in SOAs

We now describe an architecture for UC in SOAs. It is logical in the sense that it is completely independent of any implementation. Later, in §5, we sketch two deployment schemes for integrating UC with an existing SOA.

Abstractly, a service is a functional entity with an internal state that receives and sends messages under well-defined conditions (contracts). A service S sends a request to another service S'; if there is a result of the computation that S may be interested in, S' can send a response message. Messages consist of a command that the requester wants the receiver to execute, possibly including references to the receiver's state (e.g., a data base), data that the receiver needs to perform its task and that might have to be usage-controlled, and UC policies for that data.

Fig. 1 schematically shows how to incorporate enforcement mechanisms for AC and UC into this simple service model. Labeled arrows represent the main data flows

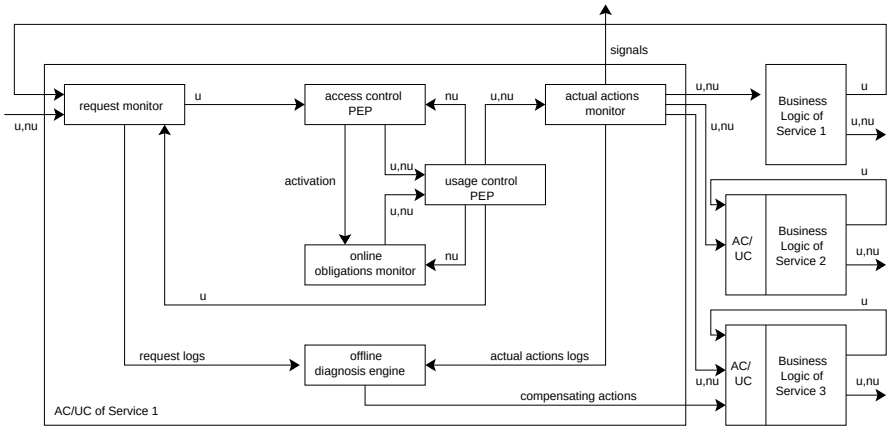

Fig. 1. Services with access and usage control

(messages) and boxes represent functional components; we will convey the meaning of the labels later. A request enters the system from the left hand side. Rather than passing it directly to the business logic of the receiving service (say, service 1, to which the request was directed), we first make the message enter the hierarchically decomposed box labeled "AC/UC". Once this box has performed its tasks, messages are sent to other services. These messages may be identical to those originally received, and the receiving services may be identical to the original receiver (service 1), but they need not, because the AC/UC enforcement mechanisms may have decided otherwise. The idea is that an AC/UC component resides within each communication channel between two services; it intercepts requests and performs its tasks.

We distinguish between messages that directly relate to usages (label *u*) and those that do not (label *nu*). Usage always relates to data for which UC policies exist. It can be classified into management, distribution to other parties, rendering, data processing, and execution of programs [15]. Usages can be combined; editing a document with a word processor, for instance, usually involves rendering, processing, and management usages. Messages that do not relate to usage include notifications, status reports, payments, fines, etc. In Fig. 1, we make the following assumption. All usage-related actions that the business logic of a service undertakes (manage, render, process, execute) as the result of processing a request (that is, usages that are not *directly* mentioned in the original request) are *not directly executed* but rather encoded as an explicit request and fed back to the AC/UC component of the service (uppermost arrow and feed-back data flows for services 2 and 3). In this way, we make sure that *all* usage-related actions (messages) always pass the AC/UC component of a service and are hence subject to UC. This is of course a strong assumption. However, it can be justified by the logical nature of the architecture: UC need not necessarily be implemented by dedicated software components (§5).

The main functional components of the AC/UC component are three monitoring components (request, actual actions, and obligations monitor), the AC policy enforcement point (PEP), a UC PEP that takes policy-defined actions should this be necessary (e.g., delete a file after thirty days, notify the data owner, spell out a fine), and a compo-

nent that can be used to analyze system executions w.r.t. a set of policies in an off-line manner and to analyze existing logs for business decisions.

The *request monitor* does nothing but log incoming requests. The main requirement is to be sufficiently fast and to scale. Some requests may not be allowed to be logged; this is stipulated by monitoring policies. The logs of all requests can later be used for offline analysis purposes. Incoming requests are forwarded to the AC PEP.

The *AC PEP* decides whether a request can be granted, that is, if the command in the (command, data, UC policy) triple of the message can be executed w.r.t. the applicable AC policies. These policies can reflect both the AC functionality of the business logic of the service itself and AC functionality that reflects requirements on the entire system rather than a single service (e.g., chinese wall policies). In addition to AC policies, UC policies may be applicable. These UC policies are defined system-wide (reflecting legal frameworks), on a per-service, or a per-data item basis. How these policies are retrieved and how the system knows when to activate them is outside the scope of the logical architecture. If any UC policies are applicable to the data that is part of the data in the request, they must be tied to the data object in question and then be activated. Activated obligations are handled by the *obligations monitor* discussed below. If the AC PEP decides that a request can be granted, it forwards it to the *UC PEP*. The reason for not directly sending the request to the service's business logic is that even at the moment of granting access, UC requirements may have to be enforced—e.g., a notification may have to be sent, or a policy that was activated earlier prohibits the current request from being forwarded to the service's business logic.

The online *obligations monitor* monitors requirements on the future of a data item that was previously sent to the service and for which an applicable UC policy exists. The monitor keeps track of the actions of the service. If specific policy-defined conditions are met (or violated), it makes the *UC PEP* perform specific tasks.

The *UC PEP* enforces UC requirements based on the four classes of enforcement defined in §3. For instance, the execution of action conditions with an executor mechanism can be done by spelling out fines—i.e., sending the respective message to a respective service—notifying a data owner, or automatically issuing a payment. Note that the execution of actions may also be delayed as demonstrated by the above example of deletion in thirty days (this is different from delaying enforcement mechanisms which simply wait to see if certain favorable conditions have, by virtue of actions the requester may have taken in the meantime, become true). As a second example, the obligations monitor can also tell the UC PEP *not* to forward a request to the service's business logic, i.e., inhibit it. For instance, a UC policy might state that some action must not be executed more than three times. When the fourth request arrives, then the obligations monitor will notify the UC PEP that this fourth request must not be forwarded to the service's business logic. The other forms of enforcement can be achieved in a similar manner. If the result of the UC PEP is different from the original request yet itself a usage (an action for which UC policies may be applicable), then this result must be enveloped into a request and fed back to the request monitor: similar to actions that the business logic of a service may want to execute, UC policies may be applicable.

If according to the obligations monitor, the UC PEP has nothing to do, then the latter forwards the input received from the AC PEP, and possibly also that of the obligations

monitor. The actions of the UC PEP may affect the AC PEP and the obligations monitor, e.g., if any policy stipulates actions to be taken after three fines were spelled out. These components hence get fed back with the actions executed by the UC PEP.

The *actual actions monitor* logs all messages that the UC PEP has deemed appropriate (or modified into something appropriate) for being executed by the service. Furthermore, it forwards the requests that it receives to the service's business logic (or to another service, if this is applicable—for instance, a penalty service). The actual actions monitor also serves as the signaling component for other services, as discussed in §3. The actual actions monitor does not need to send its data to the UC PEP and the obligations monitor because this has already been achieved by the UC PEP. The monitoring activity is governed by a monitoring policy, similar to the monitoring policy that governs the request monitor.

Finally, an *offline diagnosis engine* analyzes activities in the logs. Data mining techniques can be performed for risk analysis, or it might be decided that online detection of UC policy violations is not really an issue and that offline detection—e.g., once a week—is fully appropriate for a given business scenario.

In addition to trust issues that we will discuss in §5, this perspective on UC involves a constraint that relates to side effects. A UC policy may depend on the *output* of a service. For instance, a policy may state that if the *result* of the service's computation includes specific names, then it must not be distributed to specific parties. This means that the UC PEP must first trigger the service's business logic, then retrieve the result, and check it w.r.t. its UC policies. The problem then obviously is that the service's computation may have side effects that cannot be undone.

5 Engineering Usage Control in SOAs

We now describe the mapping of this functionality to a technical architecture and to the implementation.

Dedicated Services. One approach to implementing the above functionality implements dedicated software components for each functionality of one of the logical entities. AC and UC are hence enforced at the interface level of a service. The challenge with this approach is twofold. Firstly, preventive enforcement requires that *all* actions of a service are initiated by requests that pass through the respective AC/UC component. In other words, there must be no other way for a service to receive requests, and the service exclusively performs actions that are initiated by an external request. Proactive behavior of a service—that is not initiated by sending explicit messages to itself—is obviously prohibited by this approach. Secondly, detective enforcement requires all consumer-side events that are relevant for checking compliance with a policy to be (1) generated by the signaling components and (2) received and appropriately interpreted by the monitoring components. In particular, signaling components must not miss any events that they should inform the monitoring components about (completeness) and the signaling components must not notify the monitors of actions that have not taken place (correctness). As an example for a possible technical infrastructure, Apache Axis (ws.apache.org/axis), a development framework for Java web services, allows the definition of request handlers and handler chains. These handlers intercept

requests, perform specific actions, and then send requests to the business logic (and all requests necessarily pass the handlers). The AC and UC PEPs can be implemented as part of these handlers. Because the online obligations monitor must take into account information that is not exclusively related to requests that are sent to the business logic (e.g., time or notification messages), it cannot in general be integrated into the handlers. Checking the conditions of a UC requirement must be done by a dedicated component.

Weaving. The approach of implementing the enforcement infrastructure along the lines of the functional entities is appealing because of its modular nature: AC/UC components can be added to any communication channel in a system. However, we have seen that it relies on a set of rather strong assumptions on the deployment of the services. A further possibility consists of compiling the AC/UC functionality directly into the service. This scenario is attractive when a SP is tailoring services to customer's needs anyway, as it happens in several outsourcing scenarios. We would argue that when the service binary is built, one could also alter the source code so as to incorporate some functionalities of the logical architecture. In this scenario, generic AC PEPs, online obligation monitors, and UC PEPs could be interwoven with the service's business logic at compile time, similarly to what aspect weavers do in aspect-oriented programming [16], how monitors can be interwoven with object code [17], and to what modified Java virtual machine class loaders do at runtime to implement security requirements [18].

Parsimonious Trusted Computing. Both approaches to implementing UC enforcement mechanisms rely on assumptions that relate to trust at different levels. How can it be ensured that the only way for services to receive requests is after they have passed the respective AC/UC component? The system must be trusted that there are no other ways to request actions from a service. Similarly, how can the (currently deployed) business logic of a service itself be trusted? How can signalers provide monitors with the correct and complete information that these monitors need for assessing adherence to policies? How can it be ensured that a service reacts to a message in the specified way, i.e., how can input messages and internal actions be linked? And how can it be ensured that the components of AC and UC mechanisms have been implemented correctly and may not be tampered with? These difficult questions are relevant in both deployment scenarios.

At least some technological help can be expected here. First, with trusted computing technology, hardware-based solutions for restricting the actions of an IT system and making sure that a certain configuration of a service is running on a specific host (remote attestation) are becoming increasingly powerful. For example, trusted computing technology could be used to make signaling mechanisms more tamper-resistant. Approaches in this direction have been presented in the literature [19, 20]. Second, data caging at the level of hardware (e.g., Intel's LaGrande Technology) and operating systems (e.g., Symbian OS v9) can also be implemented for general business information systems and protect both program code and cryptographic keys. These cryptographic keys can be used to restrict access to data to those who possess the respective key (which of course implies that decrypted data must not be publicly accessible). Third, there exist approaches for securing software that cannot rely on trusted hardware [21, 22]. However, these approaches are often considered weaker than hardware-based approaches. Finally, off-the shelf components such as databases can be equipped with built-in mechanisms for increasing trust (e.g., Hippocratic databases [9]).

6 Related Work and Conclusions

UC has been discussed by several authors [5–7], with few researchers explicitly catering to the notion of distribution in UC, i.e., the loss of control over a data item after giving it away. Several policy languages for UC have been proposed [23, 24, 13, 12, 8]. Enforcement by observation and penalties has been documented [25, 5, 26]; and preventive control mechanisms have been surveyed and characterized [15, 8, 14]. All this work does not relate to the specifics of loosely-coupled software architectures for business information systems (in the P2P context, related work was mentioned earlier [20, 19]). To our knowledge, this paper constitutes the first treatment of UC in SOAs.

Distributed UC is concerned with requirements on data after this data has left the data provider's scope of influence. Sources for the respective requirements are the data owners' interests but also governance rules and regulations. Some of these requirements can be controlled. For other requirements, enforcement by observation and compensation is a suitable solution. Whether or not preventive or detective control mechanisms are applied depends on the underlying business and trust models. Enforcement by observation and compensation seems to be applicable in outsourced business service scenarios rather than in DRM (§2). In SOAs, the data consumer (SP) may not want the data provider (service requester) to be so powerful; full control may also be technically impossible, inappropriate because of the wrong trust model, or too costly. In the DRM scenario, the trust model is fundamentally different. Furthermore, in the area of DRM for handheld devices, we would argue that if there is sufficient control over the handheld's operations for *observation purposes*, then there should be sufficient control to directly enforce by *control* as well.

The contributions of this paper are technical and conceptual. *Technically*, we have identified the main functional components for enforcing UC policies. We have defined a logical architecture and presented two different deployment schemes, one relying on dedicated SW components, and one relying on weaving the functionality with the service's business logic at compile-time. *Conceptually*, we have shown how the appropriateness of different enforcement schemes (preventive or detective control) depends on the business model of the SP and the applicable trust model. In other words, we have shown that UC in SOAs and UC in DRM are fundamentally different. While trust is a huge technological and also organizational problem, we have hinted at first building blocks for respective solutions. We are aware that a logical architecture is only a first step and that we will face many challenges when implementing it.

In addition to scalability issues that we did not scrutinize in this paper, we did not mention two further technical challenges. Firstly, when instantiating the abstract notions of usages with actual usages, then the question about their semantics arises: if a policy specifies that a data item has to be "deleted", does this mean that all copies have to be deleted, that the data has to be physically over-written several times, or that in case of encrypted storage the key is deleted? At the level of business processes, however, policies may directly relate to (standardized) messages exchanged between services, which means that this problem may be less relevant. Secondly, we did not touch the problem of rights delegation and propagation in iterative outsourcing scenarios.

Open research and engineering problems relate to all of the above. We need to better understand how hardware-based trusted computing technology, secure storage of keys,

application-specific enforcement schemes, modern operating systems and middleware can help establish the necessary trust.

Acknowledgments. We would like to thank F. Casati and B. Crispo for contributing to and discussing the architecture as well as V. Lotz for many useful comments on the business side of SOAs.

References

1. Karjoth, G., Pfitzmann, B., Schunter, M., Waidner, M.: Service-oriented Assurance - Comprehensive Security by Explicit Assurances. In: Proc. of QoP'05. (2005)
2. Karabulut, Y., Kerschbaum, F., Massacci, F., Robinson, P., Yautsiukhin, A.: Security and Trust in IT Business Outsourcing: a Manifesto. In: Proc. STM. ENTCS (2006)
3. Goth, G.: The ins and outs of it outsourcing. IT Professional **1** (1999) 11–14
4. Schaad, A., Moffett, J.: Delegation of Obligations. In: Proc. POLICY. (2002) 25–35
5. Bettini, C., Jajodia, S., Wang, X.S., Wijesekera, D.: Provisions and obligations in policy rule management. J. Network and System Mgmt. **11**(3) (2003) 351–372
6. Park, J., Sandhu, R.: The UCON ABC Usage Control Model. ACM Transactions on Information and Systems Security **7** (2004) 128–174
7. Pretschner, A., Hilty, M., Basin, D.: Distributed Usage Control. CACM **49**(9) (2006) 39–44
8. Hilty, M., Pretschner, A., Schaefer, C., Walter, T.: A System Model and a Policy Language for Distributed Usage Control. Technical Report I-ST-20, DoCoMo (2006)
9. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Hippocratic DBs. In: VLDB. (2002) 143–154
10. Karjoth, G., Schunter, M., Waidner, M.: Platform for Enterprise Privacy Practices: Privacy-enabled Management of Customer Data. In: Proc. PET. (2002) 69–84
11. W3C: The Platform for Privacy Preferences 1.1 (P3P1.1) Spec., Working Draft (2005)
12. Wang, X., Lao, G., DeMartini, T., Reddy, H., Nguyen, M., Valenzuela, E.: XrML–eXtensible rights Markup Language. In: Proc. XMLSEC. (2002) 71–79
13. Iannella, R.: Open Digital Rights Language - Version 1.1 (2002) odrl.net/1.1/ODRL-11.pdf.
14. Ligatti, J., Bauer, L., Walker, D.: Edit Automata: Enforcement Mechanisms for Run-time Security Policies. International Journal of Information Security **4**(1-2) (2005) 2–16
15. Hilty, M., Pretschner, A., Schaefer, C., Walter, T.: Enforcement for Usage Control—An Overview of Control Mechanisms. Technical Report I-ST-18, DoCoMo EuroLabs (2006)
16. Filman, R., Elrad, T., Clarke, S., Aksit, M.: Aspect-Oriented SW Development. (2004)
17. Erlingsson, U., Schneider, F.: SASI enforcement of security policies: A retrospective. In: Proc. New Security Paradigms Workshop. (1999) 87–95
18. Bauer, L., Ligatti, J., Walker, D.: Composing Security Policies with Polymer. In: Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation. (2005) 305–314
19. Zhang, X., Chen, S., Sandhu, R.: Enhancing Data Authentity and Integrity in P2P Systems. IEEE Internet Computing **9**(6) (2005) 18–25
20. Sandhu, R., Zhang, X.: Peer-to-peer access control architecture using trusted computing technology. In: SACMAT. (2005) 147–158
21. van Oorschot, P.: Revisiting software protection. In: Proc. IST. (2003) 1–13
22. van Oorschot, P.: SW protection and application security: understanding the battleground. In: State of the art and evolution of computer security and industrial cryptography. (2003)
23. W3C: A P3P Preference Exchange Language 1.0 (APPEL1.0) (2002)
24. Backes, M., Pfitzmann, B., Schunter, M.: A toolkit for managing enterprise privacy policies. In: Proc. ESORICS. (2003) 162–180
25. Povey, D.: Optimistic security: a new access control paradigm. In: Proc. workshop on new security paradigms. (1999) 40–45
26. Hilty, M., Basin, D., Pretschner, A.: On obligations. In: Proc. ESORICS. (2005) 98–117