Complex Event Processing for Usage Control in Service Oriented Infrastructures

Alexander Wahl, Stefan Pfister, Bernhard Hollunder
Department of Computer Science
Hochschule Furtwangen University
Furtwangen, Germany
alexander.wahl@hs-furtwangen.de, stefan.pfister@hs-furtwangen.de, bernhard.hollunder@hs-furtwangen.de

***Abstract*—Service oriented architectures (SOA) should adhere to clearly defined quality attributes, which are formalized using policies. Well-known attributes in the security realm are access control and usage control. Our approach is to analyze operations (e.g., data deletion) and data flows that occur within a SOA. We use the extracted information to monitor policies, especially usage control policies. We focus on usage control, which is by far not as well investigated as access control, but highly relevant for providers of sensitive data who do not want to lose control on their data. We show that there is a transformation that maps usage control formulas, formalized in an appropriate policy language, to rules based on Complex Event Processing (CEP) technology. We further argue that by the combination of a policy language, the CEP technology, sensor components and a transformation from policy language to CEP rules SOA infrastructures can be enabled for usage control.**

Keywords - Service Oriented Architecture; Web service; Usage Control; Complex Event Processing;Policies.

I. INTRODUCTION

Today, data are commonly exchanged in distributed computer systems. For some of these data stakeholders have increasing interest to control access to these data and to further control the usage of data once they are distributed. For example, medical data of patients generated by physicians during treatment are to be highly protected. Such data may be accessed by authorized persons only, like other physicians, but may usually not be handed over to any others without permission of the patient. In Germany this is regulated by law, e.g., the so-called "Bundesdatenschutzgesetz" [1]. Access to data is covered by access control, but to prohibit propagation of data a concept for usage control is required.

Access control deals with the question: Who may access data at first instance? For access control, there are well-known approaches, such as the role based access control (RBAC) principle [2].

A. *Usage control*

Usage control [3] deals with the question: what happens to data once they are given away? Distributed usage control [4] is an extension of usage control in distributed systems.

There are two main parties in usage control: *data providers* and *data consumers*. A data provider owns data and controls access to them. The data consumer wants to gain access to and perform operations on these data. Once the access is granted, data are handed over to the data consumer. Without usage control the data provider from then on has lost control on his data. The data consumer may perform unwanted operations on the data of which the data provider may not get informed. For example, confidential information intended to be used within a company and its suppliers would be passed to a competitor without being noticed. With usage control the data provider regains control on his data. He may now specify usage control rules.

Usage control rules describe the conditions a certain operation on data is allowed or prohibited. Such conditions are either provisions or obligations. Provisions are those that refer to the past and the present, respectively whether data may be released in the first place. Conditions that govern the present and future usage of the data are so-called obligations [5]. Typical examples for usage control rules are "Delete a particular document within 30 days", "Do not give data D to anybody else", "Data D may be copied at most 2 times", "Data D may only be sent if contract exists" or "Using data D for some purpose requires an acknowledgement of data provider". Usage control rules are of different types, as can be seen with the given examples. They either relate to time, cardinality, environment, purpose or occurrence of events [6].

A *usage control policy* is a formal representation of a usage control rule. It consists of usage control formulas. To describe usage control policies in a formally correct manner usage control policy languages were introduced, like e.g., Obligation Specification Language (OSL), Usage Control (UCON) and Extended Privacy Definition Tool (ExPDT). We will get back to these later.

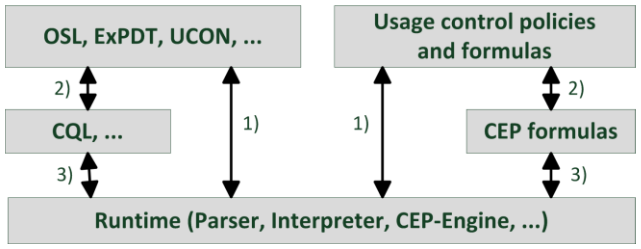
OSL, ExPDT, UCON, ...
Usage control policies and formulas
2)
1)
1)
2)
CQL, ...
CEP formulas
3)
3)
Runtime (Parser, Interpreter, CEP-Engine, ...)

Figure 1. Mapping of policies to runtime. 1) Direct mapping; 2) Intermediate mapping 3) Mapping to runtime.

Once usage control policies are formulated and applied to data, the infrastructure is in charge to ensure compliance to that sets of formulas, i.e. policies. That means that once a usage control formula is violated the execution of an operation should inform the data provider. Some usage control formulas, like “Do not give data D to anybody else” or “Data D may be copied at most 2 times”, may even hinder the execution of an operation (e.g., copying the data) and return a fault.

B. Using Complex Event Processing for Usage control

For usage control, especially for distributed usage control, the availability of appropriate technologies is still very limited. There are a few approaches on usage control, as we will see in section “Related Work”. They are either prototypic, proprietary, or they are limited to SOA infrastructures that contain specific components, such as enterprise service bus (ESB). All these approaches have in common that they enforce policy formulas, defined in a specific policy language, directly to the runtime system (see Figure 1). We think, that by introducing a solution based on a well-established and well-tested technology, like Complex Event Processing (CEP) [7], we can overcome the issues described before. We further argue that the usage of CEP simplifies the mapping from policy language formulas to a runtime environment.

In this work, we combine a usage control policy language, the CEP technology, sensor components and a transformation of usage control formulas to CEP rules. The novelty of our approach is that instead of mapping a policy directly to the runtime, e.g. by the usage of a proprietary interpreter [8], we introduce an intermediate step that maps a policy to CEP rules. The main advantages are the usage of a well-established and well-tested technology and the less complexity of mapping to the runtime, since the evaluation of CEP rules is performed by an already existing component, namely the CEP engine.

We use the Continuous Query Language (CQL) [9] to formulate the CEP rules. With CQL, the formula is still comprehensible once the formulas are transformed. The advantage is that the mapping from CEP rules to a runtime environment already exists. In addition our solution can be used to enable existing SOA infrastructures for usage control with minor modifications only. Our approach is based on the overall architecture described in [10].

This paper is structured as follows: Section 2 gives an overview on related work in the area of usage control. In Sections 3 we describe our approach in detail. In Section 4 we describe a strategy to transform usage control formulas into a technical representation. We further discuss events and show how CEP rules are evaluated based on events. We will also show that there is a direct relationship between formulas, events and necessary information to be extracted from the SOA infrastructure. Section 6 is about the relationship between CEP rules and policies. In the last section, we will conclude our work and give a forecast to our future work.

II. RELATED WORK

In this section, we will describe the most significant publications on usage control, usage control models and usage control policy languages.

Hilty, Pretschner and coauthors gave, in a series of publications, a detailed overview on enforcement of usage control [11, 12] and distributed usage control [4]. They introduce a usage control policy language called Obligation Specification Language (OSL) [6], monitors for OSL-based usage control [8] and usage control in service oriented architectures (SOA) [13]. OSL enables to formulate temporal descriptions of obligations. In “Monitors for Usage Control” a prototypical implementation of a Java based obligation monitor for OSL is mentioned [8]. In “Usage Control in Service-Oriented Architectures” they stated, that “Implementing the architecture is a next step” [13].

Park and Sandhu introduced the concept of usage control [14] and the ABC-Model for usage control ($UCON_{ABC}$) [3, 15]. This model integrates *A*uthorization (A), o*B*ligation (B) and *C*ondition (C) components. The latter, conditions, are environmental restrictions before or during usage of data. However, to our best knowledge they did not implement their approach for a SOA infrastructure.

Gheorghe et al. implemented a policy enforcement mechanism on ESB level [16]. They called it xESB, which is an ESB enhanced by an additional component based on Java Business Integration (JBI) [17]. First they used a specific policy language, but in a following publication they enforce UCON policies, respectively POLPA [18], which is used to implement the UCON model. This elegant approach is limited to SOA infrastructures using ESB technology. However, there are SOA infrastructures that are not implemented from scratch and that do not use ESB. And for those this approach is not applicable.

Kaehmer et al. [19–22] introduce ExPDT. With ExPDT, permissions, prohibitions and orders based on contextual conditions or obligations can be described. It enables for access and usage control, and also supports the comparison of two policies.

To sum up, there are several expressive policy languages for usage control. With these policy languages obligations, conditions, permissions, prohibitions and orders can be formalized, depending on their expressiveness. Also a few implementations to monitor usage or to enforce usage control in infrastructure are available. However, these are either prototypic, limit themselves by requirements on the existing SOA infrastructures, and use more complex transformations from the policy language to the runtime.

III. USAGE CONTROL POLICY LANGUAGES

To build a SOA and to apply usage control to it is a quite sophisticated task. If a SOA is built from scratch usage control mechanisms can be taken into account from the very beginning. An appropriate approach, like xESB, can be chosen. The same approach can be used for SOA infrastructures that already use an ESB. But in real world, most often there is an existing SOA that is to be enabled to usage control, as described in the example of the Hong Kong Red Cross by [23]. But unfortunately not all SOA infrastructures do use an ESB, and for those alternative approaches are needed.

Within an existing SOA based on Web service technology, service providers and service consumers exchange data. Thereby some of these data might be sensitive ones, like e.g., confidential patient data within a hospital, or data for internal use only within a company. To achieve usage control, such data are associated with usage control policies. Well-known candidates for specifying declarative policies are (among others): eXtensible Access Control Markup Language (XACML) [24], Extended Privacy Definition Tool (ExPDT) [20] (compliance validation, privacy preferences, permissions, prohibitions), Usage Control (UCON) [14], and Obligation Specification Language (OSL) [6].

XACML offers a policy language to describe general access control requirements. It offers a request/response language to determine whether an action is allowed or not. The focus of XACML is on access control. However, to a certain extend it can be used for usage control, especially if enhanced by additional features. U-XACML [25], for example, enhances XACML with UCON features.

OSL supports the formalization of a wide range of usage control requirements. It mainly focuses on obligation. The language contains propositional operators (AND, OR, NOT, IMPLIES), temporal operators (UNTIL, AFTER, DURING, WITHIN), cardinality operators (REPUTIL, REPMAX) and permit operators (MUST, MAY).

UCON is a general model for usage control. With UCON$_{ABC}$ a policy specification is provided to support pre- and post-authorization, obligation rules and conditions.

Finally, ExPDT is a policy language developed to define privacy preferences. It allows describing permissions, prohibitions and orders that are to be followed once certain contextual conditions are met or if obligations have to be fulfilled.

In this work, we will limit to a transformation from OSL formulas to CEP rules (due to the restricted number of pages). The expressivity of OSL is sufficient for the examples used in this work. Transformations from other languages, like ExPDT, will be future work.

IV. APPROACH

In general, a SOA consists of multiple collaborating entities: i) service providers and service consumers, ii) data providers and data consumers, iii) infrastructure, iv) data and v) events. Service providers offer some functionality that is utilized by service consumers. A service consumer itself may be a service provider for another service consumer (transitivity). Similar to that, data providers offer data to data consumer.

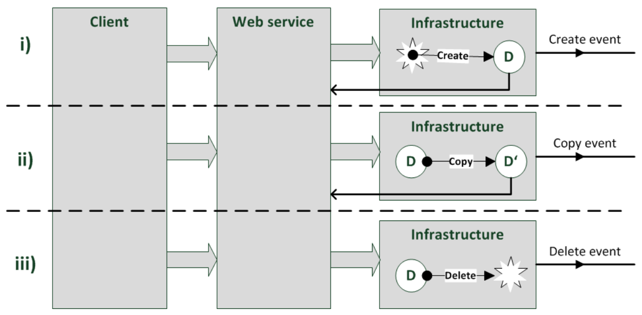

Figure 2. Operations on data produce events

Transitivity also applies here. The infrastructure is the collectivity of system components, frameworks, applications, etc. necessary to run the SOA.

Data are information sets that are generated, copied and deleted at the infrastructure (see Figure 2). Creating data means that information is passed to and stored within the infrastructure, e.g., adding a dataset for a person to a database. Copying data means, that data existing within the infrastructure is duplicated and sent elsewhere. For example, copied data are exchanged between data provider and consumer within the body of a message. Deleting data describes removing of data from the infrastructure. In any case the infrastructure is involved at any operation that is executed on data, and is therefore potentially able to inform about these operation.

An alternative to inform a third party, e.g., a usage control monitor, about an operation on data is the following one: At an existing SOA infrastructure so-called sensor components are applied at appropriate entities, e.g., a SOAP message handler attached to a Web service, a JBI component for an ESB (if part of the SOA), or a sniffer that analyses the traffic at the application servers port (see also [10]). The aim of the sensor component is i) to detect operations executed on data (either before or after the execution), and ii) to inform a third party by emitting an event.

In the context of sensor components an *event* is a message that identifies the executed operation and the affected data. It contains a timestamp and additional data, e.g., the principal of an operation. The event is emitted by the sensor component and received by the third party, like the usage control monitor. Within the usage control monitor the included formulas are evaluated based on the received events.

The usage control formulas to be evaluated are specified by a developer via a usage control policy language. Afterwards they are transformed by the developer to a representation the monitor is able to interpret.

For example: A service consumer calls a Web service in order to get a copy of a certain data. The Web service therefore calls the infrastructure for a copy of this data. Within the infrastructure, each time a data is copied a sensor component is involved prior to copying. This sensor component produces a copy event. This event, for example, includes information that identifies the data to be copied, a timestamp and principal of the copy request. The copy event is then emitted by the sensor component and received by a usage control monitor.

The usage control monitor is responsible to evaluate if the usage control formulas are satisfied. The usage control monitor, in essence, is the CEP engine equipped with formulas as required. The evaluation is performed based on the events received by the CEP engine.

V. Strategy

In the previous section, we described an approach that uses events (emitted by sensor components that are attached to the SOA entities) to evaluate a CEP rule, i.e., a transformation of a usage control policy. In this section we will consider in more detail the transformation from a usage control formula, specified in a policy language, to a CEP rule (Figure 1). We will further illustrate how a CEP rule is evaluated based on events. We will also show that there is a correspondence between CEP rules and the events that are necessary to determine if a CEP rule is satisfied, i.e., a corresponding usage control policy is satisfied.

The notion “satisfied” in the context of usage control policy means that its related formulas are fulfilled. In the context of CEP rules satisfied means that based on the collected events the preconditions of a CEP rule evaluate to true and the CEP rule fires. If a CEP rule fires a corresponding event is generated and emitted to a subsequent actor (see Figure 3). The actor then initiates a corresponding action, e.g., a deleting data.

Usage control is applied to a SOA by binding usage control policies to data using the sticky policies paradigm [26]. Each formula follows a pattern similar to “if <condition> then <action>” or “if <condition> then (not) <usage>” [6]. For example, assume the usage control formula “Delete document within 30 days”. This formula can be reformulated: “if document D will not be deleted within 30 days then indicate violation”. In OSL this formula is specified as follows:

```
Within(30days,delete,{(data,D)})
```

In this example *delete* specifies the event, *data* specified a parameter and *D* the value of the parameter *data*. The transformation of this formula to a CEP rule is based on the following considerations:

1) This formula specifies time duration of 30 days. So we have two points in time: a creation timestamp and deletion timestamp.
2) We want to get informed if the formula is violated.

The first consideration implies that we need to get informed on the creation of data and on the deletion of data. From that information we can derive that we will need two events to evaluate satisfaction of this formula: *create(D)* and *delete(D)*. We suppose that D is not copied in the meantime. *Create(D)* is emitted on data creation, *delete(D)* on data deletion.

As we already described, the infrastructure does have the potential to inform about an operation that is performed on data. So it needs to be enabled to emit events. We therefore modify the infrastructure by adding sensor components at appropriate positions.

A sensor component in brief is a piece of software that is attached to the SOA infrastructure at appropriate SOA entities. Sensor components can be of different types. They can be message handlers (e.g., SOAP message handlers), JMX client components, sniffers or even GUI elements. These sensor components have in common, that they collect and analyse actual data within the SOA infrastructure and emit these data as events. For a more detailed description please refer to [10].

Since sensor components are additional components one has to expect certain performance penalties once they are applied. However, the performance penalty for extracting data and emitting an event should be small. And since the expensive (in terms of execution time) evaluation of CEP rules can be performed in asynchronous manner by a third party, e.g. a dedicated machine running a CEP engine, the influence on performance can be kept to a minimum.

In our example we need to apply a sensor component that emits *create* events, and a second one that emits *delete* events. The events emitted by the sensor components are defined by a developer. His task is to analyze i) which information is necessary to evaluate the CEP rule, ii) from where in the SOA the information can be fetched. Based on this he defines the event types. Figure 2 shows a few exemplary event types. There are several other events one can think of, also complex events. For example the event *consumption(D)*, which might be defined as *create(D) OR copy(D)* $\rightarrow$ *consumption(D)*. In words, each time D is created or copied, a consumption event is produced. With that further usage control formulas can be formulated, like “IF *consumption(D) THEN check(contract)*” or “IF *consumption(D) THEN inform(data provider)*”.

The events are collected by corresponding event streams in the CEP engine, i.e. the input of a CEP rule. Each CEP rule has its own set of event streams. By these streams the CEP rule defines which events must be emitted by the infrastructure and the sensor components, respectively. So, for each CEP rule there is a well-defined set of events to evaluate its satisfaction. In consequence, there is a direct relation between CEP rules and events. A CEP rule is transformed from a usage control policy formula. Therefore the relation to events also applies the latter (and finally to the verbal formulation of them).

In the example “Delete document within 30 days” stressed before there are two event streams, namely: CreateStream (contains all collected events *create(D)*) and DeleteStream(contains all events *delete(D)*). These two event streams are the input of our CEP rule. An instance of an event *create(D)* is named createEvent, respectively deleteEvent for *delete(D)*.

As considered previously, we want to get informed on violation. By the way, it is also possible to indicate that the formula is satisfied. In either case we can again use events to denote violation or satisfaction. In our example we define an event *violation(D)* that is emitted by the CEP rule once the formula is not satisfied. In other words, the output of our CEP rule is an instance of *violation(D)*, namely violationEvent.

A CEP rule defines how events are correlated. Consider the OSL formula “Within(30days,delete,{(object,D)})”.

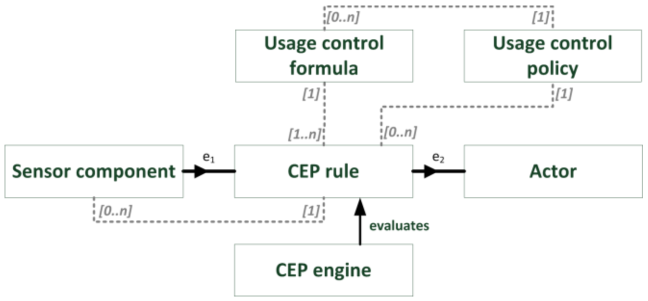

Figure 3. Relations between CEP rules and usage control policies.

Based on the information of the former text this policy can be translated into a CEP rule:

```
SELECT createEvent, deleteEvent
FROM CreateStream, DeleteStream
WHERE createEvent.Data = deleteEvent.Data
AND deleteEvent.Time - createEvent.Time > 30 days
THEN CREATE violationEvent
```

In other words: createEvents from CreateStream and deleteEvents from DeleteStream that operate on same data, and whose timespan between creation and deletion is greater than 30 days cause an violationEvent to be created and emitted.

It is obvious, that if events of *create(D)* and of delete(D) are not emitted by sensor components the formula cannot be evaluated and fire a violationEvent.

Summing up, a SOA infrastructure has to be enabled for CEP based usage control. Therefore, sensor components have to be installed in the SOA at appropriate positions. Sensor components analyze information within the SOA and emit events. Events are collected and correlated to evaluate CEP rules. There is a direct relation between a CEP rule and the events needed. Since the CEP rule is a transformation of a policy language formula this relation also applies to the latter.

VI. CEP RULES AND POLICIES

Between usage control policies, usage control formulas and CEP rules there are several relations, as depicted in Figure 3.

First, a usage control policy is a set of usage control formulas, i.e., it consists of zero or more formulas. A usage control policy is (usually) satisfied iff all of its usage control formulas are satisfied.

A usage control formula can be described by one or more CEP rules. So a usage control formula is satisfied iff all the corresponding CEP rules are satisfied.

If a CEP rule is satisfied a corresponding events is generated and emitted. The event (e_2 in Figure 3), or a set of events, is used to trigger a related actor that executes a related action.

A CEP rule is evaluated by the CEP engine based on a set of events (e_1 in Figure 3). So for each CEP rule the number and kinds of events that are necessary to evaluate it is known.

Since the kinds of events to evaluate a CEP rule are known, the sensor components that need to be installed in the SOA infrastructure is also known.

Finally, a usage control policy can be represented by a set of CEP rules. That is because a usage control policy consists of a set of usage rules, and a set of usage control rules can be represented by a corresponding set of CEP rules. So, a usage control policy is satisfied iff a corresponding set of CEP rules.

VII. CONCLUSION AND FUTURE WORK

A. *Summary*

The mapping of usage control formula to the runtime is a difficult and complex task. The introduction of an intermediate step, as shown in Figure 1, is reasonable and brings advantages. By using CEP, the mapping of a usage control policy to the runtime is reduced to mapping to a CEP rule. In this work we mapped an exemplary OSL formula to CEP rule. However, we think that this is also feasible for other policy languages, which is future work. This mapping can be performed more easily. The satisfaction of a CEP rule to runtime is determined by the CEP engine. However, it is necessary to install sensor components in the SOA infrastructure. With these sensor components the SOA is enabled to emit events on operations.

Using CEP the requirements to an SOA infrastructure and the necessary changes within to enable for CEP are kept to a minimum. The approach does not require special CEP components, like e.g., ESB, to be applicable. It is flexible to apply to a variety of SOA infrastructures. Just sensor components need to be applied. The sensor components and events needed to evaluate satisfaction of a CEP rule are in a direct relation. However, currently the sensor components need to be implemented, configured and installed manually.

With CEP not only single formulas can be mapped, but also whole policies. This is interconnecting CEP rules with each other.

B. *Perspective*

The perspective of our work is to enrich existing systems by Quality of Service (QoS) attributes insufficiently supported or yet unsupported at all. We see usage control as one of these QoS attributes. Based on an architecture described elsewhere [10] we currently implement exemplary usage control formulas using CEP technology, and the appropriate sensor components.

We further work on an implementation of a tool chain that supports developers to equip existing SOA infrastructures with QoS attributes [27]. Also a part of these efforts is to automate the transformation from a policy (formula) to a CEP rule. Beside these steps we also plan to include further analysis, like the influence of sensor components on the performance.

Finally, the statements within Section 6 need to be formulated and proved in a more formal manner. Also, this will be part of our future work.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for giving us helpful comments. This work has been partly supported by the German Ministry of Education and Research (BMBF) under research contract 17N0709.

REFERENCES

[1] Bundesministerium der Justiz, Bundesdatenschutzgesetz (BDSG). Available: http://www.gesetze-im-internet.de/bdsg_1990/index.html, last accessed 2011, June 27.

[2] M. Benantar, *Access Control Systems*: Gardners Books, 2010.

[3] J. Park and R. Sandhu, "The UCONABC usage control model," *ACM Trans. Inf. Syst. Secur*, vol. 7, pp. 128-174, 2004.

[4] A. Pretschner, M. Hilty, and D. Basin, "Distributed usage control," *Commun. ACM*, vol. 49, pp. 39-44, 2006.

[5] M. Hilty, D. Basin, and A. Pretschner, "On Obligations," in *Lecture Notes in Computer Science, Computer Security – ESORICS 2005*, S. Di Vimercati, P. Syverson, and D. Gollmann, Eds.: Springer Berlin / Heidelberg, 2005, pp. 98–117.

[6] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter, "A Policy Language for Distributed Usage Control," in *Lecture Notes in Computer Science, Computer Security – ESORICS 2007*, J. Biskup and J. López, Eds.: Springer Berlin / Heidelberg, 2007, pp. 531–546.

[7] D. C. Luckham, The power of events: An introduction to complex event processing in distributed enterprise systems. Boston: Addison-Wesley, 2002.

[8] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter, "Monitors for Usage Control," in *IFIP International Federation for Information Processing, Trust Management*, S. Etalle and S. Marsh, Eds.: Springer Boston, 2007, pp. 411–414.

[9] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: semantic foundations and query execution," *The VLDB Journal*, vol. 15, pp. 121-142, http://dx.doi.org/10.1007/s00778-004-0147-z, 2006.

[10] A. Wahl, A. Al-Moayed, and B. Hollunder, *An Architecture to Measure QoS Compliance in SOA Infrastructures.* Available: http://www.thinkmind.org/index.php?view=article&articleid=service_computation_2010_2_10_20064, last accessed 2011, June 27..

[11] Manuel Hilty and Er Pretschner et al, Enforcement for Usage Control- An Overview of Control Mechanisms Deliverables 1 and 2 DoCoMo Euro-Labs Publication, last accessed 2011, June 27.

[12] A. Pretschner, M. Hilty, F. Schutz, C. Schaefer, and T. Walter, "Usage Control Enforcement: Present and Future," *Security Privacy, IEEE, title=Usage Control Enforcement: Present and Future*, vol. 6, no. 4, pp. 44–53, 2008.

[13] A. Pretschner, F. Massacci, and M. Hilty, "Usage Control in Service-Oriented Architectures," in *Lecture Notes in Computer Science, Trust, Privacy and Security in Digital Business*, C. Lambrinoudakis, G. Pernul, and A. Tjoa, Eds.: Springer Berlin / Heidelberg, 2007, pp. 83–93.

[14] J. Park and R. Sandhu, Eds, Towards usage control models: beyond traditional access control.

[15] R. Sandhu and J. Park, "Usage Control: A Vision for Next Generation Access Control," in *Lecture Notes in Computer Science, Computer Network Security*, V. Gorodetsky, L. Popyack, and V. Skormin, Eds.: Springer Berlin / Heidelberg, 2003, pp. 17–31.

[16] G. Gheorghe, S. Neuhaus, and B. Crispo, "xESB: An Enterprise Service Bus for Access and Usage Control Policy Enforcement," in *IFIP Advances in Information and Communication Technology, Trust Management IV*, M. Nishigaki, A. Jøsang, Y. Murayama, and S. Marsh, Eds.: Springer Boston, 2010, pp. 63–78.

[17] Sun Java Community Process Program, *Sun JSR-000208 Java Business Integration.* Available: http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html, last accessed 2011, June 27.

[18] F. Baiardi, F. Martinelli, P. Mori, and A. Vaccarelli, "Improving Grid Services Security with Fine Grain Policies," in *Lecture Notes in Computer Science, On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops*, R. Meersman, Z. Tari, and A. Corsaro, Eds.: Springer Berlin / Heidelberg, 2004, pp. 123–134.

[19] Martin Kähmer and Maike Gilliot, *Extended Privacy Definition Tool.* Available: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-328/paper12.pdf, last accessed 2011, June 27.

[20] M. Kahmer, M. Gilliot, and G. Muller, Eds, *Automating Privacy Compliance with ExPDT*, 2008.

[21] M. Kahmer and G. Muller, Automating Privacy Compliance for Personalized Services.

[22] M. Kähmer, ExPDT: Vergleichbarkeit von Richtlinien für Selbstregulierung und Selbstdatenschutz, 1st ed. Wiesbaden: Vieweg + Teubner, 2010.

[23] G. Gheorghe, B. Crispo, D. Schleicher, T. Anstett, F. Leymann, R. Mietzner, and G. Monakova, "Combining Enforcement Strategies in Service Oriented Architectures," in *Lecture Notes in Computer Science, Service-Oriented Computing*, P. Maglio, M. Weske, J. Yang, and M. Fantinato, Eds.: Springer Berlin / Heidelberg, 2010, pp. 288–302.

[24] eXtensible Access Control Markup Language (XACML), 2005.

[25] M. Colombo, A. Lazouski, F. Martinelli, and P. Mori, "A Proposal on Enhancing XACML with Continuous Usage Control Features," in *Grids, P2P and Services Computing*, F. Desprez, V. Getov, T. Priol, and R. Yahyapour, Eds.: Springer US, 2010, pp. 133–146.

[26] G. Karjoth, M. Schunter, and M. Waidner, "Platform for enterprise privacy practices: privacy-enabled management of customer data," in *Proceedings of the 2nd international conference on Privacy enhancing technologies*, Berlin, Heidelberg: Springer-Verlag, 2003, pp. 69-84.

[27] B. Hollunder, A. Al-Moayed, and A. Wahl, "A Tool Chain for Constructing QoS-aware Web Services," in *Performance and dependability in service computing: Concepts, techniques and,* V. C. E. Cardellini, K. R. L. J. C. Branco, J. C. Estrella, and F. J. Monaco, Eds, Hershey: Information Sci Refer Igi, 2011.