Logging in Distributed Workflows

Christoph Ringelstein and Steffen Staab

ISWeb Working Group
University of Koblenz-Landau
Universitätsstraße 1, 56070 Koblenz, Germany
{cringel,staab}@uni-koblenz.de
http://isweb.uni-koblenz.de

Abstract. Business needs are nowadays frequently realized by business workflows spanning multiple organizations. Current infrastructures, such as SOA, support this trend on the implementation side. As a side effect these issues of privacy and data protection arise, because data is shipped across organizational boundaries. At the same time increased awareness about protection of privacy and IPR have lead to comprehensive contractual and legal constructs - including the information of services consumers about the ways their data is handled. We propose to solve such information requests in widely distributed workflow executions by gathering the related information during the execution and attaching it directly to the processed data. Together with the data this information is passed through the workflow and at the end it is returned to the service consumer.

1 Introduction

Applying the paradigm of service-orientation in an organization implies the modeling of business capabilities as independent services. A service-oriented architecture provides standardized interfaces for the communication between services. The standardization enables the loose coupling of services, supports the service provisioning for external service consumers and eases the integration of external services into internal workflows. The resulting flexibility facilitates the combination of services from different organizations into one comprehensive, integrated workflow leading at the organizational level to an agile *virtual organization* [13, 5] that is able to adapt more quickly to new organizational or business needs.

However, the resulting flexibility also shows disadvantages. An integrated workflow may forward confidential data (e.g. personal data or intellectual property) between organizations potentially violating concerns of privacy protection or confidentiality. Under such circumstances of flexible interworking between organizations, the importance of accounting for actions performed on data may become legally and/or contractually a highest priority.

To enforce accountability the privacy laws of some countries, e.g. all countries in the EU (as defined in Directive 95/46/EC), oblige organizations to *notify* about the processing of personal data (which can be described by *privacy policies*) and entitles natural persons to request information about the processing of their personal data (this process

is called *information by request*). As a second case, one may consider contractual obligations about data protection between organizations. An organization that uses a web service like Salesforce^{TM} customer-relationship management software may not want to share the data about its customers with its competitors who use the same service. Hence, a service provider may offer the possibility to retrieve data accounting information and thus to allow for some control of the data processing.

To be able to monitor agreed-upon policies the service consumer (e.g. a natural person or another organization) may request information about the processing of his data. The answer to this request must contain *who* processed the data as well as *why* and *how* the data has been processed. This information constitutes an abstracting log of the workflow execution. The log can be generated in different ways, e.g. by reconstructing or by monitoring the workflow execution. In any case the service provider needs a detailed overview of its workflows. Most frequently such an overview is lacking, even for internal workflows. For several reasons existing logging mechanisms, like the Extended Log File Format [7] or *syslog*[12], are not sufficient to gain a full overview of a workflow that is distributed among multiple organizations. The main reason is that existing logging mechanism are tailored to perform logging within one execution environment. Because the diversity of execution environments and the missing of standardized interfaces for exchanging logs, distributed logs can not automatically be combined into one log. To solve this problem we propose a logging mechanism[1] that is tailored to log all actions on a specific piece of data during its processing in a distributed workflow. To bind the logged information to the corresponding data instances the proposed mechanism attaches the logs directly to the data instances, as metadata. Therefore, according to the 'sticky policy paradigm' introduced in [9], we call this approach 'sticky logging'. The sticky logging mechanism is designed to be used in a service-oriented architecture.

The aim of this paper is to present the actual state of work at the sticky logging mechanism. The mechanism consists of two parts: First, an architecture defining rules how to collect information about the processing of private data in distributed workflows; and second, a formalism specifying how to express the collected information. Parts of the mechanism to achieve accountability, like tamper and fraud resistance, are work in progress and will be presented in follow-up papers.

This paper is organized as follows: In section 2 we analyze general requirements for a mechanisms to collect information about the processing of private data in distributed workflows. Following the requirements we introduce the architecture in section 3 and in section 4 the formalism of the sticky logging mechanism. Then, we give a scenario in section 5 demonstrating the application of the sticky logging mechanism. Before we eventually discuss our approach and conclude in section 7, we compare it with related work in section 6.

2 Requirements Analysis

In this section we present a business case where the realization of an SOA architecture includes a workflow over several organizations and we identify legal requirements.

[1] This mechanism can be used in addition to standard logging mechanisms that are needed to maintain the technical functionality of a workflow environment or service.

From the business case and the legal requirements we derive some issues that arise with regard to data protection. Then, we derive technical requirements for collecting privacy-related information about data processing in a cross-organizational workflow.

2.1 Business Case

The Small-Books Inc. is a book-selling company. Parts of the logistics like storing the books and packaging are done by Small-Books Inc. itself, but for shipping and payment it uses services provided by other organizations. For instance, the shipping of orders is outsourced to a logistics company named Fast-Shipping Inc.

Assume that a customer, Mr. Smith, orders books via the Web site of Small-Books Inc. To place his order he has to insert his address and credit card number. After the order is placed Small-Books Inc. possesses three instances of data related to the order: the list of ordered books, Mr Smith's address and credit card number. As first action Small-Books Inc. uses the payment service of the credit card company. To this purpose, Small-Books Inc. passes the credit card number and the invoice value to the payment service. Then, Small-Books Inc. packages the order and invokes the delivery service of Fast-Shipping Inc. To this end, Small-Books Inc. has to copy the address to pass it as new data instance to Fast-Shipping Inc.

2.2 Legal and Contractual Requirements

In the introduction, we have described why an organization must take responsibility for its way of handling of personal data. In addition it must be able to inform the person concerned about this handling. These implies the following requirements:

Req1 *The person concerned must be able to request or directly access the logs.*
Req2 *The logs must identify the organizations responsible for each action performed on personal data.*
Req3 *An organization must not be able to deny a log entry it has made.*

2.3 Organizational Issues with Accountability

From the business case and above-derived requirements we may derive the following organizational issues.

Loosely-coupled Architectures: At the level of implementation, the use of a service-oriented architecture hampers the generation of an overview. The reason is that services in an SOA are defined and implemented independently of each other. Hence, cross cutting concerns, such as logging, are hard to realize if they are not standardized at the interface level. Also, workflows can be configured in an agile manner making it difficult *a posteriori* to assert which organizations had accessed the data during the execution of the workflow.

Lack of Process Awareness: In order to report on previous handling of data, an organization must be aware of and account for their internal data flows at a very fine-grained level of granularity. Such awareness and accounting at the fine-grained level

is rarely available explicitly for cross-organizational workflows, because the increased complexity of the workflows [14].

Autonomy of Organizations: With organizational autonomy transparency decreases, as the autonomous organizations in general do not want their workflows to be controlled or monitored by third parties. At the level of cross-organizational workflow the issue of a lack of process awareness is further aggravated by the organizational responsibility being distributed over autonomous entities.

2.4 Technical Requirements

A technical solution for accountability must address the three organizational issues identified above. To solve these issues we demand the following technical requirements:

Req4 *To avoid ambiguities and to reach standardization, the logs must be formalized using a language with a well-defined semantics.*

Req5 *The used language must be able to express details about the performed actions, their actors, their purposes and their order. These details must have the required level of granularity.*

Req6 *A standardized interface is required to access and share logs between all involved parties (see [14]).*

Req7 *The logging implementation must enable an organization to create logs containing all privacy-related details.*

Req8 *Organizations must be able to hide process internals from third parties. Thus, both implementation and formalization must support security mechanisms.*

Additionally there exist other requirements, like trustworthiness of organisations. Even if those requirements are highly important for logging in general and for accountability in special, they are not considered in this work [2].

3 An Architecture for Distributed Logging

The above-identified issues affect the organizations ability to integrate external services in general and specifically to inform about actions performed on personal data. In the following we propose a logging architecture for distributed workflows to fulfill the above-defined requirements.

As demanded by requirement *Req1*, the person concerned must be able to access the logs about the processing of its data. However, to access the logs it is essential to know where the logs are located and how to access them. In addition, requirement *Req5* demands a detailed overview of the processing of the data. To fulfill these requirements we propose to attach all logs of a single execution of a workflow directly to the processed data, as metadata. The logs are attached by including them into the SOAP [6] messages that also transfer the associated data. Including the logs into the messages is reached by means of a SOAP module[3].

[2] Those requirements will be tackled in later work.

[3] The definition of the module is work in progress.

After the process execution the logs are made accessible to the person concerned by means of a mechanism we call back-propagation (see below). Because the logs are attached to the associated data instances, we call this approach *sticky logging*. The sticky logging mechanism demands specific actions when certain events occur. In the following a short overview of these actions is given:

- **Creation of data instances:** Each data instance gets its own sticky log starting with an entry recording various pieces of information (see section 4) about its creation. In addition, each sticky log gets its own UUID. If the created data instance contains data taken from other data instances, the source instances have to be referenced.
- **Processing and changing of data instances:** Each processing or changing of a data instance has to be logged by means of the formalism introduced in section 4.
- **Copying of data instances:** Each copying of a data instance is associated with the creation of a new instance of the data. To enable the back-propagation of logs and the updating of references (see below) in both sticky logs (the one attached to the original instance of the data and the newly-created one, which is attached to the new instance of the data) a reference to the other sticky log has to be made. A reference contains the UUID of the log and the identifier of the organization that possesses or receives (see passing of data instances) the data. After the data is copied further log entries are only added to the associated sticky log.
  If the newly-created data instance is used for purposes other than processing by the workflow (e.g. storing in a database), an alternative method for the back-propagation (e.g. e-mail, etc.) has to be specified. This method has to be used when the normal reference needed for back-propagation becomes unusable.
- **Merging of data instances:** To merge data instances the instances are processed (see above) with the purpose to create a new instance containing the result of the merge. The creation itself is handled like a normal creation (see above).
- **Passing of data instances:** To pass an instance of data (or parts of it) as parameter of a service call it (or parts of it) has to be copied (see above) and then the copy is passed to the called service. With the copy the newly-created log[4] is transfered, too. As mentioned before the transportation of the log is achieved as part of the SOAP message that is used to call the service and is transferred anyway.
  When a service is called synchronously, the log is back-propagated by means of the SOAP answer after the execution. In contrast, if a service is called asynchronously, this has to be logged explicitly in the log of the original data instance. In addition, because the SOAP answering message can not be used for back-propagation, an alternative method for the back-propagation has to be specified in the newly-created log (see above). In both cases, the organization has to integrate the log into the one of the original data instance after receiving the log.
- **Deleting of data instances:** Deleting a data instance does not cause the deletion of the sticky log. Instead the log has to be back-propagated. The back-propagation is reached by merging the log with the one referred by the reference specified in the log (see copying of data instances). The merging may be reached by means of

[4] Because only the new log is transfered, the provider of the called service gets no insight into internals of the calling organization.

directly accessing the referred log as long as both logs are possessed by the same organization. If the referred log is possessed by another organization, the log of the deleted instance may be returned by means of the SOAP answer. This is feasible as long as the deletion occurs during the execution of a service on behalf of the organization possessing the source instance. In the case that there is no SOAP answer (e.g. because the service has been called asynchronously) the alternate method is used for back-propagation. After merging the logs, all references contained in logs of direct copies of the deleted data instance have to be updated to refer to the merged sticky log.

If the deleted instance is the last instance of a specific piece of data, the log is directly returned to the person or organization that initially created the first instance of this data. This person or organization is responsible to answer requested information to the person concerned.

Beside those actions the sticky logging mechanism makes use of signatures and encryption. The signatures are used to assure that the log is not modified by another organization on its way. For this purpose each logging entity has to sign its log entries by means of a digital signature mechanism. The encryption of logs is possible, because, if the privacy law or the contract demands it, the logs may contain confidential information about the internals of an organization. This information has to be provided to the person concerned, but should not be accessible by other organizations that transfer the log back to the person concerned. We propose to use an encryption mechanism basing on a public-key cryptography algorithm. Such algorithms allow the logging organization to encrypt the logs with the public key of the person concerned or of organizations that are allowed to access the logs.

4 Logging Formalism

To enable a semi-automated analysis of logs we specify the logs by means of a RDF-based [10] semantic formalism. In the following we introduce parts[5] of this formalism.

Instances of Data: Each data may have multiple instances (e.g. through copying). Even if all instances have their own sticky logs, they have to be clearly identifiable. This is because the logs of different instances will be combined when a data instance is deleted (see below). To represent the instance of a data we introduce the **DataInstance**-class that has among others the following properties:

- **hasUUID:** A unique id that clearly identifies this data instance. To assure that the id is unique we use UUIDs [11].
- **isCopyOf:** If this is a copy, this property links to the original instance.
- **hasCopy:** If this data instance has been copied, this property links to the copy.

[5] The complete formalism is accessible at http://isweb.uni-koblenz.de/Research/SOA/StickyLogging

Listing 1 depicts a snippet of the log that is attached to the data instance of Small-Books Inc. containing Mr. Smith's address information[6]. In line 1 to 3 this instance is specified.

Actions on Data Instances: The actions performed on the data instances are represented by the class **Action**. This class has among others the following properties:

- **isOfKind:** The kind of this action.
- **hasPurpose:** The purpose why this action has been performed.
- **performedOnDataInstance:** The data instance the action is performed on.

Line 4 and 7 of Listing 1 depict how Small-Books Inc. logs that an action has been performed on the data instance containing Mr. Smiths address information.

Log Entries: The class **LogEntry** represents one log entry. Whereas, a log entry contains the recording of all information about one single action performed on the processed data. Some of the properties of the *LogEntry*-class are:

- **hasUUID:** A unique id that clearly identifies this log entry.
- **logsAction:** The action logged by this entry.

In Listing 1 lines 8 to 10 depict the beginning of a log entry recorded by the Small-Books Inc.

Kind of Actions: The introduced formalism builds upon the two basic actions that can be performed on data: reading and writing. These actions are subdivided into few sub categories representing different reasons causing the action. We distinguish two reasons read-actions can have: First, reading data to *use* it, e.g. by a service to fulfill its purpose. Second, reading data to *copy* it - including the copying of data to invoke another service. Although copying is a special kind of using data, we define copying as distinguished action. This is because of the specialty that a new instance of the data is created as result of the copy-action. Thus, we introduce the following classes representing these read-actions:

- The class **UseAction** represents all read-actions that are performed on the data, beside read-actions made to copy data.
- The **CopyAction**-class represents the read-action that is made to read the data that is to be copied.

In addition, we distinguish three reasons that cause write-actions: Writing data to *create* a new data instance, writing data to *change* an existing instance, and writing data to *delete* a data instance. The create-action has been distinguished from the change-action, because during the create-action a new data instance is created while the change-action only modifies an existing one. The write-actions are present by the following classes:

- The **CreateAction**-class represents the action that is used to create a data instance.

[6] In the following examples we use *sl:* as namespace for the classes and properties defined as part of the sticky logging formalism.

- All actions that change the data are represented by means of the class **ChangeAction**.
- The class **DeleteAction** represents the action performed to delete a data instance.

Within our business case Small-Books Inc. invokes the shipping service of Fast-Shipping Inc. to ship Mr. Smith's order. Therefore, it creates a new instance of the address data by means of a copy-action (see lines 3 and 6 in Listing 1). In the associated sticky logs Small-Books Inc. connects the two instances by means of the properties *hasCopy* and *isCopyOf*. In parallel Small-Books Inc. creates a new sticky log for the actions that will be performed on the copy of the address data (like Small-Books Inc. has done in lines 1, 2 and 8 to 10 in Listings 1 for the first instance). Then the new instance with its new sticky log is passed as part of the service invocation to Fast-Shipping Inc.

Purpose of Actions: Besides the kind of an action its purpose is of importance, to check if a performed action has been justified. The possible purposes of an action depend on the service and thus on its domain. For instance, actions of services of the delivery domain may have purposes like the delivery of an order, tracing of an order, etc. Thus, to enable a flexible definition of purposes, concepts defined in domain ontologies based on an upper ontology for privacy[7] are used.

Line 7 of the log snipped in Listing 1 shows that Small-Books Inc. has performed the above logged copy-action for delivery purposes (*dofd:DeliverOrder*[8]).

Accountability: Fundamental for achieving accountability is the explicit identification of the entity that performs actions on the data instance. In addition, an organization has to be clearly associated with all log entries it is responsible for and its log entries may not be modified or changed by another entity. Thus, we introduce the **Entity**-class to represent an entity within a sticky log. In addition, the formalism demands the use of mechanisms to sign RDF graphs like the one described in [3]. Among others the class *Entity* has the following properties:

- **hasName:** The name of this entity.
- **hasID:** A legally effective identifier of the entity (e.g. trade register number, international securities identifying number (ISIN), etc.).
- **hasLogged:** The log entries the entity is responsible for.
- **hasPGPCertificate:** A link to the entities PGP certificate.
- **Signature:** The signature that signs all log entries connected with this entity.

The lines 14 to 19 in the example in Listing 1 depict how the Small-Books Inc. identifies itself by instantiating the *Entity*-class and setting its properties.

```
01 :addressDI1 rdf:type    sl:DataInstance
02 :addressDI1 sl:hasUUID "a26f4580-39d9-11dc-a3fa-..."
03 :addressDI1 sl:hasCopy :addressDI2

04 :action1 rdf:type                      sl:action
05 :action1 sl:performedOnDataInstance :addressDI1
```

[7] Because of the limited space, this ontology will be introduced in detail in a technical report, which is under work.

[8] :dofd is the namespace of the domain ontology for delivery.

```
06 :action1 sl:isOfKind                  sl:CopyAction
07 :action1 sl:hasPurpose                dofd:DeliverOrder

08 :logEntry1 rdf:type       sl:LogEntry
09 :logEntry1 sl:hasUUID     "cf2f30c0-39c1-11dc-80ae-..."
10 :logEntry1 sl:logsAction  :action1

11 :addressDI2 rdf:type      sl:DataInstance
12 :addressDI2 sl:hasUUID    "d3020270-3ab8-11dc-a179-..."
13 :addressDI2 sl:isCopyOf   :dataInstance1

14 :entity1 rdf:type              sl:Entity
15 :entity1 sl:hasName            "Small-Books Inc."
16 :entity1 sl:hasID              "ISIN US0001234567"
17 :entity1 sl:hasLogged          :logEntry1
18 :entity1 sl:hasPGPCertificate  "http://sbi.de/cert.asc"
19 :entity1 sl:Signature          "HrdSDFc..."
```

Listing 1. Example of a Sticky Log.

5 Scenario

In this section we step action-by-action trough our above-introduced business case to demonstrate the functionality of the sticky logging mechanism:

In our business case Small-Books Inc. possesses three data instances: One instance of the list of ordered books, one instance of the address data, and one instance of the credit card information. For each of these instances Small-Books Inc. creates a log. In each of these logs Small-Books Inc. identifies itself as logging entity. For all log entries Small-Books Inc. uses the above-defined semantic formalism (*Req4* and *Req5*).

As first step Small-Books Inc. processes the order list to package the order. This use-action and its purpose (packaging) are logged by Small-Books Inc. After the packaging is finished Small-Books Inc. hands the package over to Fast-Shipping Inc. To this purpose Small-Books Inc. copies the address data and transfers it to Fast-Shipping Inc. The transfered is reached by means of the SOAP message used to call the service. Thus, a standardized way to exchange logs is used (*Req6*).

The copy-action and its purpose (delivery) are logged by Small-Books Inc. In parallel a new log is created by Small-Books Inc. that is transfered to Fast-Shipping Inc. Both organizations need to be identified in the new log. Small-Books Inc. as responsible for the copy action and Fast-Shipping Inc. as receiver of the data instance. All organizations processing the data identify themselves in the logs (*Req2*). Because the delivery service is called asynchronous, a alternative return method is specified in the newly-created log.

As next step Fast-Shipping Inc. uses the address data to deliver the package. This use-action and its purpose (delivery) are logged by Fast-Shipping Inc. into the log attached to its instance of address data. For the logging Fast-Shipping Inc. uses also the above-defined semantic formalism (*Req4* and *Req5*). After the package is successfully

delivered Fast-Shipping Inc. deletes its instance of the address data. The delete-action is also logged by Fast-Shipping Inc.

Before returning the log, Fast-Shipping Inc. signs the log entries it has recorded. Then Fast-Shipping Inc. returns the log by means of the specified alternative return method to Small-Books Inc. Small-Books Inc. integrates these log into its own log of the its instance of Mr. Smith's address data. At the end of the processing Small-Books Inc. signs its log entries also. All involved organizations had to sign their log entries (*Req3*). By means of the logs Small-Books Inc. generates an information page that is accessible by Mr. Smith. This Web page contains all information about the processing of Mr. Smith address data (*Req1*). In addition, all actions, their actors, and their purposes can be identified by Mr. Smith (*Req7*).

Finally, Fast-Shipping Inc. did not have insight into internals of Small-Books Inc. The other way Fast-Shipping Inc. could have used the private key of Mr. Smith to encrypt his logs. This way Small-Books Inc. also would have no insight into internals of Fast-Shipping Inc. (*Req8*).

6 Related Work

An example for another work identifying the need for accountability for data usage in distributed environments is presented by Weitzner et al. [15]. The authors find that access restrictions are not sufficient to reliably achieve policy goals, because a certain piece of information is often available at more than one point in the Web. Thus, it can not be guaranteed that a specific agent has no access to a certain piece of information. Therefore, the authors demand transparency of information usage to enable accountability.

To reach accountability, different approaches exist that propose the use of auditing mechanisms: For instance, the authors of [4] propose to use an auditing mechanism to achieve accountability, because the enforcement of policies is difficult in a distributed environment. Therefore, they introduce a framework consisting of a semantic policy language and an associated proof checking system. The policies are used to describe the permissions of agents. The auditing is done based on policies and logged actions, conditions and obligations. In difference to our approach the logs are located at the agent and not at the data. In addition, the logged information is used to audit the actions of the agent, while the purpose of our logging is the auditing of the entire processing of a certain piece of data. A similar approach is presented by Barth et al. [2]. They introduce a logic that can be used to specify and verify privacy and utility goals of business processes. In difference to our formalism their logic is not designed to be used in an inter-organizational environment and the logs are stored at the single agents and not with the data.

One example for an approach to log the communication of data is the Extended Log File Format [7]. The Extended Log File Format is one of the most prominent non-semantic logging formalisms for Web applications. This formalism is tailored to log the technical functionality of Web applications and their communication. Hence, the Extended Log File Format is not sufficient to describe actions performed on data and their purposes.

An approach to enrich existing logs with semantics is presented by Baker and Boakes in [1]. Their approach enables a more common understanding of the logs and thus helps to solve the problem of different taxonomies. However, this approach uses semantics only to enrich existing logs and thus no additional information is gained and the logs have still the restrictions identified above (e.g. missing connection with concrete workflow, etc.).

Karjoth et al. introduce in [9] the sticky policy paradigm. When data is transmitted to an organization via Web form the applicable policies and the users opt-in and opt-out choices are also included into the form. If the data is transferred to another organization, the sticky policies are transferred also. In difference to our work Karjoth et al. attach policies to data. Furthermore their focus is data collected via Web forms, while we consider data transferred at service calls. Another mechanism that attaches privacy-related information to data is the Platform for Privacy Preferences (P3P) [16]. However, the P3P formalism is restricted to policies and allows only the use of few pre-defined categories to describe the kind of data and purpose of its usage.

7 Conclusion and Further Work

This paper presented a logging mechanism for distributed workflows - called sticky logging. As part of the sticky logging mechanism we defined a well-defined, semantic formalism to specify logs. Besides the formalism we specified logging actions trigged by various events during the processing of personal data. In addition, we have described how the logs are shared and accessed by the person concerned. All together the sticky logging mechanism fulfills the requirements as demanded by privacy law and contracts. In addition, the sticky logging mechanism is able to overcome the above-mentioned organizational issues.

The next step of our work is the formal definition of the logging rules. After this we will analyze the integration of the sticky logging in an existing middleware platform. In addition, we are going to extend the sticky logging mechanism to achieve accountability. Therefore mechanisms for tamper resistance, avoidance of fraud, etc. shall be integrated. The complete sticky logging mechanism will be published by means of a technical report, which is under work. In parallel we are working on a prototype that implements the introduced sticky logging mechanism. Once the prototype is finished it will be made available for download.

8 Acknowledgements

This work has been supported by the Federal Ministry for Education and Research in the project SOAinVO - ”Technique Analysis and Risk Management for Service-oriented Architectures in Virtual Organisations” and by the european projects Knowledge Sharing and Reuse across Media (X-Media, FP6-26978) funded by the Information Society Technologies (IST) 6th Framework Programme.

References

1. Baker, M. A. and Boakes, R. J.: Semantic Logging Using the Resource Description Framework, presented in the ”Work-in-Progress Novel Grid Technolgies” track of CCGrid 2005, (2005)
2. Barth, A., Mitchell, J., Datta, A. and Sundaram, S.: Privacy and Utility in Business Processes. Proc 2007 Computer Security Foundations Symposium. IEEE. (2007)
3. Carroll, J. J.: Signing RDF Graphs. In: D. Fensel et al. (Eds.): The SemanticWeb - ISWC 2003, LNCS 2870, pp. 369-384, Springer, Berlin, Heildelberg (2003)
4. Cederquist, J.G. Conn, R. Dekker, M.A.C. Etalle, S. den Hartog, J.I.: An audit logic for accountability, In: Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on. June 2005, pp. 34-43 (2005)
5. Davidow, W., and Malone, M.: The Virtual Corporation. HarperCollins, New York, (1992)
6. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Frystyk Nielsen, H. F., Karmarkar, A., and Lafon, Y. (Eds.): SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) - W3C Recommendation 27 April 2007. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/ (July 2007) (2007)
7. Hallam-Baker, P. M., Behlendorf, B., Extended Log File Format, http://www.w3.org/TR/WD-logfile-960323.html, (May 2007)(1996)
8. Hanson, C., Kagal, L., Sussman, G., Berners-Lee, T., Weitzner, D.: Data-Purpose Algebra: Modeling Data Usage Policies, IEEE Workshop on Policy for Distributed Systems and Networks 2007. (2007).
9. Karjoth, G., Schunter, M., and Waidner, M.: Platform for Enterprise Privacy Practices: Privacy-enabled Management of Customer Data. In 2nd Workshop on Privacy Enhancing Technologies (PET 2002). Springer, (2002)
10. Klyne, G., and Carroll, J. J. (Eds.): Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, (June 2007) (2004)
11. Leach, P., Mealling, M., and Salz, R.: RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace. Internet Engineering Task Force. http://www.ietf.org/rfc/rfc4122.txt (July 2007) (2005)
12. Lonvick, C.: RFC 3164: The BSD syslog Protocol. Internet Engineering Task Force. http://www.ietf.org/rfc/rfc3164.txt (July 2007) (2001)
13. Nagel, R., and Dove, R.: 21st Century Manufacturing Enterprise Strategy. Lehigh, Pa.: Iacocca Institute of Lehigh University (1991)
14. Ringelstein, C.; Schwagereit, F., and Pähler, D.: Opportunities and Risks for Privacy in Service-oriented Architectures, to appear at the 5th International Workshop for Technical, Economic and Legal Aspects of Business Models for Virtual Goods incorporating the 3rd International ODRL Workshop Oct 11 - 13 2007 in Koblenz, Germany. (2007)
15. Weitzner, Abelson, Berners-Lee, Feigenbaum, Hendler, Sussman, Information Accountability, MIT CSAIL Technical Report MIT-CSAIL-TR-2007 (13 June 2007)
16. Wenning, R., Schunter, M. (Eds.), The Platform for Privacy Preferences 1.1 (P3P1.1) Specification, http://www.w3.org/TR/2006/NOTE-P3P11-20061113/, (May 2007) (2006)