A Provenance-based Policy Control Framework for Cloud Services

Mufajjul Ali[1] and Luc Moreau[2]

[1] Orange Labs, London UK,
[2] University of Southampton

Abstract. In the context of software, provenance holds the key to retaining a mirror instance of the lifespan of a service, which can be replayed/reproduced from the beginning. This entails the nature of invocations that took place, how/where the data were created, modified, updated and the user's engagement with the service. With such an encyclopedia of information, it opens up a diversity of value-added features (compliance control, accountability) that can improve the usability of a service.

In this paper, we extend our previous work on the provenance-based policy language (cProvl) and model (cProv) by proposing a preliminary policy control framework. The framework provides the necessary building blocks for integrating and developing services that are able to generate and use provenance data for provenance-based compliance control, which runs on a XACML engine. We demonstrate the capability of the framework by applying it to a service case, and conduct benchmarks to determine its scalability and performance.

Keywords: Provenance, XACML, cProv, Prov, cProvl, Share, Cloud

1 Introduction

Cloud computing is built on top of many existing technologies, to support features such as the dynamic scaling, resource pooling, pay-per usage and on-demand self-services. While cloud computing adoption is gaining momentum in the industry, the compliance and accountability remain its main Achilles heel [1]. One approach to addressing this problem is through the use of provenance [2]. Provenance is a well understood area in art and digital-libraries, where lineage, pedigree and source plays a major role in understanding how things have been derived, and in determining the collection's authenticity and value [3]. Provenance helps in answering questions such as: What processes were involved in transforming the data? Did the processes conform to all necessary regulations? Where in the actual physical location within the cloud has the execution of data taken place? Answering these questions are pivotal to achieving compliance in the cloud environment.

In additional to provenance, a policy control mechanism is required to define the compliance requirements, and to be acted upon if a violation occurs. XACML [4], an industry wide standard is deployed by many organizations as standard policy-based control for their services. Organizations are looking to migrate their existing services to the cloud. Having the ability to use the existing policy control would minimize the cost of migration, reduce deployment effort, and mitigating the risk of using unproven technology. Its architecture is modular and provides scope for extensibility. However, it does not cater for provenance data.

In our previous work [5], we have defined a provenance ontology that extends the Prov model [6] for cloud-based services, and a provenance-based policy language that can be mapped to the XACML policy language. This allows us to express questions and conditions in the form of policies, and execute them using the ontology via the extended XACML engine.

The contributions of this paper are as follows: First, we propose a policy control framework that leverages on the XACML architecture and the Prov standard for industrial cloud-based applications. Secondly, the framework is integrated with a cloud-based service (a Telco's file sharing service) to support its compliance requirements. Finally, we perform benchmarks on the framework's integration with the service to evaluate its scalability and performance.

2 A Telco Service

ConfidenShare is a cloud service developed by a Telco Operator for the sharing of sensitive and non-sensitive information such as a file, meeting data and other data with users within the cloud environment. It uses Proxy re-encryption [7], a cryptographic technique that allows the sharing of all or part of user's data with one or more parties. ConfidenShare is interoperable with many existing cloud providers, and can meet varying country-specific cloud strategies. While the file sharing mechanism is secure, it does not have the necessary means of declaring constraints, capturing requirements and compliance control for them.

2.1 Service Requirements

Files are typically categorized as 'confidential', 'restricted' or 'general'.

A 'confidential' file is the most restricted and only the originator (creator of the document) is allowed to initiate the share.

A 'restricted' file, is where an originator can share with one or more recipients. Any changes or modifications can only be shared with the originator and recipients of the original document only.

A 'general' file can be shared with any users, and there are no explicit restrictions on the re-sharing. A further restriction can be added to the 'general' category to indicate if the file shared is modifiable, if it isn't it can only be shared unmodified.

Any user no longer registered with the service, all traceable files associated with that user cannot be shared, and should be removed. This is in accordance

to the "EU:Right to erasure" legislation [8]. Unless explicit permission has been given by the user to allow the retention of data they have already shared with other users.

In all cases the provenance of the documents are intact. From the service requirements, we can derive policies such as:

Policy 1 - If a file (fileA) is marked as 'confidential', only the originator is allowed to share it with another user (userB), re-sharing by userB is not allowed.

The provenance data contains information related to when the file was created, by whom, where, and other information that can be used to determine if it is in compliance with this policy or not.

Policy 2 - If a user (userA) is 'removed' from the service, any shared files (file X) by this user cannot be shared further (userB).

When a user is deleted, by law, all the data associated with the user must be deleted, this includes all the shared files. Provenance data can be used to check for the origin of a file. If the originator of the file is no longer with the system, then any derived or shared copy of the file can be identified from the provenance data and prevent further shares.

In order to fulfil these requirements, the following is necessary:

- Integration of the provenance capabilities to the 'ConfidenShare' service. The generated provenance data can be used to check for compliance breaches, which are fundamental to service level agreements.
- Declaration of requirements as policies, which are to run in a compliance control engine to determine and act upon the compliance status (this will require the generated provenance data).

2.2 Background

A number of provenance-based frameworks have been proposed [9], [10]. Kepler is a provenance framework designed to work with workflow management for collecting, and processing of provenance data. It provides three APIs: recording, query and management for handling such task, as well as algorithms for tracking and finding files. While their solution works well for workflows, it is not generic enough [9]. Karma is also a workflow-based framework [11] similar to Kepler, but does not have the additional processing algorithms and neither incorporates any support for provenance-based policy control.

Tsai, W.-T. *et al.* [12] discusses issues related to the data provenance in SOA; focusing on the security, reliability and integrity of the data. They also propose a SOA data-provenance framework [13], which is a more advanced version proposed earlier by Rajbhandari, S *et al.* [14]. This framework is based on the non-standard provenance model, and entails functionalities such as multiple data provenance classification (minimal provenance, time-based, event-based, etc.), data collection (actor-based and time-based), dynamic analysis (security

policy checking service (SPEC), integration estimation service) and others. The checking source SPEC appears to have some degree of correlation with our work. However, no information is supplied in relation to the language used, supported features, limitations, and how it operates on the provenance data.

Aldeco-Perez, R *et al.* [3] proposes a provenance-based compliance framework, based on the Open Provenance Model. The framework provides a processing view (represented as a provenance graph for a specific execution time) and usage policy definition (UPD). It uses the UPD to validate against the processing view for compliance. The framework lacks the integration with the commercial applications and policy standard such as XACML.

K.K. Muniswamy-Reddy *et al.* [15], [16] aims to address automation of provenance collection, by proposing three protocols for storing provenance for their existing cloud service. The provenance data is collected using their existing system called PASS (Provenance aware storage system) [17]. Any objects stored in the system automatically extracts the provenance data related to it, for example a system call read, write, etc. However their solution is proprietary.

In regards to policy, Cheney, J. [18] gives a formal model for security control for provenance, and Martin, A *et al.* [19] provides pertinent details of the applicability of provenance as a security control. PAPEL [20] is a provenance-based policy language which attempts to integrate with XACML with limited expressibility on the provenance data.

C.Dai *et al.* [18], proposes a confidence policy compliance query evaluation, that restricts or grant based on a certain confidence level. Howver the policy language is fairly restricted.

Much of these works are complementary to our previous work [5], on the provenance-based policy language, but they lack any real mapping and integration with the commercial standard such the Prov and XACML. Our focus is on using standardised policy language and model to be used in commercial applications.

3 Policy Control Framework

It is imperative for the framework to provide ease of integration of the provenance model cProv and policy language (cProvl). In order to support the provenance-based compliance control, with the existing and new commercial cloud-based services. For this purpose, we have leveraged two industrial standards: Prov and XACML architecture, that forms the backbone of the framework's stacks (figure 1).

3.1 Client Side Stack

The client stack handles operations such as the integration and generation of provenance data, as well as the request for provenance-based compliance control. More concretely, it is structured as a six layered stack (left image of figure 1).

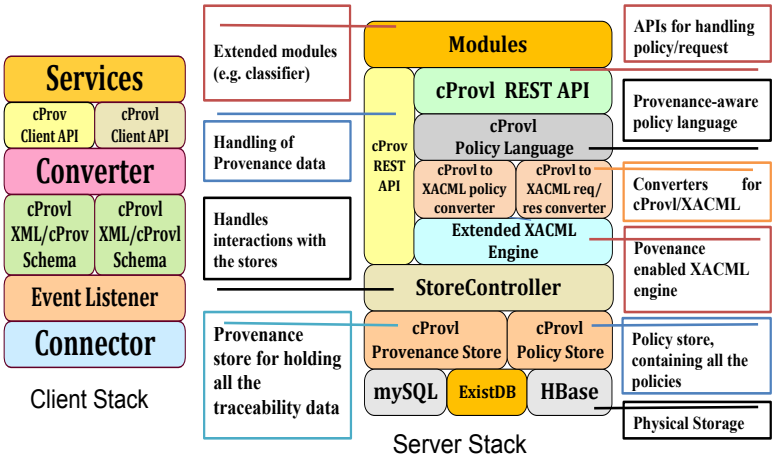

Fig. 1. Framework Stacks

Layer 1 - Defines the actual integration with a service. This is where one or more services are modified to provide provenance capability (this has been applied to the ConfidenShare service (section 2)).

Layer 2 - Provides two APIs (provenance and policy) that assist the generation of the provenance data, and declaration of a request for compliance control.

Layer 3 - Defines a list of converters (native to XML provenance and cProvl XML policy request).

Layer 4 - Provides the underlying schemas for cProv provenance model and cProvl policy request for their XML representations.

Layer 5 - Handles the generated provenance statements via the event handler, statements are placed in a temporary queue for permanent storage.

Layer 6 - Transfers the provenance statements to permanent storage and sends the policy request to the policy controller.

3.2 Server Side Stack

The server side stack defines operations for storing, querying and updating the provenance store. For compliance control, it provides the mechanism for handling policy requests, translation and execution in the extended XACML policy engine.

It contains five layers (right image of figure 1).

Layer 1 - Builds modules for extending functionalities, such as a classifier (not discussed in this paper).

Layer 2 - Provides the server side integration. It has two core APIs (cProv REST API and cProvl REST API). One for handling the provenance data and the other for compliance control. This layer also supplies converters (cProvl to XACML, and XACML to cProvl) for interacting with the XACML engine.

Layer 3 - Provides the mechanism for interfacing with the provenance and policy store.

Layer 4 - Defines the hierarchical storage structure. It contains the provenance and policy store, which consists of one or more services.

Layer 5 - The actual underlying storage (currently the framework uses the eXist DB).

By adopting these standards (prov, XACML), the framework is likely to be more compatible with the existing software development processes, tools and infrastructure.

XACML does not have any support for provenance, we have addressed this deficiency by extending its core architecture to provide provenance support using our cProvl policy language.

3.3 Extended XACML architecture

Figure 2 shows how the five core XACML components: PEP (Policy Enforcement Point), PDP (Policy Decision Point), context handler, PAP (Policy Administration Point) and PIP (Policy Information Point) [4] were extended to support the *provenance-based compliance control*.

PAP (writes XACML policies and makes then available to PDP) module has been extended to allow the creation of cProvl policies, and provides a mapping from cProvl policies to XACML policies, as well as providing storage for these policies.

The PEP (handles the initial incoming service specific request typically from an application) module has been extended to cater for a service request to be translated into cProvl request and stored in the policy store with its provenance. The service response is treated in the same manner.

The context handler is responsible for converting a service request into an XACML request. We provide the support for a cProvl request to be translated into an XACML request. The request is then transferred to the PDP module.

The PDP module determines the outcome of a request. We have introduced new functions to accommodate the handling of provenance data (used by the translated XACML polices). Before making a decision, it may request the context handler for additional attributes via the PIP module (in our case, attribute references to provenance statements).

The PIP module has been extended to interface with the provenance store. It returns the necessary statements requested by the PDP module for decision making.

The context handler receives an XACML response from the PDP module. We have also added the support for an XACML response translated to a cProvl response (stored in the policy store), which is then sent to the PEP module. The PEP translates it to service specific response and enforces the control, i.e. Permit/Deny (detailed mapping is discussed in our previous paper [5]).

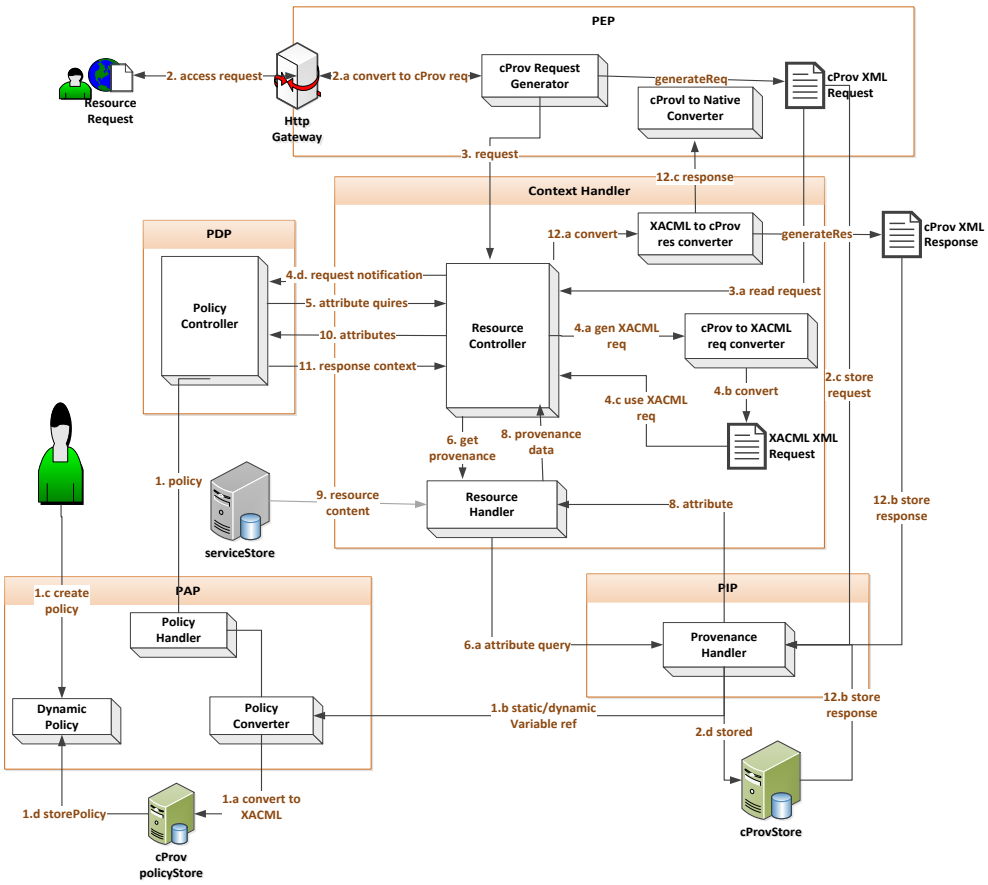

Fig. 2. XACML Extended Architecture to Support Provenance

To our knowledge, this is the only framework that enables ease of integration of the extended Prov provenance model with the XACML architecture for cloud-based services. The benefit we can see in using this framework is that only the high level APIs can be utilised without the developers requiring knowledge of the underlying complexity of extended XACML architecture or the provenance model. This can save integration and development time, is less prone to errors, and minimizes the integration complexities, which ultimately will allow developers to focus their efforts on the business logic.

4 Framework Service Integration

We have successfully integrated the framework with the ConfidenShare service (section 2). The service is able to generate provenance data, and apply provenance-based control.

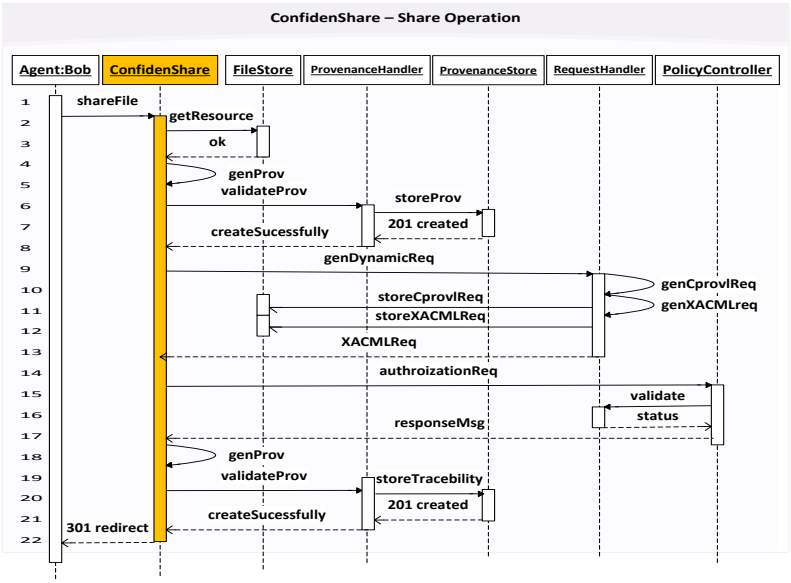

Fig. 3. Framework Integration with the ConfidenShare Service

The sequence diagram (figure 3) demonstrates the interactions between the framework's components with the service. It shows a user, Bob, invoking a resource share request on the ConfidenShare web client (line 1-3). The client (using cProv client API) generates provenance data for this invocation and interacts with the 'ProvenanceHandler' for translating it to XML Prov elements, then storing it using the cProv server API (line 4-8).

The next sequence (line 9) on the diagram is the ConfidenShare service generating and initiating a request (using the cProvl Client API) to validate against the service requirements for compliance (as defined in section 2.1). The policy controller executes the request using the defined cProvl policy (section 2.1) in the XACML engine (cProvl to XACML translation/mapping is discussed in the previous paper [5]) (line 14-16). If the response is granted, then the resource share is permitted, and the provenance record is updated (line 17-22).

An example of a dynamic request using the Client Stack (cProvl Client API) for a share request is as follows:

```
// service provenance-based control request integration
dpr.constructRequest(session.get(SESSION_USER_NAME), false, filename.getName(), false,
    null, 'a-share', true, null); //generates a cProvl request (see below)
```

This example can be read as a 'ConfidenShare' session user ('Bob') is requesting for authorization to share a file (document1). This request gets automatically translated into an cProvl request, as follows.

```
<cprovl:PolicyRequest ....> <cprovl:Agent isRef="false" prov:id="confidenshare:ag-Bob"/>
    <cprovl:Entity prov:id="confidenshare:e-document1">
```

```
            <cprovl:reqField>cprovd:Resource</cprovl:reqField>
            <cprovl:fieldValue isRef="false">confidenshare:e-document1 </cprovl:fieldValue>
        </cprovl:Entity> ...
    </cprovl:PolicyRequest>
```

An XACML equivalent of this request is as follows.

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17  ... CombinedDecision="false">
   <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
      <Attribute ... AttributeId="urn:oasis:names:tc:xacml:3.0:subject-id">
         <AttributeValue DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
         XPathCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">ex:ag-Bob
         </AttributeValue> </Attribute> ...
   </Attributes>...
 </Request>
```

This request is used by the extended XACML engine to determine if it is compliant with the defined policies (section 2.1) using the 'ConfidenShare' service's provenance data.

By making use of the APIs, major alterations to the service and business logic were avoided when integrating the framework with the ConfidenShare service. This will ultimately increase the level of trust using provenance-based compliance control in order to empower the user to verify the compliance of SLAs of cloud-based services.

5 Evaluation of Performance

Following is an evaluation of our integration of the framework with the ConfidensShare service in terms of performance and scalability. Our interest is in the provenance model, compliance control engine and policy statements. The machine used is an Intel (R) Core (TM) i7-2820QM CPU @2.30 GHZ, with 6Gb of RAM and 600Gb of disk space.

Hypothesis 1 (Service Statements) *The integration of the cProv provenance model with the 'ConfidenShare' service generates and stores provenance data at a relatively constant time in relation to the running of the service.*

Method We generate and store the provenance statements using the cProv client API, and cProv REST API. Policy one requires a minimum of 10 statements to execute, while 20 statements for policy two. This process is repeated 1000 times and added to the existing provenance graph. This produces two graphs of 10,000 and 20,000 statements. The time it takes between the creation and storage of statements are recorded as a unit of 10 statements in the first graph and 20 in the second (resulting in 1000 measurements).

Analysis Figure 4 shows a good correlation between the provenance entries (generation & insertion) and the time. For every statement, on average, it required 34.371ms. On average per unit it took 314ms in graph one and 746ms in graph two. This indicates the provenance store performance for both policies are linear and in theory the store is scalable. The 34.371ms overhead for provenance integration is favorable for the 'ConfidenShare' service.

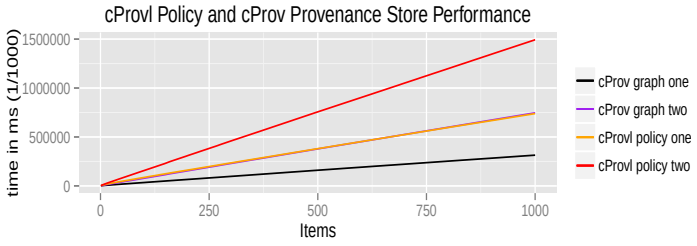

Fig. 4. Policy and Provenance Store Result

Hypothesis 2 (Compliance Control) *The cProvl policies related to the 'ConfidenShare' runs in a XACML engine to support compliance control. It is likely to add some overhead costs relative to the number of provenance statements that are required in the policy execution.*

Method We use the static cProvl policy one and two of the ConfidenShare service (section 2). The requests for policies are generated dynamically using cProvl client API, which are then translated into an XACML equivalent and executed in a extended XACML engine. The engine uses the provenance data obtained based on the previous method to evaluate each policy. This process is repeated 100 times and the start/finish times are recorded.

Analysis From figure 4, we can also see policy one's execution took on average time of 731.99ms (386.37ms without prov generation/storage time) per execution and for policy two it took 1265.22ms (518.77 without cProv). The addition of the provenance compliance control almost doubles the overhead cost. This may be due to the complex architecture (see fig 2), however, the performance is still relatively good.

Hypothesis 3 (Policy Statements) *The number of statements within a policy determines the execution time. Target statements are likely to take less time to execute compared to the conditional statements, but both should have a relatively constant execution time.*

Method Policy one(section 2) contains four targets and three conditional statements (see our previous paper [5] for further explanation). *A new policy statement (resource related) is added incrementally* to the existing policy per execution. This process is repeated 100 times, first with conditional statements, and then with target statements. The time it takes to execute a policy, from the request to the response and excluding the policy update time, is recorded. A total of 200 measurements (100 target statements and 100 conditional statements).
Analysis - As it can be seen from figure 5, with each addition of a policy statement, there is proportional increase in the time it takes to execute the policy,

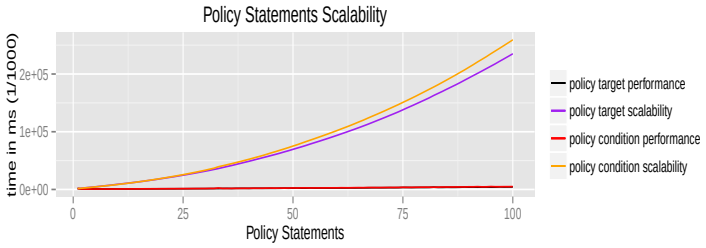

Fig. 5. Policy Statements Scalability Result

which is linear. We can see the condition statements take longer to execute than the target statements. This is as expected because conditional statements are multi-valued and contain dynamic variable references, whereas targets are typically single valued statements.

6 Conclusion

In this paper, we have presented a provenance-aware policy control framework that provides client and server stacks for integrating provenance model and provenance-based compliance control seamlessly.

We have successfully integrated the framework with the 'ConfidenShare' service, and have been able to run few benchmarks. The results show a good linear relationship between the generation and storage of provenance statements with an average of 34.3ms per statement. The integration of policy language based on the policies adds between 1 to 1.5s. Both, in theory, are scalable. In regards to policy statements, with each additional statement, the execution time increases by around 30ms.

We can conclude from the benchmark results, the integration of the framework with the 'ConfidenShare', can add between 1-2 seconds to support compliance based control, which is reasonable and encouraging. However, for a commercial deployment, we would need to take into account the network lag, bandwidth, distribution of the service components, and other factors to get a true value of the overhead cost of adopting provenance based policy control.

Acknowledgments. The first author would like to thank Rafel Uddin, Kashif Chawdhry, Tansir Ahmed and other members of Orange Labs for the on going support of the work.

References

1. Pearson, S.: Toward accountability in the cloud. Internet Computing, IEEE **15** (2011) 64–69

2. Weitzner, D.J., Abelson, H., Berners-Lee, T., Feigenbaum, J., Hendler, J., Sussman, G.J.: Information accountability. Communications of the ACM **51** (2008) 82–87
3. Aldeco-Perez, R., Moreau, L.: Information accountability supported by a provenance-based compliance framework. (2009)
4. Rissanen, E.: extensible access control markup language (xacml) version 3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf` (2010)
5. Ali, M., Moreau, L.: A provenance-aware policy language (cprovl) and a data traceability model (cprov) for the cloud. In: Cloud and Green Computing (CGC), 2013 Third International Conference on. (2013) 479–486
6. Moreau, L., Missier, P., et al.: Prov-dm: The prov data model. W3c recommendation 30 april 2013, W3C (2013)
7. Ateniese, G., Benson, K., Hohenberger, S.: Key-private proxy re-encryption. In: Topics in Cryptology–CT-RSA 2009. Springer (2009) 279–294
8. Ambrose, M.L., Ausloos, J.: The right to be forgotten across the pond. Journal of Information Policy **3** (2013)
9. Mouallem, P., Barreto, R., Klasky, S., Podhorszki, N., Vouk, M.: Tracking files in the kepler provenance framework. In: Scientific and Statistical Database Management, Springer (2009) 273–282
10. Simmhan, Y.L., Plale, B., Gannon, D., Marru, S.: Performance evaluation of the karma provenance framework for scientific workflows. In: Provenance and Annotation of Data. Springer (2006) 222–236
11. Simmhan, Y., Plale, B., Gannon, D.: A framework for collecting provenance in data-centric scientific workflows. In: Web Services, 2006. ICWS '06. International Conference on. (2006) 427–436
12. Tsai, W., Wei, X., Chen, Y., Paul, R., Chung, J.Y., Zhang, D.: Data provenance in soa: security, reliability, and integrity. Service Oriented Computing and Applications **1** (2007) 223–247
13. Tsai, W.T., Wei, X., Zhang, D., Paul, R., Chen, Y., Chung, J.Y.: A new soa data-provenance framework. In: Autonomous Decentralized Systems, 2007. ISADS '07. Eighth International Symposium on. (2007) 105–112
14. Rajbhandari, S., Walker, D.: Incorporating provenance in service oriented architecture. In: Next Generation Web Services Practices, 2006. NWeSP 2006. International Conference on. (2006) 33–40
15. Muniswamy-Reddy, K.K., Macko, P., Seltzer, M.: Making a cloud provenance-aware. In: First Workshop on on Theory and Practice of Provenance. TAPP'09, Berkeley, CA, USA, USENIX Association (2009) 12:1–12:10
16. Muniswamy-Reddy, K.K., Macko, P., Seltzer, M.: Provenance for the cloud. In: Proceedings of the 8th USENIX Conference on File and Storage Technologies. FAST'10, Berkeley, CA, USA, USENIX Association (2010) 15–14
17. Seltzer, M., Muniswamy-Reddy, K., Holland, D., Braun, U., Ledlie, J.: Provenance-aware storage systems. In: Proceedings of the USENIX Annual Technical Conference (USENIX06). (2006)
18. Cheney, J.: A formal framework for provenance security. In: Computer Security Foundations Symposium (CSF), 2011 IEEE 24th. (2011) 281–293
19. Martin, A., Lyle, J., Namilkuo, C.: Provenance as a security control. TaPP. USENIX (2012)
20. Ringelstein, C., Staab, S.: Papel: a language and model for provenance-aware policy definition and execution. Business Process Management (2010) 195–210