AAAI Technical Report SS-12-04
Intelligent Web Services Meet Social Computing

Using Web Services and Policies within a Social Platform to Support Collaborative Research

Edoardo Pignotti and **Peter Edwards**
{e.pignotti, p.edwards}@abdn.ac.uk
Computing Science
University of Aberdeen
Aberdeen, AB24 5UE

Abstract

In this paper we present an architecture for provenance policies which can be used to describe and enact behavioural constraints in a system in order to ensure compliance with user and organisational policies. We discuss how this architecture has been used in order to manage the behaviour of the services powering an existing virtual research environment while reasoning about the relationships between users, their social network, their roles in a project, their groups and the provenance of research data.

Introduction

The Web has drastically changed the way in which information is exchanged between individuals, organizations and service providers. However, these exchanges are often predicated on little or no information about the context surrounding users, their social network, their role in an organisation etc. Web services are commonly used as open components that supports the creation of distributed applications (Peltz 2003).

Heinis and Alonso (2008) have argued that capturing the provenance of information is crucial in understanding its context. Provenance (also referred to as lineage or heritage) aims to provide additional documentation about the processes that led to the creation of a resource (Groth et al. 2006). Goble (2002) expands on the Zachman Framework (Zachman 1987) by presenting the '7 W's of Provenance': Who, What, Where, Why, When, Which, & (W)How. Each of these provides a unique type of provenance information which can be used individually or in combination with others to support the assessment of trustworthiness and quality of data.

While provenance provides useful context, there is a need for intelligent services able to make use of this information to support applications. Policies provide a possible means for specifying the behavioural constraint on a service. In security, for example, policies have been used to control the behaviour of a system by controlling access to resources (Sloman and Lupu 1999). In the multi-agent systems community, policies have been used to regulate the behaviour of agents representing a variety of interests in electronic institutions (García-Camino et al. 2009).

To illustrate the issues highlighted above we now present an example drawn from our experience with a web-based virtual research environment (Reid et al. 2010). *ourSpaces*[1] has been developed to facilitate collaboration and interaction between researchers by enabling users to track the provenance of their digital artefacts and processes, and to capture the provenance of social networks, e.g. activities within the environment, relationships between members, and membership of projects and groups. Within this environment, there is a need to manage users and their behaviours so they comply with certain user or organisational policies. For example, a Principal Investigator of a project may stipulate that all digital artefacts must be associated with a geographical location. As a result, any user uploading an artefact associated with that project will be obliged to provide an associated location. Alternatively, a user may impose certain access constraints on digital artefacts that they own, e.g. an artefact may only be accessible to users who are members of a particular project and who contributed towards the artefact itself (i.e. were named as a co-author).

In the remainder of this paper we present an architecture for provenance policies which can be used to describe and enact behavioural constraints associated with different entities in a collaborative (social) platform in order to ensure compliance with user and organisational policies. We continue by discussing how this architecture has been used in order to manage the behaviour of the services powering the *ourSpaces* virtual research environment (VRE) while reasoning about the relationships between users, their social network, their roles in a project, their groups and the provenance of research data.

An Architecture for Provenance Policies

We have developed a provenance policy architecture as illustrated in Figure 1. This architecture is composed of three main components: an ontological framework capturing the provenance of agents, artefacts, processes, social networks, system events and policies; a number of associated data services and a policy reasoner.

[1] http://www.ourspaces.net

Ontological Framework & Repositories

Provenance in an important aspect of our framework as it provides the evidence for the assessment of information quality and trust. To date, the majority of work on provenance (Bose and Frew 2005) has been limited (in scope) by the application or domain in which the solution has been applied. The Open Provenance Model (Moreau et al. 2010) has made important progress towards the creation of a platform for interchange of provenance between systems. OPM defines an agnostic model of provenance based on three primary entities: *Artifact*, *Process* and *Agent* and five causal relationships: *used*, *wasGeneratedBy*, *wasTriggeredBy*, *wasDerivedFrom* and *wasControlledBy*. We have created a provenance framework based on an existing OWL representation of the Open Provenance Model.[2] This framework (illustrated in Figure 2) supports the creation of a provenance graph by defining the primary entities of OPM as well as the causal relationships that link them. OPM is a generic solution and as a result, our framework supports additional domain-specific provenance ontologies that are created by extending the concepts defined in the OPM ontology. For example, in a news ontology, one might have *article* and *comment* as types of *Artifact* and *editing* as a type of *Process*. Moreover, we define the concept of an *Event* capturing the provenance events occurring in a system (an extract of this ontology is also presented in Figure 2). The top-level concepts defined in this ontology are *CommunicationAction*, *ResourceAction*, *QueryAction* and *StateChangeAction*. Each type of event process is linked to related artefacts by the provenance graph which defines edges such as *Used* or *WasGeneratedBy*.

In a collaborative social platform, provenance should represent information regarding artefacts and processes, but must be able to situate these alongside people and their associated organisational structures. Friend-of-a-Friend[3] (FOAF) is an established RDF vocabulary for describing people and their social networks and we have opted to utilise this within our framework; a *foaf:Profile* has been defined as a sub-class of the OPM entity *Agent*. Organisational structures such as projects or employer institutions can also be defined, and users may belong to several projects or groups. We argue that it is important to capture all actions that occur within a social network and to situate these alongside artefacts and processes within the provenance graph.

Online communication is also a crucial aspect of the provenance representation of a collaborative social platform. However, the current OPM specifications support limited information about the relationship between person (*opm:Agent*) and the processes (opm:Process); as there is no regard for relationships associated with the wider social context. As a result, we have integrated the social networking vocabulary SIOC - with our provenance framework. The SIOC [4] (Semantically-Interlinked Online Communities) ontology is designed to enable the integration of online community information by providing a model to express user-generated content such as posting a message in a blog or posting a comment.

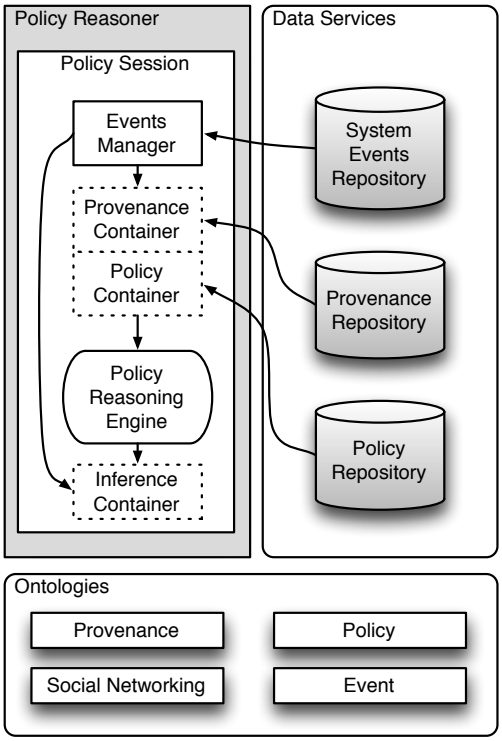

Figure 1: The service architecture behind the policy framework.

As mentioned earlier there is a need for a framework to support reasoning about policies that utilise provenance information. We define such policies as a combination of conditions such as obligations, prohibitions or permissions. Each of these conditions is based on information derived from the provenance record (e.g. an article is obliged to have at least one author). We have combined the existing OWL binding of the Open Provenance Model with an OWL ontology (inspired by the work of (Sensoy et al. 2010)) defining the concepts introduced above. An extract of the provenance policy ontology is shown in Figure 2 (Policy). Moreover, we make use of the SPIN ontology[5] to support the use of SPARQL to specify rules and logical constraints necessary to reason about policies. The SPIN ontology allows SPARQL queries to be represented in RDF and associated to classes in an ontology using two pre-defined description properties: *spin:constraint* can be used to define conditions that all members of a class must fulfill; *spin:rule* can be used to specify inference rules using SPARQL CONSTRUCT, DELETE and INSERT statements.

In our ontology a policy is a combination of *PolicyCondition* instances described by the property *hasCondition**. Each condition can be defined as an *Obligation*, *Prohibition* or *Permission* depending on the nature of the policy.

[2] http://openprovenance.org/model/opmo

[3] http://www.foaf-project.org/

[4] http://sioc-project.org/

[5] http://spinrdf.org/spin.html

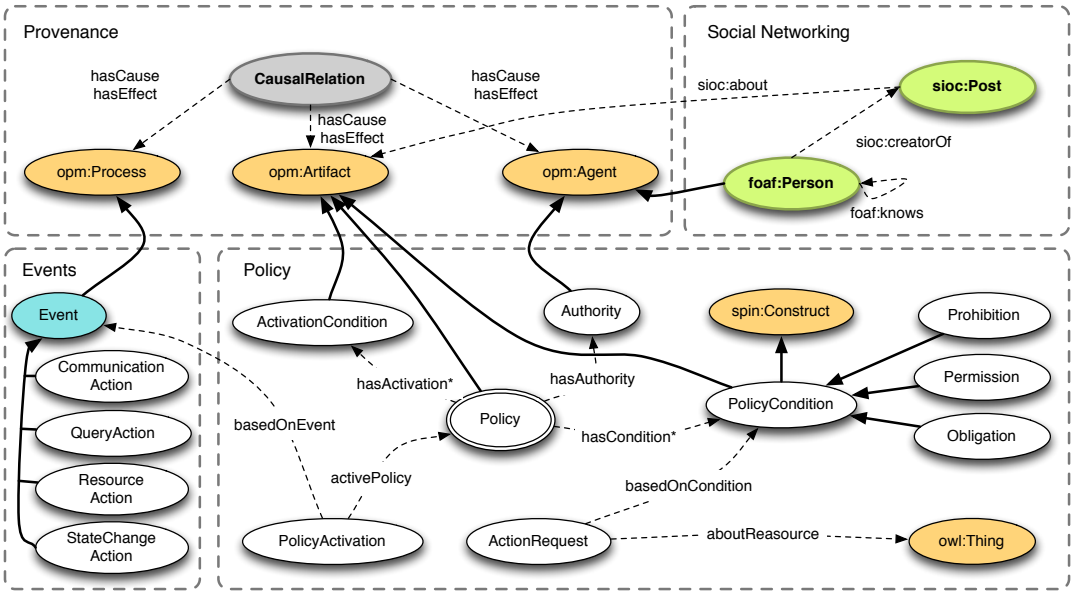

Figure 2: Ontological framework.

We define a condition as a *spin:Construct* query describing its logic in the form of a rule. Once processed by the SPIN reasoner a *spin:Construct* can assert a new *ActionRequest* instance which is constructed as part of the query.

A policy in our ontology has also one or more *ActivationCondition* instances describing the activation condition of the policy via a *spin:Construct* query. As a result of an activation the *spin:Construct* query asserts a new *PolicyActivation* instance. A *PolicyActivation* links a specific policy instance to the event that activated the policy, e.g. *UploadFile*. Finally, a policy has one or more authorities defined by an *opm:Agent* which is the person or organisation responsible for the policy.

Provenance, policies and events are stored in separate repositories in the form of RDF triples. Our policy architecture is compatible with repositories deployed using the Sesame 2.6.2 server.[6]

Policy Reasoner

We have developed a policy reasoning component based on the ontology described above as illustrated in Figure 1 (Policy Reasoner). This component uses a combination of semantic technologies such as the Jena API 2.6.4[7] and the TopBraid SPIN API 1.2.0[8]. The SPIN API performs reasoning based on the notion of a *spin:constraint* or *spin:rule* and supports the creation of user-defined functions which are essential to the implementation of actions in our framework.

A system using the policy reasoner can create instances of a policy session. A policy session is composed of three RDF containers implemented using individual Jena modes: a *provenance container* - directly linked to the provenance repository; a *policy container* - where the required policies can be loaded from the associated policy repository; and an *inference container* which contains the outcome of the policy reasoning process.

In our framework, a policy activation rule is based on user events recorded by the system. The events and their links are used by the *policy reasoner* to gather necessary context while processing a rule. For example, when uploading a resource into a system, the uploaded resource would be linked to the process using the *opm:wasGeneratedBy* edge. When a policy session is created and the relevant policies are loaded into the container, the *event manager* listen to events taking place in the system by continuously checking the system event repository. The *event manager* checks if policies stored in the *policy container* can be activated by running the SPIN reasoner against the *spin:rule* instances associated with the policies and stores the outcome of the activation in the *inference container*. An example of an activation *spin:rule* is presented in Figure 3 (details are discussed later).

Once a policy has been activated, the *event manager* invokes the *policy reasoning engine* to reason about obligations, permissions or prohibitions associated with the policy. The *policy reasoning engine* is able to check conditions over a provenance graph by processing the RDF in the *provenance container* against the rules defined by the policies in the *policy container*. For an obligation, conditions have to be met; for a prohibition, the condition can not be met; and for a permission, the condition might (or might not) be met. Figure 4 demonstrates how a *spin:Construct* can be used to describe an obligation condition. The outcome of policy conditions (e.g *ActionRequest* instances) are also stored in the *inference container* and action requests can be subse-

[6]http://www.openrdf.org/
[7]http://incubator.apache.org/jena/
[8]http://topbraid.org/spin/api/

quently handled by the system by using the *policy reasoner API*. The policy reasoner API is developed in Java and contains classes and methods to creante and manage policy sessions. For example the *PolicySession* class contains methods that can be used to extract *ActionRequest* from the *inference container* which can be used to invoke relevant functions in the host system (e.g. prompt the user for information).

Policies in a Social Research Platform

To illustrate the use and benefits of the provenance policy architecture we now present our experience drawn from development of a Semantic Web Services architecture for the *ourSpaces* virtual research environment. *ourSpaces* has been developed to facilitate collaboration between researchers by enabling users to track the provenance of digital artefacts and research processes, and to capture a record of events within their social network. *ourSpaces* makes use of Linked Data principles (Bizer, Heath, and Berners-Lee 2009) to expose, share and connect information. As mentioned earlier, within this environment, there is a need to manage services and their behaviours in order to comply with user and organisational policies. Consider the following simple examples:

- Only the members of a project are permitted to see certain information about a research artefact;
- An artefact is only accessible to users who are both members of a particular project and who contributed towards creation of the artefact;
- A project PI stipulates that a geographical location must be associated with all digital artefacts uploaded into the system.

The provenance architecture described in this paper has been used within *ourSpaces* to reason about the user's social network and the wider provenance graph to control the behaviour of the web services powering the system. Such policies are able to reason about relationships between members, roles within projects and groups, the creation history of artefacts, etc.

```
CONSTRUCT {
  _:b0 a pol:PolicyActivation .
  _:b0 pol:activePolicy :UKDADocPolicy .
  _:b0 pol:onDate ?date .
  _:b0 pol:basedOnEvent ?up .
}
WHERE {
  LET (?date := afn:now()) .
  LET (?project := <http://.../project.owl#E.coli0157> ) .
  ?up a vre:UploadResource .
  ?edge opm:cause ?up .
  ?edge opm:effect ?resource .
  ?resource pggen:producedInProject ?project.
}
```

Figure 3: An example activation condition for the UKDA documentation policy.

```
CONSTRUCT {
  _:b0 a pol:InformationRequest .
  _:b0 pol:onDate ?date .
  _:b0 pol:requestAboutResource ?this .
  _:b0 pol:requireProperty pggen:dateOfPublication .
}
WHERE {
  LET (?date := afn:now()) .
  ?this a pggen:Paper .
  ?policy pol:basedOnEvent ?up .
  ?policy pol:activePolicy :UKDADocPolicy .
  ?up a vre:UploadResource .
  ?edge opm:cause ?up .
  ?edge opm:effect ?this .
  NOT EXISTS {
    ?this pggen:dateOfPublication ?x .
  } .
}
```

Figure 4: An example obligation rule for the UKDA documentation policy.

We now discuss a scenario based on the relationship between a research project (E. coli 0157 risk) present in *ourSpaces* and the UK social science data archive (UKDA)[9]. In this scenario there is a need to enforce policies which define the kinds of metadata required for artefacts that will eventually be archived to the UKDA at the end of the project. More specifically, policies are created by the PI of the project and are addressed to its members for example one such policy may stipulate that the title, author and date of publication are required when describing a paper.

This policy is activated when a person uploads an artefact as part of the E. coli 0157 risk project that will ultimately archive its outputs to the UKDA. Figure 5 illustrates the execution of the policy reasoner. In this example, a user (Steve), has opened an upload resource popup window (shown in Figure 5 right) in order to upload a new paper for the project. The upload form is initially generated by a Web service based on a fixed template containing fields such as file, and resource type. File represent the file path of the resource to be uploaded in the VRE and resource type represent the ontology class used to describe the resource. At the same time, the *event provenance service* is invoked to assert the provenance of this event i.e. the user Steve (*foaf:Profile*) is controlling the process vre:uploadEvent (*opm:Process*). As a result of this, the *form rendering service* is invoked via an AJAX call from the form which analyses the ontology describing the concept *Paper* in order to determine mandatory fields based on the OWL constraints defined in the ontology. The resulting fields are then injected dynamically into the form (e.g. *Title* and *Has author*). Once the user confirms the title of the paper, the *event provenance service* is invoked to extend the provenance of the event by asserting that a pa-

[9] http://www.data-archive.ac.uk/

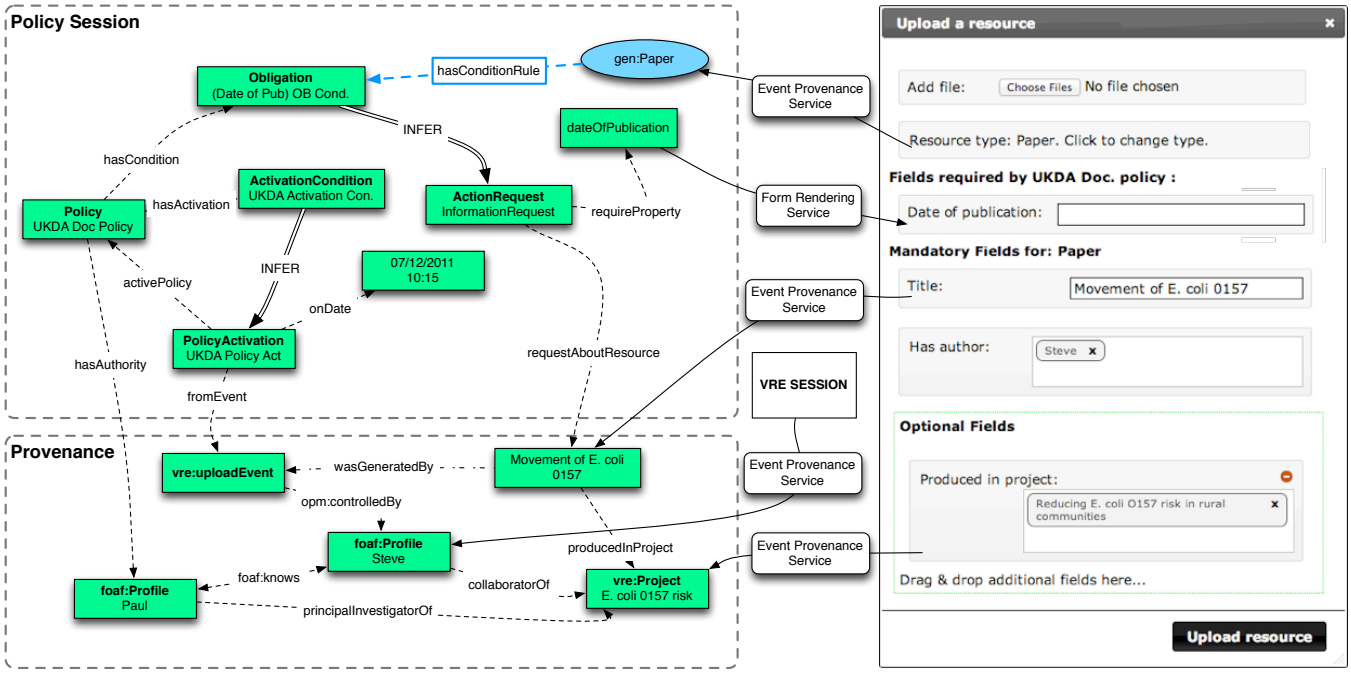

Figure 5: An example execution of the policy reasoner triggered by a user uploading a new paper into the *ourSpaces* VRE.

per titled "Movement of E. coli 0157" was generated by the *vre:uploadEvent* process (see Figure 5 - Provenance). Similarly when the user specifies the optional field *Produced in project*, the provenance of the event is updated by the *event provenance service*. However, in this case as the project selected is E. coli 0157 risk, the *event manager* triggers a policy activation based on the condition defined by the rule in Figure 3. The rule constructs a new instance of *PolicyActivation* (see *UKDA Policy Act* in Figure 5) whenever a user is uploading a paper as part of the E. coli 0157 project. The *PolicyActivation* instance contains links to the event that activated the policy (e.g. a vre:*UploadEvent*), the policy itself (*UKDA Doc Policy*), and the date and time the policy was activated.

As a result of the policy activation, the *policy reasoner* invokes the *policy reasoning engine* in order to determine if any of the conditions in the current *policy container* are met. In this example, the obligation condition defined in Figure 4 is violated as the date of publication of a paper is not specified (NOT EXISTS). As a result an *InformationRequestAction* instance is generated by the *policy reasoner* requiring the property *dateOfPublication*. The outcome of policy conditions are also stored in the *policy session* as illustrated in Figure 5 (Policy Session). The information request action is handled by the *form rendering service* which consequently inserts the *Date of publication* field into the upload form.

In this example we have shown how policies can be used to change the behaviour of services based on the specific requirements of a project. This was achieved by developing the services supporting the VRE in such a way that their behaviour could be influenced by the outcome of the policy reasoning. It is also important to note that in *ourSpaces* social networking plays a vital role in supporting the policy reasoning process. First of all, it is possible to tailor the activation of a policy to a specific user context, e.g. a user being directly (or indirectly) involved in a group or a project. Moreover, the user is more likely to accept an unanticipated behaviour of the system as the authority behind a policy is represented by a *foaf:Person* and that individual may belong to their existing social network (see Figure 5 Provenance).

Conclusions

In this paper we have presented an architecture designed to reason about provenance policies in such a way that the behaviour of services controlling a system could be influenced by the outcome of the policy reasoner. We have discussed the deployment of this architecture to the *ourSpaces* VRE where we developed the services controlling the system in such a way that their behaviour could be influenced by the outcome of the policy reasoner. We also discussed the implication of social networking in providing the context for the activation of a policy and additional evidence so the user is more likely to accept an unanticipated behaviour of a service.

Policies and particularly the SPIN reasoner have a certain cost in terms of required computational resources. Some policies do not require much data to be evaluated. Others, such as uploading an artefact, might require large provenance graph to be evaluated. We have performed a preliminary experiment to find out the time required to run policies, using the *ourSpaces* provenance repository. The hardware used for the test was a Sun Fire X4100 M2 with two dual-core AMD Opertron 2218 processors and 32 GB of memory.

The dataset is a snapshot of the *ourSpaces* metadata repository consisting of 8760 RDF triples describing the provenance of 624 digital artefacts uploaded by 171 users into the environment. Based on 50 runs of the reasoner the average time to run a policy was 19ms. For 10 policies, the average time was 62ms and for 100 policies it was 460ms. The data shows that for one policy an average 4.2ms is required by the reasoner to process it. (Miller 1968) and (Card, Robertson, and Mackinlay 1991) argue that system response times of less than a second do not interrupt the user activities. Based on our initial results from this experiment we determined that no more than 100 policies can be evaluated by the reasoner for all users at one given time without compromising the overall performance of the system.

Policy reasoning also presents a cost in terms of data storage. The inferences generated by it could be stored in the RDF repository in order to provide additional provenance about user actions. However, it is not always necessary to do so, as some actions are not very relevant in terms of provenance, e.g. knowing that during an upload a user forgot to complete a required field. On the other hand, knowing that a user tried to log in to a different account twenty times in a row with an incorrect password can indicate a potential security threat and thus is something worth registering for later inspection.

We are in the process of extending the use of our policy reasoner to other services in the *ourSpaces* VRE. We have already successfully integrated policies into a natural language generation (NLG) service which allow the user to visualise a textual description of RDF resources available in the VRE. Policies in this service are used in order to tailor the content of the textual description, e.g. to remove certain information if there might be a risk of a breach of privacy.

We designed our provenance policy architecture with the ambition that it would be compatible with any collaborative (social) platform. However, to date we have only tested it within the *ourSpaces* VRE. We acknowledge that if another system were to use this architecture there are a number of requirements. The most important of which is the need to represent agents, artefacts and processes in the system based on the OPM OWL specification and being able to store such metadata using the Sesame service.

Acknowledgments

This work is supported by the UK Economic & Social Research Council (ESRC) under the Digital Social Research programme; award reference: RES-149-25-1075.

References

Bizer, C.; Heath, T.; and Berners-Lee, T. 2009. Linked data - the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*.

Bose, R., and Frew, J. 2005. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.* 37(1):1–28.

Card, S. K.; Robertson, G. G.; and Mackinlay, J. D. 1991. The information visualizer, an information workspace. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, CHI '91, 181–186. New York, NY, USA: ACM.

García-Camino, A.; Rodríguez-Aguilar, J. A.; Sierra, C.; and Vasconcelos, W. 2009. Constraint rule-based programming of norms for electronic institutions. *Autonomous Agents and Multi-Agent Systems* 18:186–217.

Goble, C. 2002. Position statement: Musings on provenance, workflow and (semantic web) annotation for bioinformatics. Workshop on Data Derivation and Provenance, Chicago.

Groth, P.; Jiang, S.; Miles, S.; Munroe, S.; Tan, V.; Tsasakou, S.; and Moreau, L. 2006. An architecture for provenance systems. Technical report, ECS, University of Southampton.

Heinis, T., and Alonso, G. 2008. Efficient Lineage Tracking for Scientific Workflows. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, 1007–1018. ACM.

Miller, R. B. 1968. Response time in man-computer conversational transactions. In *Proceedings of the joint computer conference 1968*, AFIPS 68, 267–277. New York, NY, USA: ACM.

Moreau, L.; Clifford, B.; Freire, J.; Futrelle, J.; Gil, Y.; Groth, P.; Kwasnikowska, N.; Miles, S.; Missier, P.; Myers, J.; Plale, B.; Simmhan, Y.; Stephan, E.; and den Bussche, J. V. 2010. The open provenance model core specification (v1.1). *Future Generation Computer Systems*.

Peltz, C. 2003. Web services orchestration and choreography. *Computer* 36:46–52.

Reid, R.; Pignotti, E.; Edwards, P.; and Laing, A. 2010. ourspaces: Linking provenance and social data in a virtual research environment. In *International World Wide Web Conference*.

Sensoy, M.; Norman, T. J.; Vasconcelos, W.; and Sycara, K. 2010. Owl-polar: Semantic policies for agent reasoning. In *International Semantic Web Conference*.

Sloman, M., and Lupu, E. 1999. Policy specification for programmable networks. In *Proceedings of the First International Working Conference on Active Networks*, 73–84. London, UK: Springer-Verlag.

Zachman, J. A. 1987. A framework for information systems architecture. *IBM Systems Journal* 26(3):276–292.