A Provenance-aware policy language (cProvl) and a data traceability model (cProv) for the Cloud

Mufajjul Ali
Orange Labs
France Telecom R&D UK Ltd
London, UK
Email: mufajjul.ali@orange.com

Luc Moreau
Electronics & Computer Science
University of Southampton
Southampton, United Kingdom
Email: l.moreau@ecs.soton.ac.uk

***Abstract*—Provenance plays a pivotal in tracing the origin of something and determining how and why something had occurred. With the emergence of the cloud and the benefits it encompasses, there has been a rapid proliferation of services being adopted by commercial and government sectors. However, trust and security concerns for such services are on an unprecedented scale. Currently, these services expose very little internal working to their customers; this can cause accountability and compliance issues especially in the event of a fault or error, customers and providers are left to point finger at each other. Provenance-based traceability provides a mean to address part of this problem by being able to capture and query events occurred in the past to understand how and why it took place. However, due to the complexity of the cloud infrastructure, the current provenance models lack the expressibility required to describe the inner-working of a cloud service. For a complete solution, a provenance-aware policy language is also required for operators and users to define policies for compliance purpose. The current policy standards do not cater for such requirement.**

To address these issues, in this paper we propose a provenance (traceability) model cProv, and a provenance-aware policy language (cProvl) to capture traceability data, and express policies for validating against the model. For implementation, we have extended the XACML3.0 architecture to support provenance, and provided a translator that converts cProvl policy and request into XACML type.

***Index Terms*—policy language; provenance; cloud; cProvl, cProv; Prov; data traceability; XACML;**

I. Introduction

Cloud computing relies on many existing tools and technologies reducing the cost of service delivery whilst increasing the speed and agility of service deployment [1]. The core technology behind cloud computing is virtualization [2], [3]; it empowers the whole cloud computing paradigm by creating an abstract layer between the physical hardware and the operating system. This allows a greater degree of flexibility by being able to share the same physical resources virtually by more than one OS. Currently, there are three well defined service models: Infrastructure as a Service (IaaS [4]), Platform as a Service (PaaS) [5], and Software as a Service (SaaS)[6].

Today, *one important piece of the jigsaw missing from cloud is accountability, and provenance is the solution*. Provenance is a well understood area in art and digital-libraries, where lineage, pedigree and source plays a major role in understanding how/where things have derived from, and in determining its authenticity and value [7]. In the cloud, provenance is fundamental in answering questions such as: What processes were involved in transforming the data? Did the processes conform to all necessary regulations? Where the execution of data did take place (both from virtual to physical references)? Who had access to these data? In order to answer such questions, one needs to look at how provenance in the cloud can be modelled, captured and queried in the context of cloud computing.

The capturing of provenance data via the model is vitally important for accountability and compliance purpose, as well as in determining the course of action(s) to be taken. The current approach to addressing these is via access control. Access control (AC) plays a pivotal role in safeguarding systems from unauthorized access and providing different levels of access granularity. There has been several language specifications developed over the years, ACL (Access Control List) [8], Role Based Access Control (RBAC)[9], Attribute based access control (ABAC) [10] [11] and more recently policy based access control (PBAC) [12].

Amongst these policy mechanisms, there are few distinct features which are in common; they all rely on current data to evaluate their policies, and *do not leverage on provenance data*. It is very difficult for these languages to achieve a good level of accountability or compliance, since the data set they use does not contain information on how a piece of data was originated, who had access to it, nature of processing took place on it, etc. This can lead to a premature decision making. The data defined within a policy generally tend to be fixed. In other words, changes in the domain are not directly reflected by the policies (require explicit modifications). However, the volatility of distributed environments such as the cloud, where environments are constantly changing, would greatly benefit from policies that can offer some level of adaptability to its context changes. These deficiencies can be addressed by a provenance-aware policy language that uses the historical data and its relations for assertion, as well as variable like feature for handling generic/dynamic data. The language should have some level of interoperability with the existing widely deployed standard in the industry, such as the XACML [13].

The contributions of this paper are, first an ontology (traceability model) for cloud-based provenance, which allows a

cloud-based service to describe and represent its provenance (cProv). Secondly, a provenance-aware policy language that allows assertions on the provenance data for accountability and compliance purpose (cProvl). Finally, a mapping of the policy language to XACML 3.0, that allows running of cProvl policies in a XACML based engine.

II. Scenario

Telco operators hold vast amount of data from its users. They range from personal data (name, address, tel, Dob, sex, etc.) to call logs, location information, interests, likes and others; kept for short (typically months) to longer periods (years). Some with their knowledge (i.e. Bank details) and consent (e.g., sharing with 3rd parties), while others without being aware of (call logs, location information, etc.) and in exceptional circumstances shared secretly (intelligent services).

With the buzz word "BIG data" in the industry, there is an increasing temptation by operators to use and process such gigantic data for serious analytic in seeking knowledge for competitive advantage, target marketing and monitory purpose. This is also attracting big interests from 3rd parties. Regulators prevent operators from sharing personal data where a user can be uniquely identified. However, anonymizing or aggregating (removing fields, obfuscating) data would make it possible.

With the increase of cloud services, data can be more freely moved around without users their knowledge or consent, and with the recent revelation of the PRIMS snooping, the trust remains a gray area for the users.

A. requirements

To increase the trust of the users of the operator, an interactive dashboard that lists all the cloud services subscribed by the users. It provides a trace of how their data were generated, used, stored, used and shared by subscribed and unsubscribed services. More importantly, users are able to write policies that can trace which other services using their data, and reserves the right to grant and revoke access. This also applied to the anonymized, aggregated data.

Examples of sample policies are:

policy one - Personal data of a user cannot be taken outside the resident country of the user by any services. Such breach, access to the data would be revoked.

- Using my traceability model, it is possible to trace the history of data to find the physical location from the virtual location of the user's personal data, and the service that copied and stored it to a location. The original and copy should have the same location footprint, otherwise a violation has occurred.

policy two - Any non-provisioned Telco services can access user's personal data, but cannot allow access or share of these data to any 3rd party services that the user is not aware of.

- The traceable data are required to check for which Telco services are using the user's personal data. Then we can identify if any of these services are exposing data to 3rd parties. This can be in the form of APIs, or direct calls (explicit or implicit). The traceable model is able to differentiate between different types of calls.

policy three - After de-provision of a service, all the associated data must be deleted completely. In such breach, any access to personal data will be denied.

- EU legislation '*right to be forgotten*' [14] require all the data associated with a user must be deleted permanently. Using our traceable model, the historical data can be used to prove the process of deletion and check of any existence of any data after deletion. Failure to comply can result in a hefty fine.

III. cProv - Provenance Model

In the cloud, data may be transmitted from various sources such as a PC, laptop, mobile and other devices. Data residing outside the cloud is referred to as a physical resource as opposed to in the cloud. In order to necessitate the transfer of the data to the cloud, it is essential to virtualize the data with necessary redundancies for optimal availability, and scalability. Data within the cloud can be shared, modified or deleted by one or more participants, services or agents. One or more operations are grouped and executed as an event.

We propose a provenance model cProv (Fig 1) to facilitate cloud services by capturing of contextual related data at the service/platform level.

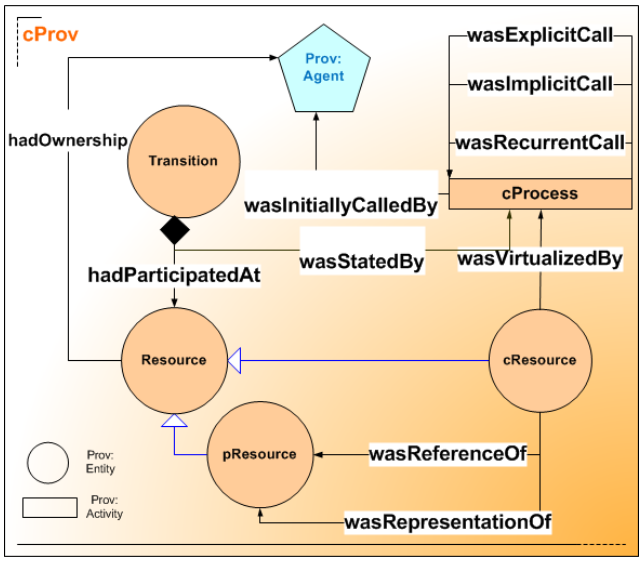

Fig. 1: cProv Model

The model provides a representation of provenance history using Prov notation [15], consisting of nodes (vertices) and relationship (edges). Node represents the building blocks of a service. There are five new derived nodes (cprov:Transition, cprov:cProcess, cprov:Resource, cprov:pResource and cprov:cResource). The ellipsis are subtypes of prov:Entity, and rectangles are subtypes of prov:Activity.

The node properties include: location details, event (*comprises of a list of operations that were executed either in a*

sequence or parallel as a unit; typically on data) information, virtual-to-physical mapping, time-to-live and others.

In reference to the scenario, using the 'entity' node, we can represent a user's data as follows:

```
entity(ex:e001, [prov:type='cprov:cResource', cprov:
  type='cprovd:call-record', cprov:trustDegree="1.0"
  %%xsd:float,cprov:userCloudRef="http://orangecloud
  /user@mufy/clusterX/imageX" %% xsd:anyURI, cprov:
  vResourceRef="/platformX/ServX/resX"%%xsd:anyURI,
  cprov:pResourceRef="//ClusX/ServNameX/6.23.3.5/
  00:12:00:11:00"%% xsd:anyURI,"true"%% xsd:boolean
  ,cprov:TTL="2014-11-16T16:05:00" %% xsd:date])
```

The above entity refers to a user's (ex:ag001) call record called 'ex:e001', and contains information such as time-to-live, virtual and physical location of the data. The user associated with this data is represented as an agent.

```
agent(ex:ag001,[prov:type="person",prov:label="Fu"])
```

To represent association between the user and data, edges are required. We refer to them as relationships.

The relationships define the nature of interactions between the nodes. They play an essential role in defining how each building block in software interacts with each other, in other words they define the flow of executions of a service. There are a total of ten relationships proposed to allow a greater degree of expressiveness of a cloud-based service (see figure cProv Model). These are sub-classes of the Prov edges (wasInformedBy, wasDerivedFrom, wasAssociatedWith, wasGeneratedBy and wasAttributedTo) [16].

We can define the relationship between the agent (Fu) and entity (call record) using the 'wasAttributedTo'edge.

```
wasAttributedTo(ex:e001, ex:ag001, [prov:type='cprov:
 hadOwnership',cprov:ownershipType='cprovd:originator'])
```

The ownershipType can be either 'originator', 'contributor' or 'possession'. This relation can be read as call record (ex:e001) was originated (in this case it would be implicit) by Fu ('ex:ag001).

Another example, assuming ex:a001 (activity) invoked another process ex:a002 (not shown here). This invocation can be an implicit explicit or a recurrent call. Knowing this information, can be used to determine if a service shared users' information automatically, or a user (agent) was involved, i.e with or without his/her consent.

```
wasInformedBy(ex:a001, ex:a002,[prov:type='cprov:
  wasImplicitCall',cprov:type ='cprovd:notification',
  cprov:callComm='cprovd:synchronized',cprov:
  callMedium="server/S-CSCF",cprov:callNetwork="3G"])
```

Above example can be read as, process (ex:a002) made an implicit notification call to process (ex:a001) from the server over the 3G network.

From a cloud prospective, we can use this model to determine if any external parties had access to user's data, physical storage location of their data, and where it was processed. Having such knowledge can implicitly increase the trust of the provider and the end user. However, given the size and complexity of the provenance data, it is relatively challenging in interpreting such information manually and to take any meaningful actions. A more automated process is required whereby policies can be defined to detect for such violations and appropriate control can be enforced.

IV. cProvl-Provenance-aware Policy Language

Cloud Provenance-aware policy definition and control language (cProvl) requires modelling of complex relationships defined in the Prov [17] and its extension (cProv) in order to facilitate policies and rules required by cloud service providers (auditing, compliance of SLAs/OLAs [18] and access control) and consumers (violations and access control). The declaration of policies and rules itself should be provenance-aware to allow ease of integration between the provenance data and the policy. This close integration should enable users to define more complex policies and rules with greater expressibility on the provenance data.

The Figure 2 below, shows a proposed policy structure with a policy language called cProvl. It consists of three layers: application, policy engine and persistence.

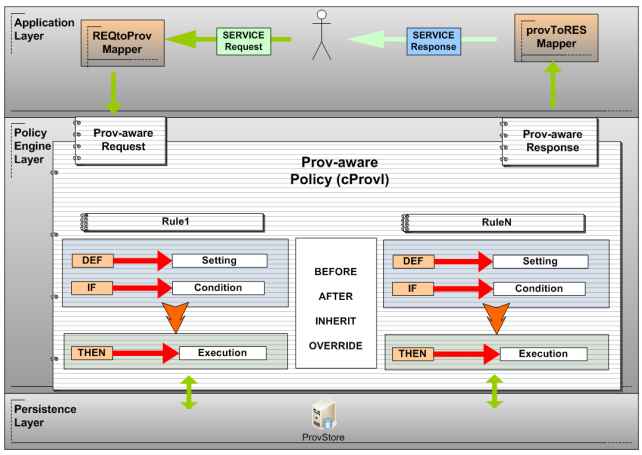

Fig. 2: Policy Structure

At the application layer a user composes a service request, this could be a form of a REST [19] request. This is forwarded to the REQtoProvMapper, that encodes the request into a Provenance-aware request. The request is then sent to the policy engine layer.

In this layer, the request is handled via the relevant policy. The policy may execute one or more rules to validate the request using the persistence layer (Prov store). The outcome of the policy creates a provenance-aware response. This response is forwarded to the provToRESmapper (provenance to request format mapping, i.e JSON), which converts it into the format the client can consume. In this paper our focus is on the policy language and policy engine.

A. Policy Syntax

The cProvl's grammar for defining the structure of a policy and a rule has been defined using key words (POLICY-BEGIN, RULE, etc.). The syntax/semantics for policy and rules are declared using the prov notations.

B. Policy Structure

In order to create a policy, a unique identification is required. It is declared using an entity statement Id (ex:policy1), followed by a description attribute cprovl:description. This is preceded by the rule declaration.

```
POLICY-BEGIN ()
    entity(ex:policyId,[cprovl:description=""])
    //declaration of a policy
    RULE
        entity(ex:ruleId,[cprovl:description=""])
        //declaration of a rule
              - -  conditions - - -
    RULE
        entity(ex:ruleId,[cProvl:description=""])
        - -  conditions - -
POLICY-END
```

A policy can have one or more rules explicitly declared using an entity, with a unique Id reference (ex:ruleId).

C. Rule Structure

The rules are designed to execute certain business logic for controlling resources. This can in the form of granting/denying access to users/processes based on fulfillment of conditions. A rule has the following structure:

```
RULE
    entity(ex:ruleId,[cprovl:description=""])

    INHERIT || BEFORE || AFTER || OVERRIDE
    //rule constraints
    DEF
        entity(ex:scope, [cprovl:range=""])}
        //scope declaration
    IF  THERE EXIST || FOR ALL] SUCH THAT
        [CONDITIONS]
    THEN
        [EXECUTION]
```

Each rule is identified by an entity that contains a unique identifier Id, followed by the optional rule operators.

1) Rule Operator: A policy consists of one or more rules, which may contain dependencies. Dependencies can be in various forms. For example, a rule may be required to be executed before another or output of one rule may be an input to another. A total of four operators have been defined to handle such functionalities: INHERIT, BEFORE, AFTER and OVERRIDE.

INHERIT is required when one rule inherits from another, when residing in a different policy. It is possible to have multiple inheritance, however, only one level of inheritance is permitted. An inheritance can be overridden by the 'cprovl:part' attribute from the OVERRIDE operator.

The BEFORE and AFTER operators are used to determine the execution ordering of rules. This is followed by the declaration of the scope.

2) Scope: The scope allows users to be selective in defining the range of data (structure) to be used by a policy. It is declared under the keyword 'DEF'. A scope is set by using the cprovl:range attribute. An example:

```
DEF
    entity(ex:scope, [cprovl:range='cprovd:all'])
```

This attribute can take a range of values based on the selection of scope (granularity) required: cprovd:all, cprovd:event, cprovd:node and cprovd:edge. The scope is followed by the target section.

3) Target: The existential quantifiers have been used in many systems [20], [21]. A target defines one or more IDs for a rule, which is matched using the existential quantifications.

- FOR ALL - match all occurrence of the criteria
- THERE EXIST - match at least one occurrence of the criteria

```
IF THERE EXIST (IDs) SUCH THAT [CONDITIONS]
IF THERE EXIST (ex:e001) SUCH THAT [CONDITIONS]
```

The ID can either be denoted as an entity, agent, activity, or a variable. The above expression can be read as, match an instance of an ex:e001 in the prov store. Please note, the ex:001 Id is a static reference, and cannot change. However, there may be cases where dynamic values are required, for example any generated files. This can be expressed by using variables.

4) Variable : Variable allows dynamic and reference values to be assigned either at the execution time or during the declaration. In order to distinguish a variable from a content Id, they are declared using the 'r' namespace.

```
    [new | s-ref | d-ref] [r]:[varName]}
```

Variables are of arbitrary data type and do not require any explicit typing. The actual data type is determined dynamically. Variables can be of two types: 'new' and 'ref'.

- **new -** keyword is used to define a *dynamic variable*. It acts as a placeholder for an outcome of an expression. Once declared, it can be used in multiple places within a rule (the *content* and *type* is determined at the runtime).

```
IF  THERE EXIST (new r:req) SUCH THAT
    (wasGeneratedBy (r:req, ex:session))
```

  This expression declares a dynamic variable called r:req, and the value is obtained from the execution of the wasGeneratedBy. During the execution, the r:req variable acts as a placeholder, essentially acting as an anonymous entity.
- **ref -** keyword expresses a *reference variable*. It is used to obtain a reference from an existing object, typically from an input request. There can of two types:
  - **s-ref** - Refers to direct one-to-one match with a request value.

– **d-ref** - Refers to the variable holding values matching of its category type, typically from a request.

```
IF  THERE EXIST (d-ref r:user) SUCH THAT
 (hadOwnership(ex:e001, r:user))
```

This expression says, check if the requested user has an ownership to entity ex:e001. The r:user is generic, and can handle any users from the request.

5) Conditional Statement & Operator : The Conditional statements are expressed using the Prov notation. Each statement can either be an entity, activity, agent or a relation. If no operator between statements an implicit logical 'and' is assumed, and == *(equal)* operator is applied.

6) Logical Operators & grouping: The logical operators are also supported. These are: && (and), ||(or), and ! (not). Grouping are sued if more than one operator is used, denoted by curly brackets { }. It is possible to have nesting groupings.

```
wasGeneratedBy  (ex:meetingRequest, ex:session)
{
&&
    {
       wasAssociatedWith(ex:session, ex:matt)
      || wasAssociatedWith(ex:session,ex:john)
    }
}
```

7) Execution: The execution phase of the rule determines the right course of action to be taken based on the satisfactory outcome of conditional statements.

The permission (outcome) can be expressed by the creation of a new entity.

```
new ex:p0 entity(ex:p0,[prov:type='cProv:cResource',
    cprovl:actionId="", cprovl:resourceId=""])
```

The actionId is a mandatory field which defines the nature of the outcome of the rule. The following values are valid outcomes:

- new cprovd:permit - grant permission
- new cprovd:deny - deny permission
- new cprovd:indeterminate - cannot determine
- new cprovd:not-applicable - irrelevant request

By default, the request defines the resource to be accessed. However, the resourceId attribute can be used to restrict to a specific resource to grant access.

V. cProvl to XACML mapping

We have chosen to design the provenance traceability model (cProv) and the policy language (cProvl) using XML schema. This ensures that both can be easily extended, and are not implementation dependent. The cProv schema contains elements and attributes declaration, simple and complex types, groups and element participles. It is designed for capturing of provenance trace for cloud services. The cProvl schema uses the cProv schema for declaring statements, and defines its own language syntax for creating provenance-aware policies, rules, request and response.

We have successfully modelled policies for the scenario (due to the size of the policies, it is not shown here). For execution of the policy language, we have chosen to leverage on the existing policy language standard XACML 3.0. The standard is widely deployed in the industry, and is relatively mature. XACML however does not have support for handling the provenance data, hence is not directly compatible with our policy language syntax and semantic. To address this issue, we have decided to create cProvl-to-XACML mapping, that would allow cProv policy and request to be executed in a XACML engine.

Our approach to creating the mapping between XACML and cProvl is to build a converter, that can take a cProv policy as an input and map its entries to XACML equivalent. Since both languages are based on XML, we have chosen to use XSLT stylesheet language. The language is simple yet powerful for manipulation XML/HTML documents.

A. Extending the XACML Architecture to Support Provenance

Figure (3), shows the extended functionalities required for the XACML engine to execute a cProvl-based policy and request. The policy engine is also referred to as Policy Decision point (PDP), which forms the heart of the Access control mechanism; execute targets and conditions using various functions. While many of these functions are reusable, there is a necessity to introduce new once to address the following challenges.

1) Coupling of policy assertions: XACML policies are by default tightly coupled with requests. The policy uses request values (current values) to grant or deny access. cProvl, on the other hand is more loosely coupled, and while it may take some values from the request, policies are primarily focused on the data from the Prov store (historical meta-data and relations) for its assertions. This requires XACML XPath functions to operate on the Prov store. However, they are restricted to 'content' XML from the request. To overcome this issue, we have introduced a new function called *urn:oasis:names:tc:xacml:3.0:function:ext:xpath-provenance-s-id-match* for target. The target IDs are handled by this function (it matches against the Prov store (see 1a, 1b and 1c on the diagram 3).

2) Static & dynamic variable holder: Provl require static and dynamic variable holders for its statements; one statement may generate reference IDs stored in a variable, which is later required/used by another statement. *Such concept is not present in XACML.* In order to address this issue, we have introduced another function called *urn:oasis:names:tc:xacml:3.0:function:ext:xpath-provenance-d-id-match*. The 'd-ref' and 'new' variable declared in the target is handled by this function. The d-ref values are extracted from the input request type (e.g. subject) and stored in the attStore. For 'new' variable it creates an entry in the store for the statement assertion (see 2a, 2b, 2c and 2d). The function urn:oasis:names:tc:xacml:3.0:function:ext:xpath-provenance-s-id-match, generates the content of a new variable, as it executes the conditional statements.

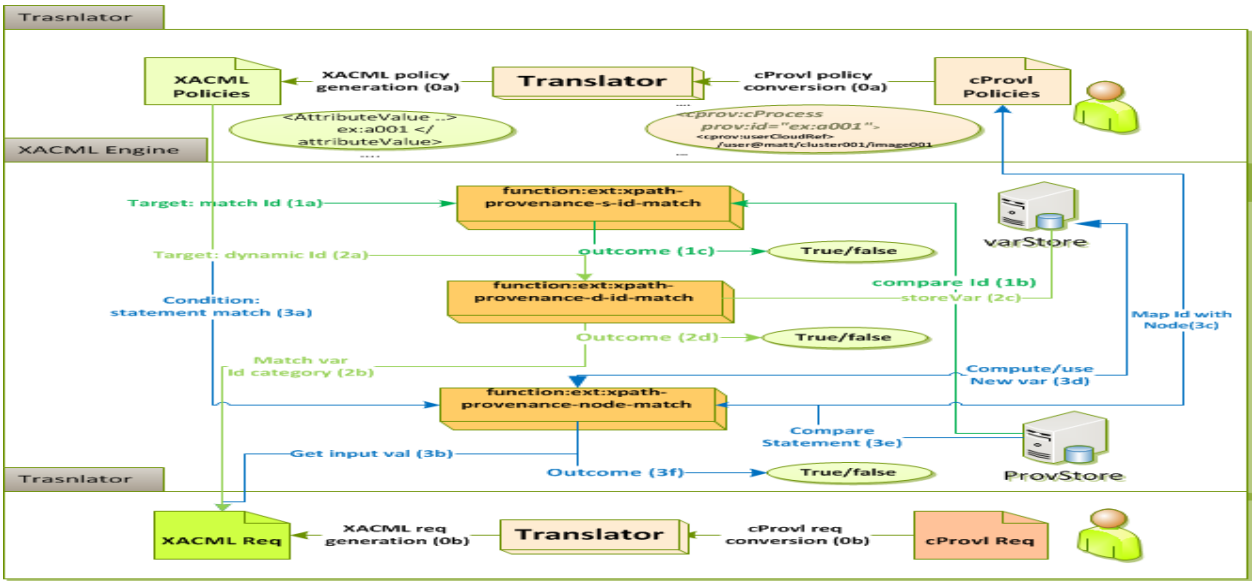

Fig. 3: XACML Policy Engine Extension

3) Single to multi-value mapping: XACML conditional statements are single value entry attributes, whereas Provl statements are multi-valued nodes/edges. In order to map single-to-multi-values, we have created a new function called urn:oasis:names:tc:xacml:3.0:function:ext:xpath-provenance-s-id-match. This function first obtains the attribute value of an XACML policy conditional statement (this value needs to be a unique ID). It uses this as a xPath reference to a node in cProvl Policy. If a match found, the node and its properties are matched against the ProvStore. If all successful, it will return true otherwise false.

B. cProvl to XACML policy example

The diagram (Fig 4) shows policy one from the scenario, modelled using the cProvl language on the left, and mapping to XACML on the right.

The policy and rule Id from cProvl are directly mapped to XACML's policyId and RuleId, denoted by the red color lines. The target Ids (ex:p-dat and c-data) which represents a user data, and copy data which are mapped to XACML using our custom function "xpath-provenance-s-id-match" (shown in green lines). This function matches both values against the provenance store. The conditional statements (can be read as allow access if the copy resides in the UK, assuming original is also in the UK) are mapped to XACML using our defined function "xpath-provenance-node-match" (shown in blue lines). This function handles the multi-valued conditional statements of cProv. The outcome of the policy is mapped to the "Effect" attribute filed of the Rule in XACML.

VI. Literature Review

Much of the earlier work that has been done for provenance is in the area of scientific workflow [22], [23], [24], and many models defined which can be mapped to the core of the OPM [25]. OPM can be seen as the common subset of all these languages. OPM however has been superseded by the W3C backed model called Prov [15]. A greater number of relationships are defined to describe interactions between entities, activities and agents. It has support for extensibility via custom attributions. The other efforts in this area are from, P. Macko *et al.* [26] on an approach for collecting provenance via the Xen Hypervisor [27], K.K. Muniswamy-Reddy *et al.* [28] [29] on the automation of provenance collection (Provenance aware storage system) [29].

As regards to the policy language, Y. Doganata *et al.* [30] proposes a model for authoring and deploying business policies dynamically for compliance monitoring. Their work is quite similar to that is being proposed here, but differs in a few ways. First, the provenance model is proprietary and specifically designed for business related applications. Whereas our proposed model open, is an extension to the standardised provenance model prov, for the dedicated cloud environment. Secondly, the language does not capture of the provenance of policy decision making, so there is a loss of provenance information. Our approach is end-to-end provenance aware.

B. Stepien *et al.* [11] on the other hand proposes a human readable form of a policy language. Unlike the previous policy templates, this is based on a well known standard XACML [31],[32]. A policy can be defined easily using natural language, which is then converted to XACML format. However, it does not cater for provenance data.

PAPEL [33] is a provenance-aware policy execution language. The language tries to integrate the popular XACML general purpose policy language with the well defined provenance model called OPM, albeit a relatively loose integration. It uses a step primitive to represent a single processing step which depicts a process in OPM or cProcess in cProv. Step only defines primitive primitive parameters. This can restrict the expressibility and extensibility required for modelling

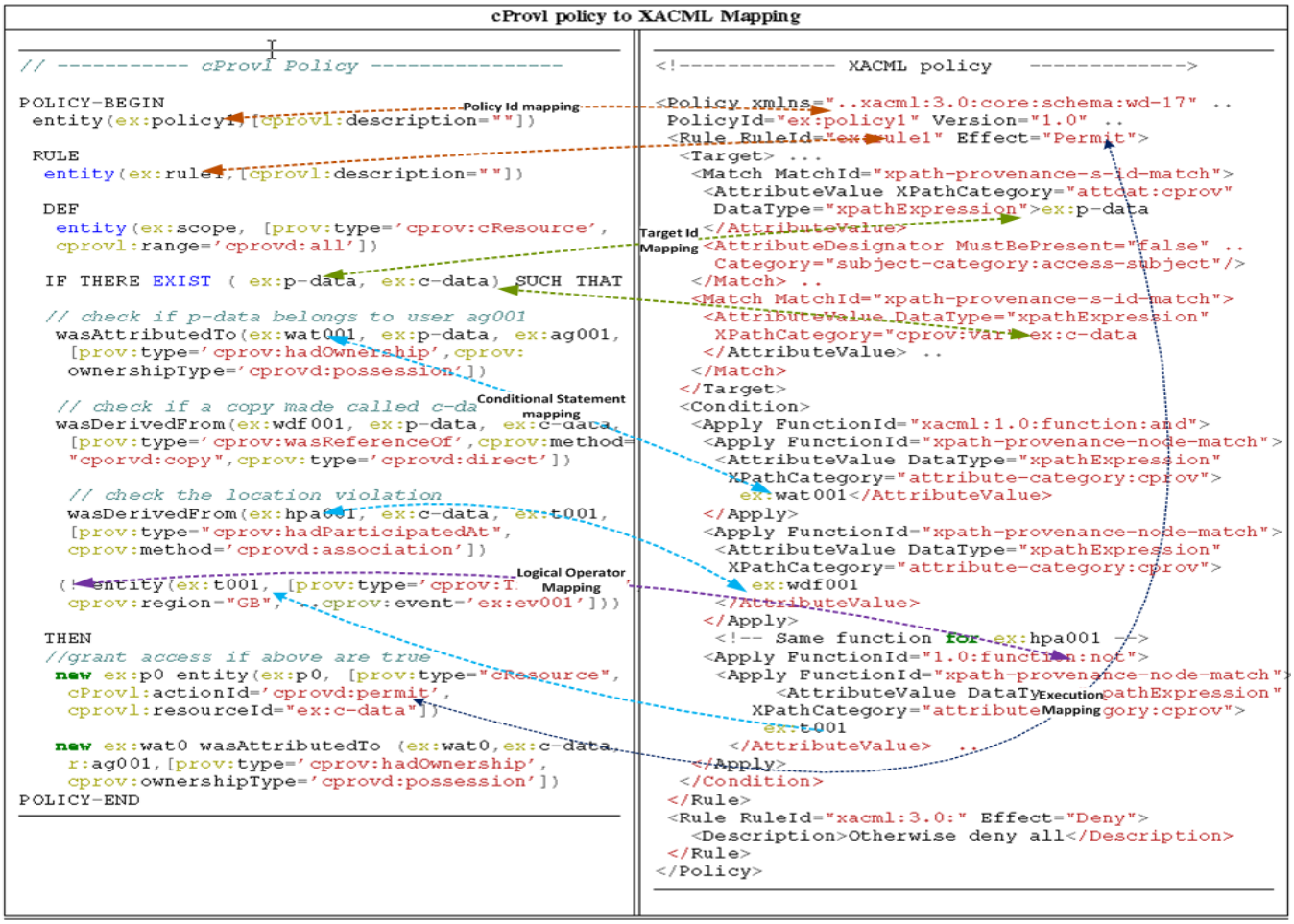

Fig. 4: cProvl to XACML - Scenario Policy One

complex provenance structure (cProv). Beyond capturing a step, it does not have a natural way of expressing relationships that exist between processes, entities and agents. However, it may be possible to encode such information using attributes, which can be tedious and cumbersome.

A. Syalim *et al.* [34], proposes an access control method for provenance based on a direct cyclic graph. Their approach is to define policies within a relational database that operates on nodes and edges of a graph (modelled as DB tables). The access control is coarse-grained (supports grouping), it lacks the flexibility to define policies attribute-based access control (cannot define policies operating at the property level of nodes and edges).

T. Cadenhead *et al.* [35] proposes an extension to the ACL [36] with regular expression grammar, to operate on provenance graphs (OPM). This allows the policies to take OPM's node and edge names into account when declaring policies. This work is very much analogous to our proposed work, however, our approach is from a holistic view, and improves in the following areas. Firstly, the policy declaration using XML is not fully coupled with the OPM model (difficult to define properties associated with nodes and edges). Secondly, It does not define rules, therefore a policy is likely to be relatively large and complex, which can affect the performance time, and likely to be prone to errors. Thirdly, the provenance of policy execution is not captured or recorded. Fourthly, the declaration of the policy values is static, and does not accommodate dynamic policy values. And finally, it is not designed to run within an existing XACML policy engine. However, a graph grammar approach for rewriting redaction policies over provenance [37] has been proposed.

VII. Conclusion

In this paper we have presented a provenance model for the cloud, and a provenance-aware policy language to operate on the model. The model defines a number of extensions of the W3C Prov to cater for cloud-based services. The policy language is designed to be tightly coupled with the Prov notations and has a greater degree of expressibility on the provenance data (relations, meta-data). We were able to successfully model our policies for the scenario using the language (cProvl). One potential drawback of our language is, that policies generally tend to be relatively large; this is primarily due to both policy and provenance statements being

presented in XACML. In the future we may consider more compact representations.

We have also defined a translator that converts cProvl request and policies into XACML policy & request. Also, additional functions to handle single-to-multi value mapping, coupling of policy assertions, and static/dynamic variables.

We have implemented the policy language with an open source XACML engine BALANA [38], and the next phase is to deploy it in our service.

ACKNOWLEDGMENT

I would like to thank Raffel Uddin, Kashif Chawdhary, Tansir Ahmed, Saiful Alam, Emanuel Mayer, Patrick Launey and other members of Orange Labs for the sponsorship and on going support of the work.

REFERENCES

[1] J. Voas and J. Zhang, "Cloud computing: New wine or just a new bottle?" *IT professional*, vol. 11, no. 2, pp. 15–17, 2009.

[2] A. Keith and A. Ole, "A comparison of software and hardware techniques for x86 virtualization," in *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS XII. New York, NY, USA: ACM, 2006, pp. 2–13. [Online]. Available: http://doi.acm.org/10.1145/1168857.1168860

[3] R. Uhlig, G. Neiger, D. Rodgers, A. Santoni, F. Martins, A. Anderson, S. Bennett, A. Kagi, F. Leung, and L. Smith, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[4] J. Rittinghouse and J. Ransome, *Cloud computing: implementation, management, and security*. CRC, 2009.

[5] A. Zahariev, "Google app engine," *Helsinki University of Technology*, 2009.

[6] A. Khalid, "Cloud computing: Applying issues in small business," in *2010 International Conference on Signal Acquisition and Processing*. IEEE, 2010, pp. 278–281.

[7] A. Lawabni, C. Hong, D. Du, and A. Tewfik, "A novel update propagation module for the data provenance problem: A contemplating vision on realizing data provenance from models to storage," in *Mass Storage Systems and Technologies, 2005. Proceedings. 22nd IEEE / 13th NASA Goddard Conference on*, april 2005, pp. 61 – 69.

[8] R. Sandhu and P. Samarati, "Access control: principle and practice," *Communications Magazine, IEEE*, vol. 32, no. 9, pp. 40 –48, sept. 1994.

[9] D. Ferraiolo, J. Cugini, and D. Kuhn, "Role-based access control (rbac): Features and motivations," in *Proceedings of 11th Annual Computer Security Application Conference*. IEEE Computer Society Press, 1995, pp. 241–48.

[10] E. Yuan and J. Tong, "Attributed based access control (abac) for web services," in *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, july 2005, pp. 2 vol. (xxxiii+856).

[11] B. Stepien, S. Matwin, and A. Felty, "Advantages of a non-technical xacml notation in role-based models," in *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*. IEEE, 2011, pp. 193–200.

[12] L. Zhi, W. Jing, C. Xiao-su, and J. Lian-xing, "Research on policy-based access control model," in *Networks Security, Wireless Communications and Trusted Computing, 2009. NSWCTC '09. International Conference on*, vol. 2, april 2009, pp. 164 –167.

[13] E. Rissanen, "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.

[14] P. Druschel, M. Backes, and R. Tirtea, "The right to be forgotten between expectations and practice," http://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/the-right-to-be-forgotten, 2012.

[15] L. Moreau, P. Missier *et al.*, "Prov-dm: The prov data model," W3C, W3C Candidate Recommendation 11 December 2012, 2012.

[16] L. Moreau, P. Missier, J. Cheney, and S. Soiland-Reyes, "Prov-n: The provenance notation," W3C, W3C Candidate Recommendation 11 December 2012, 2012.

[17] J. Cheney, P. Missier, and L. Moreau, "Constraints of the provenance data model," 2012.

[18] P. Patel, A. Ranabahu, and A. Sheth, "Service level agreement in cloud computing," in *Cloud Workshops at OOPSLA*, 2009.

[19] J. Christensen, "Using restful web-services and cloud computing to create next generation mobile applications," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*. ACM, 2009, pp. 627–634.

[20] M. Mannion, "Using first-order logic for product line model validation," in *Software Product Lines*. Springer, 2002, pp. 176–187.

[21] K. Mahbub and G. Spanoudakis, "A framework for requirents monitoring of service based systems," in *Proceedings of the 2nd international conference on Service oriented computing*. ACM, 2004, pp. 84–93.

[22] Z. Bao, S. Cohen-Boulakia, S. Davidson, A. Eyal, and S. Khanna, "Differencing provenance in scientific workflows," in *IEEE 25th International Conference on Data Engineering (ICDE'09)*. IEEE Computer Society, 2009, pp. 808–819. [Online]. Available: http://www.cis.upenn.edu/~zhuowei/docs/PDiffView/ICDEResearchLong-ZBao-diff.pdf

[23] K. Anand, S. Bowers, T. McPhillips, and B. Ludaescher, "Exploring scientific workflow provenance using hybrid queries over nested data and lineage graphs," in *Proceedings of 21st International Conference on Scientific and Statistical Database Management (SSDBM'09)*, New Orleans, LA, USA, 2009, pp. 237–254. [Online]. Available: http://daks.ucdavis.edu/~sbowers/ssdbm-09.pdf

[24] O. Biton, S. Cohen-Boulakia, S. Davidson, and C. Hara, "Querying and managing provenance through user views in scientific workflows," in *International Conference Data Engineering (ICDE'08)*. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 1072–1081. [Online]. Available: http://www.inf.ufpr.br/carmem/pub/icde08.pdf

[25] S. Sahoo, P. Groth, O. Hartig, S. Miles, S. Coppens, J. Myers, Y. Gil, L. Moreau, J. Zhao, M. Panzer, and D. Garijo, "Provenance vocabulary mappings," http://www.w3.org/2005/Incubator/prov/wiki/Provenance_Vocabulary_Mappings, 2010.

[26] P. Macko, M. Chiarini, M. Seltzer, and S. Harvard, "Collecting provenance via the xen hypervisor," 2011.

[27] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, E. Kotsovinos, A. Madhavapeddy, R. Neugebauer, I. Pratt *et al.*, "Xen 2002," *University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-553, Jan*, 2003.

[28] K. Muniswamy-Reddy, P. Macko, and M. Seltzer, "Provenance for the cloud," in *Proceedings of the 8th USENIX conference on File and storage technologies*. USENIX Association, 2010, pp. 15–14.

[29] M. Seltzer, K. Muniswamy-Reddy, D. Holland, U. Braun, and J. Ledlie, "Provenance-aware storage systems," in *Proceedings of the USENIX Annual Technical Conference (USENIX06)*, 2006.

[30] Y. Doganata, K. Grueneberg, J. Karat, and N. Mukhi, "Authoring and deploying business policies dynamically for compliance monitoring," in *Policies for Distributed Systems and Networks (POLICY), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 161–164.

[31] D. Abi Haidar, N. Cuppens-Boulahia, F. Cuppens, and H. Debar, "An extended rbac profile of xacml," in *Proceedings of the 3rd ACM workshop on Secure web services*. ACM, 2006, pp. 13–22.

[32] A. Anderson, "Xacml profile for role based access control (rbac)," *OASIS Access Control TC committee draft*, vol. 1, p. 13, 2004.

[33] C. Ringelstein and S. Staab, "Papel: a language and model for provenance-aware policy definition and execution," *Business Process Management*, pp. 195–210, 2010.

[34] A. Syalim, Y. Hori, and K. Sakurai, "Grouping provenance information to improve efficiency of access control," *Advances in Information Security and Assurance*, pp. 51–59, 2009.

[35] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. Thuraisingham, "A language for provenance access control," in *Proceedings of the first ACM conference on Data and application security and privacy*. ACM, 2011, pp. 133–144.

[36] Q. Ni, S. Xu, E. Bertino, R. Sandhu, and W. Han, "An access control language for a general provenance model," *Secure Data Management*, pp. 68–88, 2009.

[37] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. Thuraisingham, "Transforming provenance using redaction," in *Proceedings of the 16th ACM symposium on Access control models and technologies*. ACM, 2011, pp. 93–102.

[38] "Getting start with balana," http://xacmlinfo.com/2012/12/18/getting-start-with-balana/, 2012.