Extending and improving the usability of an application to investigate the performance of ICT access control systems

Shahzada Ali Saleem, BSc.

School of Science and Computing

Waterford Institute of Technology

This dissertation is submitted for the degree of

Masters of Science

Supervisors: Dr. Brendan Jennings and Dr. Bernard Butler

September 2016

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Masters of Science, is entirely my own work and has not been taken from the work of others save to the extent that such work has been cited and acknowledged within the text of my work.

Shahzada Ali Saleem, BSc.
September 2016

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisory team *Dr. Brendan Jennings and Dr. Bernard Butler* for their continuous support for my MSc study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisory team and mentors for my MSc research.

I would like to thanks *WIT* who provided me this opportunity and Science Foundation Ireland (SFI) for funding A dimensioning TooL for Access control Systems (ATLAS) project supporting my studies financially. Also I want to thanks to *Telecommunications Software and Systems Group (TSSG)* who provided me extra ordinary study environment and resources, without their resources and environment I would never be able to complete my project and thesis on time.

I thank *Asela Pathberiya* for assistance with performance of Balana Policy Decision Point (PDP).

I would also like to show my gratitude to Dr. Alan Davy, ENL Unit Manager, TSSG for sharing his pearls of wisdom with me during the course of this research, and I wish to thank three anonymous reviews for helpful insights based on a paper submitted for publication.

Last but not the least, I would like to thank my family: my parents, my wife and to my brothers and sisters for supporting me spiritually throughout writing this thesis and my life in general.

Abstract

Information Technology (IT) platforms enable digital artifacts such as spreadsheets and videos to be shared, but sensitive data such as trade secrets and personal information needs to be protected from unauthorised access. Security rules are checked whenever access is requested. Security checking can quickly become a bottleneck and make the IT platform much more difficult to use, so the Scalability Testbed for Access Control Systems (STACS) testbed has been built in TSSG (Butler et al., 2010) to make it easier to study the conditions where access control performance becomes unacceptable. To use the testbed, it is necessary to configure it to match the scenarios being studied. Butler and Jennings (2015) introduced the ATLAS framework that incorporates this testbed and adds extra components a) to configure it to undertake more complex performance experiments (notably, by generating suites of policies and requests with differing characteristics) and b) to analyse the results of these experiments.

This dissertation describes work relating to the extension of the ATLAS system 1) to provide an easy-to-use GUI for specifying parameters relating to the generation of XACML policy and request sets and 2) for the support of XACML 3.0 as well as XACML 2.0. The key contributions are:

1. A workflow and GUI that breaks down the task of configuring the policy and request generator of ATLAS into easy steps using an attractive editing application with a focus on information design and usability;

2. Adding the option to export sets of policies and requests in the EXtensible Access Control Markup Language Version 3.0 (XACML3.0) standard, building upon the existing ATLAS infrastructure for exporting EXtensible Access Control Markup Language Version 2.0 (XACML2.0) policies and requests;

3. Adding an adapter to integrate a new implementation of PDP for evaluating requests based on XACML3.0 standard called BalanaPDP in STACS.

Contents

List of Figures

Chapter 1

ATLAS System

1.1 Introduction

Many IT organizations ensure that their resources are protected from unauthorized access. Systems that are built to manage the rules governing access to resources become more complex as policies for the organization grow. Researchers have devoted considerable effort in developing methods to measure the correctness of their access control rules. Nonfunctional requirements, specifically the performance of the assessment of those policy sets, have recently become more important. This is because application distributors have attempted to apply access controls to fine-grained resources such as parts of websites, and these access controls decisions need to be produced in real-time.

As seen by those experiencing the problem, the end-user problem is: access control requests are sent and the delay before the response is given exceeds what is acceptable. The system administrator's problem is: they try to increase performance but are unsure what changes to make or why the problems arose in the first place. The system supplier's problem is: how do they reconfigure the system and/or support the system administrators' efforts. Our work seeks to *extend* a software system that helps address these concerns.

1.2 Background and Related Work

Currently there is a DomainManager (Butler and Jennings, 2015) system that generates the sets of policies and Requests sets, by getting

(a) Access rules (the primary components of a policy)[1], defined by the *security* administrator, and

(b) A static model of the domain, defined by the overall *system* administrator

as input and then it transforms the given information into its internal representation. In many cases, more than one set of policies and requests will be generated, e.g., if the intention of the investigation is to compare different “flavours” of policy and request set. For example, the purpose of the study might be to compare the effects of different ways of formulating the policies, or different types of organisation. In such cases DomainManager will generate multiple policy and request combinations from this data. Thus DomainManager is more than just a policy editing system: it also creates many of the artifacts needed for running access control performance experiments.

The graph defines the permissible actions that can be performed by authenticated users to access resources in an organization. DomainManager also provides a way to export policies and sets of requests in selected formats that are suitable for policy evaluation and as the input for STACS. Currently the operation of DomainManager has to be manually controlled and configured for STACS (Butler et al., 2010), see Fig 1.1.

STACS takes the output of DomainManager as its input and tests the performance of a supplied set of requests against given policy sets and generates performance statistics under different circumstances. The basic data is measurements of service time per policy-request combination, where the service time is the time needed to produce a response for a given request. The generated output is then processed to get results in the required format.

STACS processes sets of requests against policies and outputs the decisions against each request and also measures the performance of each request evaluation. Butler et al. (2010) compared two different XACML PDPs using the same policy and request sets to compare their performance profiles 1.2.

According to fig 1.2 results, the PDPs (EnterpriseXACML PDP and SunXACML PDP) have different service time profiles. Most of the requests have the same service times for EnterpriseXACML PDP (approx. 1.4ms) but the service times for SunXACML PDP are more varied. So, the comparative analysis using STACS with those policies and requests shows that EnterpriseXACML PDP has much more predictable performance than SunXACML PDP.

Butler et al. (2011), described an experimental testbed (STACS) to predict the performance of XACML policy decision points. This experiment was conducted on the same two XACML PDP implementations. An analytical model for predicting performance was

[1]In this dissertation, we refer to such access rules as a template policy (set).

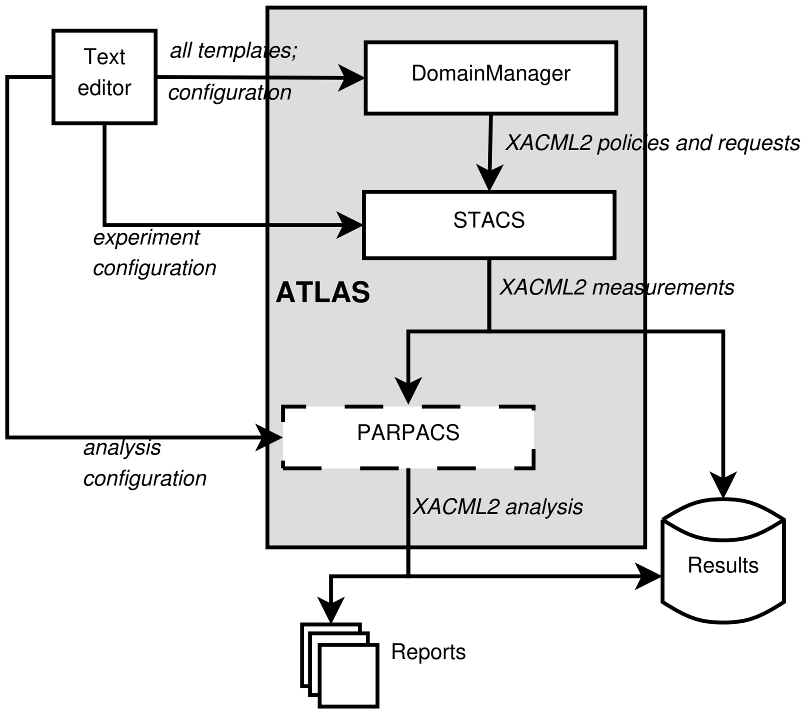
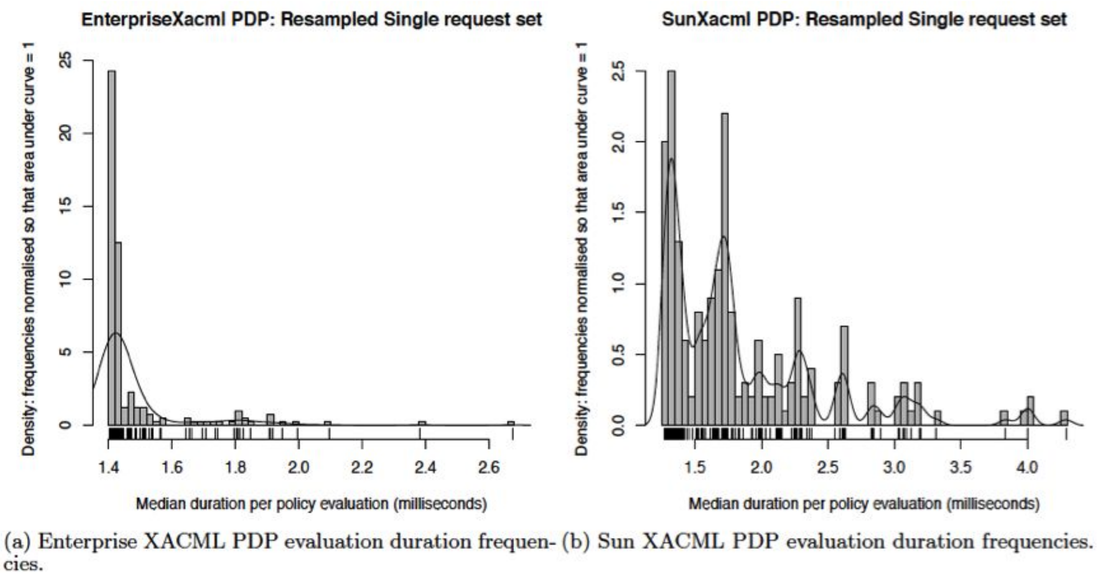


Fig. 1.1 Capabilities of ATLAS *before* adding a GUI for easier policy editing and XACML 3 policy and request export option.



(a) Enterprise XACML PDP evaluation duration frequencies. (b) Sun XACML PDP evaluation duration frequencies.

Fig. 1.2 XACML Policy Performance Evaluation Using a Flexible Load Testing Framework, from (Butler et al., 2010).

proposed and validated by comparison against a discrete event simulation of a PDP subject to additional load. These predictive models extend STACS by enabling it to consider the effects of different a) relative frequencies of requests and b) request arrival patterns.

The experiments in (Griffin et al., 2012) suggests that newer technologies offer better performance and analysis suggests that this is because they offer a more efficient data representation and make better use of computing resources. They used the Node.js framework because of the non-blocking nature of JavaScript. Node.js allows an elegant solution for traditional scalability problems and an alternative approach for domains that might benefit from a non-blocking I/O approach. For storage of policies they used the NoSQL database Redis in their system to deliver horizontal scalability.

> "The data structure-oriented Redis Server, Lerner (2010), was chosen because of its speed (in 1 minute, on one instance of the experimental platform, it averaged 11300 answered requests per second) and its data structures directly supported sets, lists and hashes, easing policy management." (Griffin et al., 2012)

STACS was used to run the performance experiments in Griffin et al. (2012), but it needed to be configured manually. This makes it more difficult to consider other enhancements, such as new policy languages.

In this regard, XACML3.0 is an enhanced version of the XACML 2.0 standard for access control policies used in Butler et al. (2010, 2011) and Griffin et al. (2012) with lots of new features. Such features might increase the performance and scalability when writing and processing policies and requests. Some major changes were described in XACML-TC (2013), notably the fact that some elements such as <Subject>, <Resource> etc., have been reclassified as attributes that can be combined in new and more flexible ways in the <Target> element, e.g., as the union (ANY) or intersection (ALL) of such entities. The new structure is both less rigid and more compact. This new feature has the potential to reduce the textual size of policies which might also speed up the process of reading policies. Along with reclassification of attributes, XACML3.0 comes with other features such as Multiple Decision Profile (MDP), which extends the more limited XACML 2.0 concept of multiple responses.

1.3 Research Questions

This section outlines the research questions that scoped the work proposed in the research programme. It outlines the high level options for the extension of the ATLAS System that will build upon the existing DomainManager and STACS components. ATLAS can be used to write policies and requests based on business requirements and to predict policy evaluation performance.

RQ1 How do we extend DomainManager and STACS to make it easier (and less error-prone) for system administrators to specify policies within ATLAS?

RQ2 What performance benefits can we obtain from offering new policy structures, such as XACML3.0, when using the ATLAS system to analyse the performance benefits (if any)?

These Research Questions address separate concerns about the existing ATLAS system. The first is concerned with the usability of the system, particularly relating to the configuration of ATLAS. One of the biggest configuration challenges relates to how policies used in performance tests should be specified. To answer this Research Question, it is necessary to understand how such policies are specified, and to look at ways in which the task can be made easier and less prone to errors.

Given a more reliable set of policies for use in performance experiments, it is then instructive to see how to leverage that policy specification in new ways. For example, the DomainManager component of ATLAS has a very flexible internal representation of policies and requests, but *to date* the policies and requests that it shares with STACS have all been expressed in XACML2.0-compliant XML files. Furthermore, previous STACS experiments have been restricted to XACML2.0-based access control infrastructure. The obvious question is: what about other policy representations and formats? Both DomainManager and STACS have been designed to support other policy formats, but does this work in practice? Furthermore, does changing the policy format improve performance or not? ‘

1.4 Research Methodology

This research was conducted in a phased approach, as follows:

1. Understanding of Existing DomainManager and STACS: We learned how the existing DomainManager application works and how it produces the sets of requests and policies and also we have done in-depth study of the current STACS implementation.

2. Selection of an appropriate tool and platform for development: After understanding the requirements of the ATLAS system, we searched for the tooling and platform that helps the system administrators to configure DomainManager and STACS to setup

and run performance experiments and to understand the results of those experiments, greatly simplifying the control of the current system.

3. Development of a system to incorporate DomainManager and STACS: After selection of the tools and platform we extended the current system to build upon the existing support for different hierarchical policy languages to add code that serializes policy and request objects to XACML3.0 text format.

Many rule-based policy language are hierarchical by nature, with complex combining algorithms. Some, such as XACML use a hierarchical syntax, because they can be written as XML. Other examples include PERMIS (Chadwick et al., 2008) and NGAC (Ferraiolo et al., 2016). Many of the alternative languages have aspects that favour particular use cases, whereas XACML is intended to be completely general purpose. Therefore, as a first step towards such investigations, we consider comparisons between versions of XACML in Phase 3 above.

1.4.1 Impact of Research

We have built an ATLAS GUI that simplifies the task of creating the specification of the template policies and of the static domain. Consequently, it is much easier to provide this data to DomainManager, and to have more confidence that it does not contain errors. Because we have selected the *Play Framework* to build ATLAS GUI whose advantages are indicated in Section 3.4.

We have also extended DomainManager so that it can generate XACML 3.0 policies and requests from the same source that is used for XACML 2.0 polices and requests. Lastly, we have added an extra adapter to STACS so that Balana PDP can participate in STACS performance experiments. Therefore ATLAS can be used to answer more varied research questions relating to access control performance.

The schematic of the ATLAS system after our extensions have been added can be seen in fig 1.3. The following points are noteworthy. Firstly, ATLAS GUI takes over responsibility for specifying template policies, but other configuration specification (of template requests, other DomainManager settings and *all* STACS and Performance Analysis, Reporting and Prediction of Access Control Systems (PARPACS) settings) continue to rely on text editors. Secondly, the PARPACS component of ATLAS (delineated by a broken line in both fig 1.1 and fig 1.3) is not used in this dissertation, because simple tables and plots are sufficient for our purposes. Thirdly, although the inputs to DomainManager largely remain the same (apart from an extra flag to indicate which version of XACML is to be used), the extended

DomainManager has more output options and STACS has been extended to support these. Lastly, the direction of the arrows in the diagram indicate the workflow followed by ATLAS.

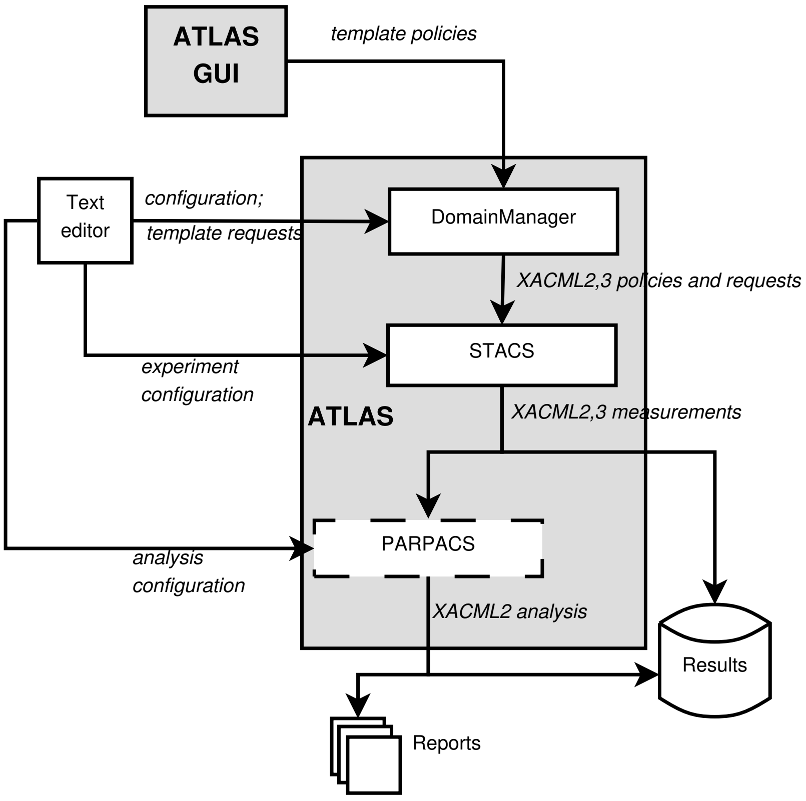


Fig. 1.3 Capabilities of ATLAS *after* adding a GUI for easier policy editing and XACML 3 policy and request export option.

Summarising, ATLAS helps the security administrator to write suites of policies and requests for performance experiments. The user just needs to specify template policies and requests, and DomainManager will generate full policies and requests matching that template. The added features make it easier for users to create these template policies, and they also support new types of policy comparison (XACML2.0 versus XACML3.0). The ATLAS components were designed to facilitate such new comparison functionality. When the performance measurements form a comparison experiment have been analysed, the security administrator can review the generated results and revise the policies and configuration based on performance needs.

Chapter 2

Literature Review

2.1 Background

In this section we review what access control is. We also consider how access controls can be specified as *policies* and how those access controls are applied in practice. We also discuss the issue of access control evaluation performance, which motivates the research described in this dissertation.

We favour the definition presented in (Samarati and Vimercati, 2001) as

> “Access control is the process of mediating every request to resources and data maintained by a system and determining whether the request be granted or denied. The access control decision is enforced by implementing regulations established by a security policy. Different access control policies can be applied, corresponding to different criteria for defining what should, and what should not, be allowed, and, in some sense, to different definitions of what ensuring security means."

According to this definition, the organisation needs to specify protection policies (see Section 2.1.3) and to ensure that each time a protected resource is to be used, an access request should be created and sent to an entity that checks whether that access should be granted, by checking the relevant access control policies.

2.1.1 Access Control as a Security Mechanism

We can understand access control in practice as the following. When a client and a server have established a secure channel, the client can issue requests to the server. Requests can be granted only if the client has sufficient access rights. The verification of access rights is called *access control*, and the granting of access rights is called *authorization*. These two terms are often used interchangeably.

In order to protect the company resources from unauthorized access, several policy models have been defined and refined as needed by the complexity of any organization. Among these policy models Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role-Based Access Control (RBAC) have been popular access control models. Initially DAC has been introduced to restrict the access of any object based on the owner of that object. DAC has two types of policies: *Closed* DAC which requires authorization explicitly because of its default decision is to deny access and *Open* DAC in which we have to specify denials explicitly as its default decision is to permit access. But as a security measure there is a danger of incorrect user configuration and also the dissemination of information is not controlled: for example a user who is able to read data can pass it to other users not authorized to read it without the security system being able to intervene.

MAC comes into play to minimize the main security weakness in DAC, where a person can who has only read access can send the object secured by DAC to another user who otherwise might not be allowed to access the object. MAC requires that each subject and object/resource is assigned a security level. For subject it will specify the level of trust associated with it and resource/object will have a specification which ensures the minimum level of trust required by the subject that wishes to access that particular resource. This technique addresses the transfer of rights problem but the security levels reduces the flexibility of policy authors. There are several implementations of MAC, such as Military security policies, grouped individual and resource policies and other policy concepts which include “Chinese Wall" policies.

	Asset 1	Asset 2	File	Device
Role 1	read, write, execute, own	execute	read	write
Role 2	read	read, write, execute, own		

Table 2.1 Example of Access Control Matrix (ACM)

Lampson (1974) described ACM or Access Matrix (AM) as a model for identifying the rights of each subject with respect to every object in the system. ACM is designed in a matrix pattern to fire the rule mentioned in a specific cell as shown in table 2.1. This way of representing access controls became very popular but it had flaws as well. To follow this technique for writing access controls there were issues of scalability, memory and flexibility. As we can see in Table 2.1 that, in order to add new resources or subjects, we have to add specific new rows or columns which will consume memory and so it is difficult to maintain these access controls in matrix form as the number of resources and subjects increase. Access Control List (ACL) have been used to implement ACM but they are not flexible enough for modern authorization requirements, where access control requirements are expressed as policies.

Jin et al. (2012) claimed that Attribute Based Access Control (ABAC) has strong capabilities to overcome the restrictions that have been identified in many of discretionary access control models like DAC, MAC and RBAC along with integrating their advantages in ABAC. ABAC is formulated in terms of conditions on attributes, and the choice of what attributes are included, and what conditions might be placed on those attributes, is largely under the control of the policy author. Consequently, it is possible to represent these older models in terms of an ABAC representation. However, that degree of flexibility can have drawbacks too. Researchers have already done good research on ABAC and we can find several implementations of ABAC in industry. But having said that, ABAC still lags behind the discretionary access control models when it comes to acceptance rate of ABAC over simpler and more established models. Jin et al. (2012) described how the ABAC model can easily be configured to have the advantages of DAC, MAC and RBAC. They proposed a generic model called ABAC “alpha" model and they described that this model can be used to naturally configure three classical models, i.e, DAC, MAC and RBAC. Also they proposed that some classical models can be improved, such as in MAC if we categorize the subjects into different types as read-only and read-write for both security and availability. By making this change, the rule governing their actions can be different in that read-only subjects are allowed to read all levels of objects. By contrast, read-write subjects’ actions are strictly regulated. Another example is in RBAC, where a certain level of automatic permission-role assignment can be achieved by interpreting permissions as accessing a group of objects with the same attribute expression.

2.1.2 Applying Access Controls

As we have seen in Section 2.1.1, ABAC offers great scope and flexibility to write policies. Indeed, the XACML standard has been developed to write fine-grained policies consistent with the ABAC model. Also, RBAC policies can also be implemented in XACML standard as a specialization of ABAC (De la Rosa Algarín et al., 2014).

Organization for the Advancement of Structured Information Standards (OASIS) is a nonprofit organisation which helps to develop, maintain and foster the adoption of standards for security and interoperability. An OASIS committee, with a core group of members drawn from IBM, Sun Microsystems, and Entrust Inc, defined the first Extensible Markup Language (XML)-based standard called XACML for writing fine grained ABAC policies. In first version of the XACML standard from OASIS, (XACML1.0-draft) defines the structure of XACML policies and requests. There are also several resources that are structured as hierarchies within this mark-up language. This structure focuses on the resources represented as nodes in the XML text files or characterized in a non-XML pattern. In this draft, they introduced three top level policy elements "<Rule>","<Policy>" and "<PolicySet>" which form the basic structure for XACML1.0 policies. "<Rule>" contains the "<Target>" comprising the "<Subject>" which needs "<Permission>" to perform certain "<Action>" on requested "<Resource>"s. "<Policy>" can contain sets of "<Rule>" elements and some supporting elements. "<PolicySet>" contains sets of "<Policy>" elements. Also, the XACML context in (XACML1.0-draft) can include "<Environment>" elements to add extra context. All of these can contain <Target> elements with each Target mapping to a set of cells in an ACM. Furthermore, "<Rule>" elements can be further constrained by "<Condition>" elements, with extensible complex logic. This policy language is defined in an XML schema which describes the required structure of the inputs and outputs of the PDP, as shown in Figure 2.1. Attributes referenced by a XACML policy need to be resolved to actual subjects, resources, etc. How this is achieved is an implementation detail that is beyond the scope of the XACML standards. Typically, implementations must convert between attribute representations in the application (e.g., SAML, J2SE, CORBA, and so forth) and the abstract representation in the XACML policy.

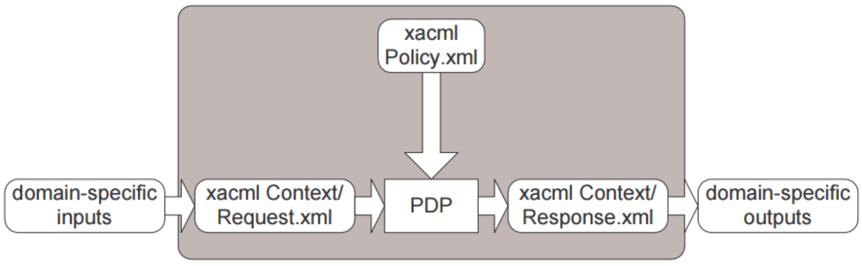


Fig. 2.1 XACML context

Rouse (2005) mentioned that the XACML standard was originally designed to work in sync with the Security Assertion Markup Language (SAML), which is another OASIS standard. SAML defines a means of sharing authorization information between separated security systems, such as passwords or security clearances. A program which examines established rules in order to suggest behaviors that comply with those pre-defined orders is a *rules engine*. For policy evaluation this rules engine is more commonly called a PDP. XACML policy evaluation takes the following steps:

1. (Offline), policy authors use the Policy Administration Point (PAP) component to write policies and policy sets and make them available to the PDP. These policies or policy sets represent the complete policy for a specified target.

2. The access requester sends a request for access to the Policy Enforcement Point (PEP).

3. The PEP sends the request for access to the context handler in its native request format, optionally including attributes of the subjects, resource, action and environment.

4. The context handler constructs a XACML request context and sends it to the PDP.

5. The PDP requests any additional subject, resource, action and environment attributes from the context handler.

6. The context handler requests the attributes from a Policy Information Point (PIP).

7. The PIP obtains the requested attributes.

8. The PIP returns the requested attributes to the context handler.

9. Optionally, the context handler includes the resource in the context.

10. The context handler sends the requested attributes and (optionally) the resource to the PDP.

11. The PDP evaluates the policy.

12. The PDP returns the response context (including the authorization decision) to the context handler.

13. The context handler translates the response context to the native response format of the PEP. The context handler returns the response to the PEP.

14. The PEP fulfills the obligations.

15. (Not shown) If access is permitted, then the PEP permits access to the resource; otherwise, it denies access.

Apart from the first step in the list above, all the above steps are the basic steps in which a request comes to be evaluated in XACML. But the main elements here to discuss are “context handler” and “PDP”. The “context handler” is responsible for creating a XACML based request context which is then passed to PDP where it will be processed according to the implementation of given PDP. Different organisations have implemented PDPs to evaluate XACML requests, for example Sun developed SunXACML PDP to evaluate XACML requests written in XACML 1.0 standard. Later on it added the support to XACML 2.0 policies.

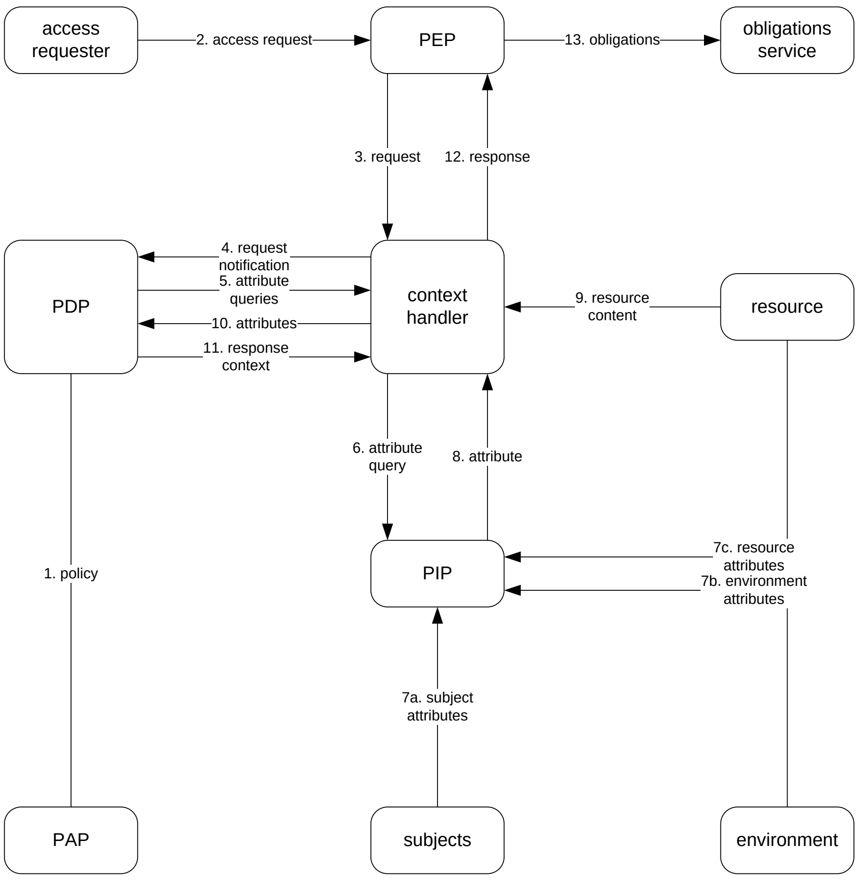


Fig. 2.2 XACML Data Flow

XACML 2.0 became established and many organisation have written and evaluated their policies according to that standard. However, XACML 3.0 was introduced in 2011 (XACML3.0-specification-draft). Axiomatics were among the first commercial vendors to implement the XACML 3.0 standard. The most remarkable enhancements in this new standard include multiple decision profile, delegation, obligation expressions, advice expressions, new and revised policy combination algorithms, new attribute functions and data types, new profiles and enhanced profiles.

MDP The *multiple resource* feature in XACML 2.0 has been extended to the *multiple decision* feature in XACML version 3.0. This feature enables policy authors to specify that the PEP can process several requests in one run, since the PDP can reply with multiple decisions in one answer. The main use of this feature is to support web-based scenarios where the grouping of decisions in this fashion reduces the next for context switches and redundant communication between the PEP and PDP.

Delegation This is a completely new feature. This option enables local administrators to make authorized decisions according to the policies that the main administrator has shared previously. This feature is especially useful in (multi-tenant) cloud and federation scenarios where local administrators are essential.

Obligation expressions Another new feature which ensures that any actions (such as logging the fact that access has either been permitted or denied) consequent on a decision are passed to the PEP for implementation.

Although the new features are powerful, the move to XACML 3.0 is not trivial as policies need to be re-engineered. Version 3.0 will become more common as more enterprises, vendors and service providers update their platforms.

XACML3.0 vs XACML2.0 specifies the main differences between version 2.0 and 3.0 of XACML. Whereas obligations can be added to policies and policy sets exclusively in version 2.0, it is also possible to include obligations in the rules in the version 3.0. Another difference involves the content element, XACML 2.0 accepts XML content inside the ResourceContent element whereas the ResourceContent element becomes a more generalized section in the version 3.0 included in the Content element that now appears in ANY category. More upgrades relate to the scope of XPath expressions, the target elements, the creation of custom categories, new profiles and updated profiles in version 3.0.

Multiple resource profile of XACML2.0 provides the specifications and standards for the multiple resource profile of XACML 2.0. The profile used for requesting access to several resources in the same XACML Request Content is explained in detail in these pages. Access requests receive one single answer to multiple inquiries that belong to the same category.

To evaluate the XACML3.0 based policies WSO2 extended the sun-XACML PDP implementation. Pathberiya (2012) talked about the latest open source XACML implementation which is based on SunXACML PDP. Currently, WSO2 Balana supports the XACML 3.0

specification with MDP features. Balana offers the following features to developers: the most powerful XACML 3.0 support, multiple decision profile assistance, a file-based policy finder module as default, proper logging with *log4j*, *maven* support for compilers with unit tests, several samples, diverse utility methods to create XACML 3.0 policies, and much more.

Butler et al. (2010) noted that performance is a concern in any complex ICT system such as a XACML PDP.. On top of that, the scalability of the access control systems is a matter growing in relevance as a larger number of institutions deploy more complex communications structures and, therefore, require ever more complex content management systems. in that regard, security administrators require fine-grained access control. As the decisions happen to be more frequent and the policy sets become larger due to the large-scale structures, policy evaluation time can become significant and affect the performance of the overall system, as perceived by end users. Butler et al. (2010) introduced a testing framework to enable experiments to be undertaken, providing security administrators with instrumentation and analysis tools to find where performance problems arise.

To increase the performance of the decision process, Ömer Malik Ilhan et al. (2015) showed that multiple service providers in combination with high demanding customers who send requests 24/7 demand a solid communication system. Moreover, there are additional requirements for online resources, such as the enforcement of legal regulations as well as compliance with enterprise requirements. The technologies and systems must ensure security and privacy at the same time in order to create a realistic environment for users all over the world. Managing the access rules and enforcing the authorizations aimed to control the data and the resources at the same time is a big challenge.

2.1.3 XACML XML editor history

The first step when writing XACML policies is to derive the access rules that apply to the given domain. Often those rules are not available in explicit form, so the rules are collected in stages from multiple sources. The resulting formal policies can have gaps, conflicts and/or obsolete rules, all of which can lead to specification errors (Barron, 2013).

If the security administrator wishes to create XACML policies directly, there is a high cognitive load because of the need to manipulate both the domain concepts and their expression in verbose XML format. Therefore, writing XACML policies using a simple text editor was very difficult in the past. Most of the model structure is based on XML type tags. To handle this issue many researchers have created XML editors which are often based on a tree view of an XML document (such as `XMLPad` which has three different views).

Bootstrap is one of the most dominant framework to develop HTML, CSS and Javascript enabling responsive and versatile web-oriented designs. The predominant aim of this utility is to provide a front-end agile developing environment that suits every project and any coder. Also, due to its versatility for mobile friendly user interface it became very popular as a front end framework which helps the designer to focus more on design rather than designing different interface for other devices such as tablet, mobile etc. This framework comes in a role if organisation has to write, modify or run the policies remotely in order to test their new modification in their policies.

(Stepien et al., 2009) highlighted the contribution of the first XACML-specific editor, which was developed by the University of Murcia. However, that first XACML editor was not easy to use by non technical users. The more recent *XACML Studio* is an improvement but it still emphasizes the XACML tree structure when representing policies. To overcome this issue, (Stepien et al., 2009) proposed a new notation to create a hybrid between XACML formalism and plain English to represent policies, and they also divide the policy writing into two main parts. The first is data typing definitions (which are tricky and need expert input) while the second (which is more high level) focuses on adding rules that are expressed in terms of these data elements.

More recently, Butler and Jennings (2015) took a great step forward to make it easier for non-expert users to write complex organisational policies, because subtle/tedious aspects of policies are no longer required. However, the outstanding difficulty with their approach is that users edit text files directly. Unfortunately, non-expert users are prone to making mistakes when editing text files, e.g., by mistyping a name. Such mistakes are not caught at the time of editing those policies, so this is a problem.

2.1.4 System Performance Improvement

Researchers have tried to improve policy evaluation performance. Among the solutions they have proposed are adjusting the policy set, re-structuring the PDP, decomposing policies completely, and summarizing their evaluation. The benefits and compatibilities of these improvement proposals depend strongly on the scenario to be deployed and implemented. Security administrators do not really have the chance to test every possible method in order to come up with a performance improvement approach that works.

Liu et al. (2008) addresses the significant need in policy-based processing for fast policy analysis. Also Liu et al. (2008) suggested two fundamental approaches to evaluate policies, policy normalization, and canonical representation, which promote and support the separation of correctness and performance concerns for policy designers. Further they designed an algorithm to process XACML policies and implemented that algorithm in XEngine to compare the performance with SunXACML PDP. Their experimental results shows that XEngine evaluates policies faster than SunXACML PDP. Furthermore their study contributed more in evaluating XACML policies and their policy normalization and canonical representation can be used in other policy languages. XEngine helps to speed up the evaluation and to support their research work they have extended their XEngine algorithms to Enterprise Privacy Authorization Language (EPAL) and not just to the original XACML formulation (Liu et al., 2011).

Kohler and Brucker (2010) describe caching strategies for access control policy evaluations, and there are numerous proposals for better PDPs, notably (Liu et al., 2008; Ngo et al., 2013; Pina Ros et al., 2012) and better arrangements of policies (Marouf et al., 2011; Miseldine, 2008) designed to improve performance.

2.1.5 Understanding System Performance

The first PDP-PDP comparisons were published in (Turkmen and Crispo, 2008). However, they do not give enough information regarding the experimental setup (notably, the policies and requests they use) so that other researchers can recreate their measurement experiments. Nor do they derive a performance model. That is what the ATLAS framework does: experiments are controlled, repeatable and statistically well founded. The OASIS standard, XACML, is clearly the most popular policy specification language meant to direct the access control systems. Among the bright sides, there is the simplicity in the syntax, the strength in coverage and the overall versatility which makes this language compatible with several environments, such as Service Oriented Architecture (SOA) and Peer-to-Peer (P2P) structures. Although the implementations may vary for XACML, there are open source releases which are the focus of the study in this dissertation. The policy and request settings differ from one implementation to another among the samples of study, but the study comes up with some general guidelines which can serve as hints to policy writers or system developers. But there is a difficulty in their publication that they did not share their policies and requests, or a means to generate them, so it is impossible to use them as a basis for future experiments, e.g., using new PDPs. Also, it is difficult to tell whether the policies and requests have similar characteristics to those in a given domain, e.g., banking or education etc.

Butler et al. (2011) examined two of the most important features of XACML policy evaluation. These characteristics are two of the top concerns in any complex communication network or content management environment. An off-line experimental testbed can be used to identify many performance concerns prior to an on-line deployment. In their approach, timing measurements are collected from the PDP under test, using public policies and requests consistent with these policies. The request service times were then processed by an algorithm for request cluster identification, to classify the requests by *evaluation type*.

Butler et al. (2011) show that knowledge of the evaluation types makes it possible to estimate the aggregate policy evaluation performance for different access control scenarios. That is, if we can classify the requests that are associated with a particular access control scenario by evaluation type, we can predict the performance of this access control scenario based on analysis of measurements from the testbed under controlled conditions. Using the (Butler et al., 2011) approach, with off-line testing and the fitting of performance models to the measured service times, network administrators are able to explore the steady-state capacity of the PDP in different overall scenarios (characterized by requests per unit time and the relative frequency (by evaluation type) of access requests).

Butler and Jennings (2015) claim that access control systems are becoming more pervasive, as the traditional boundaries become less distinct (with shared infrastructure) and as data finds new uses. This means that access control performance will become even more challenging, and any weaknesses will become more apparent. They note that, in the absence of comprehensive performance models, off-line experiments supplemented with tailored performance models, as presented in (Butler et al., 2011) remain promising. However, Butler and Jennings (2015) describes how more powerful configuration is needed, the experiments need to be extended and the analysis of the service times greatly enhanced to make this approach more generally useful. Butler and Jennings (2015) present the ATLAS framework in response to this need, which offers.

- `DomainManager` for specifying performance experiments
- `STACS` for running performance experiments
- `PARPACS` for analysing the results of these experiments

The `DomainManager` component facilitates the modeling process of the domain and automatizes the generation of large numbers of policies and their corresponding requests. The extended STACS component enables controlled experiments to be performed, resulting in service time measurements that are as representative of the actual deployment as possible. The `PARPACS` component fits performance prediction models, and validates those models so that they are as reliable as possible. Consequently, security administrators are able to specify/configure experiments that take account of specific access control scenarios, perform experiments to measure service times under a range of possible conditions, and use powerful performance models to predict performance in the access control scenarios of interest.

Earlier versions of STACS were presented in Butler et al. (2011) and similar papers, and they themselves extended the concepts of Turkmen and Crispo (2008), which was the first paper to consider building *testbed* for comparing the performance of different PDPs. STACS extended this to consider other types of comparison, e,g., between different server hosts.

The textual notation used by ATLAS for generating policies has almost no bloat, so it is easier for humans to *read* than highly verbose XML. However, only simple text editors are available to create the configuration files (notably the policies), so the cognitive load on policy authors is high, possibly leading to the accidental introduction of errors when *writing* policies.

ATLAS uses an extremely flexible representation of policies and requests internally and has modules to serialize its internal format to XACML 2.0 policies and requests. Hooks have been left to support serialisation to other policy languages, but the serialisation modules for those languages (such as EPAL and XACML 3.0) need to be written.

ATLAS system supported XACML2.0 for writing and processing policies. Although XACML2.0 is the main dialect in use at present, it is limited in various aspects. XACML3.0 was released in 2013 and is claimed to be an improved version of XACML2.0. Vendors are moving to/promoting XACML 3 but users are reluctant to change their policies to take advantage of the new features. XACML policies are very complex and administrators are nervous about implementing any change that might affect their meaning, even unintentionally. Also, they are not convinced yet that XACML3.0 will be a benefit or a problem for policy evaluation performance. Vendors are trying to make migration easier by streamlining the policy authoring process, the idea being that XACML3.0 policies would be supported by better tools and so this would encourage policy authors to switch. However, the concerns about performance have not been resolved yet. DomainManager supports different export formats by providing hooks for different export language syntax; the XACML2.0 XMLformat is the default.

2.2 Summary

As can be seen, access control policy evaluation performance is an important consideration for security administrators, because access controls are generally expressed as policies, those polices are becoming more complex and they need to be evaluated more frequently. In practice, this means that XACML policy evaluation is critical: Axiomatics AB (a vendor) claims that XACML is the predominant language for expressing attribute-based access control (ABAC) policies and there are many ways to express those access controls in XACML form. Furthermore, other characteristics of the access control environment can affect its performance: the choice of PDP, size of server, mix and frequency of requests, among other factors, can be significant.

Performance prediction based on experiments can show promise, but there are some gaps that need to be filled.

1. It is not yet possible to compare different languages (hence PDPs) that can be used to represent access control policies. For example, it is possible to compare two XACML 2.0 PDPs given common XACML 2.0 policies, or even two versions of XACML 2.0 policies given the same XACML 2.0 PDP. However, it is not possible to compare XACML 2.0 policies to those written in another language (even a related dialect such as XACML 3.0) without significant, typically manual, extra effort.

2. More practically, the ATLAS components were implemented as sets of command line scripts that need to be manually configured and executed to perform a set of experiments. We have identified a need to make the specification of policies easier, so that more users can use ATLAS effectively. Furthermore, even though the policy representation has been greatly simplified, the tooling (simple, syntactically-unaware text editors) to support editing these policies is quite primitive.

These research gaps give rise to the Research Questions presented in Section 1.3.

Chapter 3

ATLAS Graphical User Interface

3.1 Introduction

Butler and Jennings (2015) introduced a system called `DomainManager`. The system takes two inputs; a set of properties files to configure a static domain model, and a template policy set. `DomainManager` produces a set of policies and requests in different size and complexities as its output. The output from `DomainManager` then becomes input to another system, introduced by Butler et al. (2010) and extended in (Butler and Jennings, 2015), called STACS. STACS is a testbed which is designed to evaluate the performance of requests in different PDPs. This system generates the statistics of the performance of requests under different circumstances, i.e., measuring the service time per policy-request combination to collect the service time of response evaluation. The generated output is then processed to get results in the required format. These systems can be highly productive when used together to generate *related sets* of policies and requests to understand the performance of request evaluation. Before DomainManager, policies were written individually, typically in XML format (Figure 3.1a), which contained identical information repeatedly and hence they were prone to errors.

Butler and Jennings (2015) introduced a new syntax (Figure 3.1b) for writing sets of template policies, which is relatively easier to write with less errors. However there are still some major complexities in configuring these systems to run properly. Firstly, the static domain model has to be initialized using a set of property files that the user has to write manually. These property files contain attributes and their values that are used to setup the static domain model for the first time. The property files look simple but all files have specific formats and administrators can easily make logical errors when writing these files manually. Secondly, DomainManager is provided with template policies that administrators have to write manually in the new syntax (Figure 3.1b) introduced by (Butler and Jennings, 2015). Thus the template policies can also contain logical errors. Such logical errors in the template policies can be subtle and so can be very difficult to identify and correct.

To handle these issues, we introduce a new component that can bridge the existing independent systems of ATLAS and eliminate the hassle of configuring these systems manually. This new component should help users to initialize the static DomainModel by capturing information about the attributes and their values and also define a way that can generate logical error-free template policies by simply selecting different attributes and combining them to make rules and rule groups. In order to achieve all these features, we built an ATLAS GUI that helps users

1. to configure the static DomainModel by providing property files written by GUI,
2. to write template policies with less effort and more accuracy by getting output from DomainManager.

This flow is represented in Fig. 1.3.

In this chapter, we discuss the design, implementation and usage of the ATLAS GUI. The requirements are identified in Section 3.2. One of the critical requirements was the need to develop a flexible data model (Section 3.3) to enable the GUI to store user choice data before that data is exported as property files. Section 3.4 describes the framework and other technology choices that were made when developing the ATLAS GUI, especially what features guided that choice and how the technology choices influenced development. Section 3.5 details the scope of the ATLAS GUI. Figure 3.8 distinguishes between the `Admin` role (Admins are responsible for defining the static model) and the `Employee` role (Employees are responsible for writing template policies). The various screens used in each scenario are presented as screenshots, supported by UML sequence diagrams that outline the interactions between the ATLAS GUI and its users.

```
<Resource>
   <ResourceMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">
              Asset.type = Marketing plan
      </AttributeValue>
      <ResourceAttributeDesignator
       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
       DataType="http://www.w3.org/2001/XMLSchema#string"
      />
   </ResourceMatch>
</Resource>
```

(a) XML Format

```
RESOURCE.0=Asset.type,equals,marketing plan
```

(b) DomainManager Format

Fig. 3.1 Single Element of Template Policy

3.2 Requirement

To understand the working of ATLAS system, we identified the main users for ATLAS GUI and their associated use-cases. ATLAS GUI is designed to perform two major tasks which are:

1. Initialize DomainModel and DomainManager
   - Initialize DomainModel with existing attributes and values
   - Manage properties files for DomainManager
2. Manage template policies set for DomainManager

We identified that ATLAS GUI has two major users who are responsible for doing the two operations above, in order to integrate existing isolated components of ATLAS system. The first user is an "Administrator" who is responsible for initializing the DomainModel and DomainManager, and the second user is the "policy author" who manages the template policies set for DomainManager. The "Administrator" initializes the host organization's domain model by providing static attribute-value pairs to be used in specifying asset, member (of organisation) and action entities. After providing this "vocabulary", the Administrator specifies how these attribute-value pairs are to be combined to generate asset, member and action instances. This static model data is then available for use by template policy authors using the ATLAS GUI.

The template policies are specified in the syntax described in (Butler and Jennings, 2015). The set of *template* policies and other configuration data are processed by DomainManager to produce sets of *full* (actual) policies and requests. These policies and requests are then provided to STACS for performance measurement experiments. Given the performance results, the "policy author" can modify template policies, where needed, to improve the performance of evaluation of requests. This is the overall working of ATLAS. The ATLAS GUI helps policy authors to specify the template policies in a way that is less prone to specification errors, which has the potential to improve the reliability of the results from ATLAS.

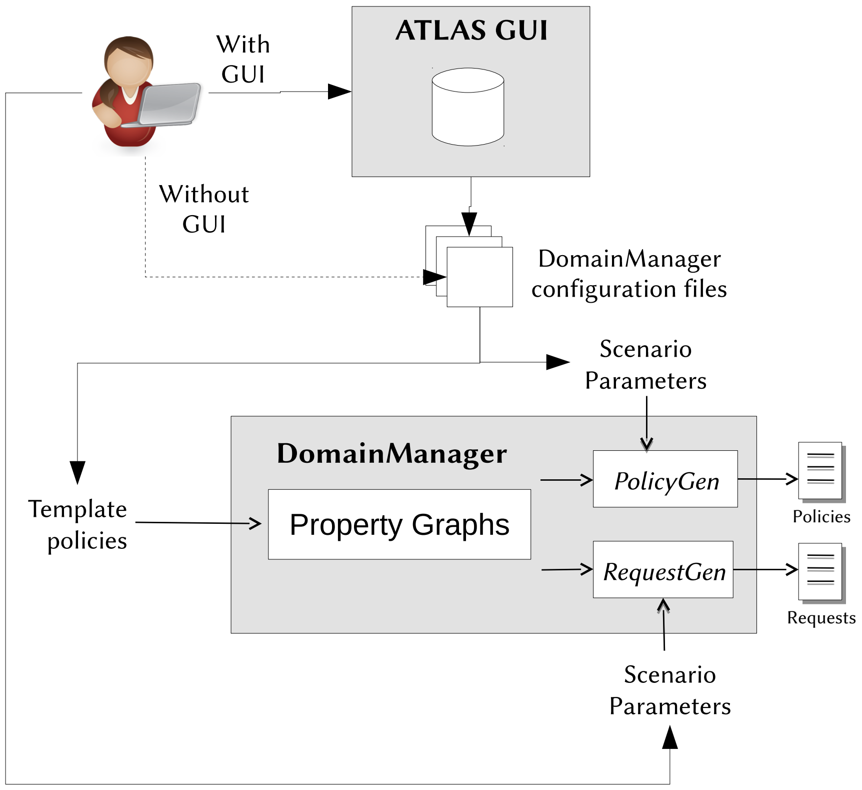
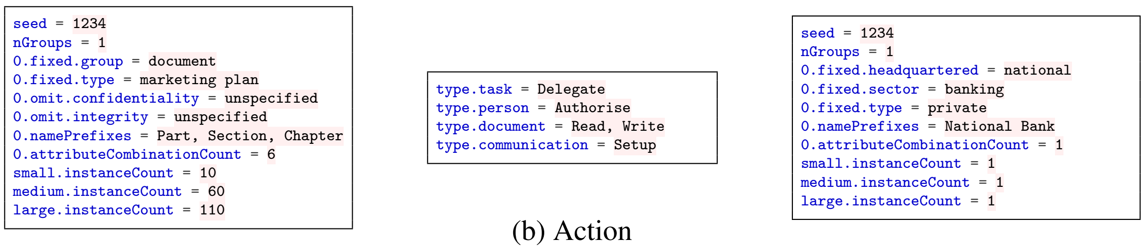


Fig. 3.2 Relationship between the ATLAS GUI and DomainManager.

The relationship between the ATLAS GUI and the DomainManager component of ATLAS is shown in Figure 3.2. Previously, the security administrator wrote the configuration files directly, following the dashed path. Now that the GUI is available, ATLAS users can continue to use this approach, or can choose the GUI component to write the template policies, static data (such as the attributes of persons and resources of interest to the access control system) and policy generation configuration parameters. Note that the template policies provided by ATLAS users form the basis of suites of policies, with characteristics such as size and complexity of the policy suites specified in the policy generation configuration parameters. All these properties are written in configuration files, either manually or via the GUI, for use by DomainManager. These supporting properties files were written manually in past using the format in (Butler and Jennings, 2015). But now with the help of ATLAS GUI, an “Administrator” can write these property files by just selecting combinations of different attributes and values required to produce property files. By doing this, ATLAS GUI minimizes the chance of the user making logical errors because it adds necessary but not sufficient constraints to the policy editing process.

Figure 3.2 also indicates that the user still needs to configure, in a manual fashion, the request generation part of DomainManager. In principle, the ATLAS GUI could be extended to support this form of editing too. Support for new flows would need to be added, and the data model would also need to be extended, but this would be future work.

Fig 3.3a and 3.3c look very similar and have a simple property file format, but logical errors are still possible. For example, as we can see in fig 3.3a, a group is formed by combining different attributes and values with their properties and group is grouped by prefix digit starting from 0 to N-1 groups. While writing these properties files manually one can easily mix two groups by starting with the wrong group number, and these errors are very difficult to track until you run the complete evaluation test from start to end which is very time consuming. The second issue that we identify in writing these property files is that files have their own pattern, see fig 3.3a, 3.3b and 3.3c.



```
seed = 1234
nGroups = 1
0.fixed.group = document
0.fixed.type = marketing plan
0.omit.confidentiality = unspecified
0.omit.integrity = unspecified
0.namePrefixes = Part, Section, Chapter
0.attributeCombinationCount = 6
small.instanceCount = 10
medium.instanceCount = 60
large.instanceCount = 110
```

(a) Asset

```
type.task = Delegate
type.person = Authorise
type.document = Read, Write
type.communication = Setup
```

(b) Action

```
seed = 1234
nGroups = 1
0.fixed.headquartered = national
0.fixed.sector = banking
0.fixed.type = private
0.namePrefixes = National Bank
0.attributeCombinationCount = 1
small.instanceCount = 1
medium.instanceCount = 1
large.instanceCount = 1
```

(c) Organisation

Fig. 3.3 Group Properties Files

Fig 3.3a and Fig 3.3c are specify groups of assets and groups of organisations. Both property files have a similar pattern: they look very similar to each other but they have different attributes according to their nature. Also, the pattern we see in Fig 3.3b is totally different from that in other property files. By observation, we saw that Admin users can easily make mistakes in writing different files because of their different formats. Even more commonly, users can misspell the names and/or values of entities, which can result in more errors.

Our observations also suggest that the same issues (of misunderstanding the syntax and/or mistyping identifiers) can occur when writing the template policies. Butler and Jennings (2015) introduced a terse syntax (see Fig. 3.1b for an example) that made policies easier for humans to read without being distracted with lots of detail that detracts from understanding the policy's intention. However, errors can still occur. For example, Fig 3.4a contains only one resource condition and looks very easy to write. Fig 3.4b shows a slightly more complex template policy fragment. If the second condition had a misspelling such as "Corporate Strategy" (note the presence of upper case letters "C" and "S") a condition might not match as intended. Another potential error might occur if the second line was also labeled as `RESOURCE.0` rather than `RESOURCE.1` as intended. This could happen as a result of an oversight when copying and pasting from the preceding line. However, if such an attribute is repeated, this is a *semantic* signal to DomainManager that the underlying clauses are "ANDed" together, rather than being "ORed" together as was the intention of the policy author.

Template policies combine subject(s), resource(s), action(s) and a decision into rule(s) that are then combined with other rules using combining algorithms to form rule groups. Rule groups can also be combined into more complex rule groups. This recursive structure can be arbitrarily complex, and simple text editors do not help the template policy author to interpret the template policy while it is being written. Thus errors can be present, and these errors do not become apparent until *after* a performance test has completed. Therefore, depending on the nature of the test, the errors might not be seen until after many hours. Therefore, to improve the efficiency of the testing process, it is vital to take any reasonable steps to minimise the need for rework and/or faulty performance results. The use of ATLAS GUI is one such step.

```
RESOURCE.0=Asset.type,equals,marketing plan
```

(a) Single Resource

```
RESOURCE.0=Asset.type,equals,marketing plan
RESOURCE.1=Asset.type,equals,corporate strategy
```

(b) Multiple Resources

Fig. 3.4 Template Policy

To minimise logical and typing errors, ATLAS GUI enables users to select values of attributes from *filtered* dropdown lists rather than by entering free text into unvalidated fields. To achieve this, attribute-value pairs are stored as "static" data by the Administrator in a centralized database which can later be read by the policy author to write template policies. This database is discussed in section 3.3.

3.3 Data Design Model

Butler and Jennings (2015) introduced an extremely flexible static model of Organisations, Members, Assets, etc. The model is structured as a graph, with nodes and edges, so new attributes and even new attribute types can be added easily. However, the source data typically does not take this form. Therefore DomainManager accepts an intermediate representation (a Domain Specific Language (DSL)) which “flattens” most of the relationships (equivalent to denormalizing a relational model). This representation is more suitable for textual input. Therefore *users* (with the “Administrator” role described above) are responsible for serializing the static model structures and data as property files. This serialisation process is complex so the cognitive load for users having only basic tools (such as syntax-unaware text editors) is very high. Therefore, we designed a *normalized* relational model to support the GUI to make adding this data easier. The GUI takes over responsible for denormalizing the data provided by its users and writing it, as an instance of the DSL introduced in (Butler and Jennings, 2015), to property files that provide suitable input to DomainManager.

Every system that is used to process information has to have some sort of structure that stores data in it for long term use. Similarly ATLAS GUI performs operations on data in term of generating properties files and template policies. To generate these files and policies, it needs some data in form of attribute values which has to be stored somewhere in system so that it can be accessed while writing properties files and template policies. Not only because of this we need data storage, but also we need to make sure that the values of attributes and rules that we are making for template policies remain consistent and easy to modify if there comes a need to modify it. So, these are the two main reasons we have re-designed existing data models for ATLAS. The new database not only stores values and other useful information but it also restructures previous data structures which contained extra tables.

Initially we designed a data model in which there were a different tables to store similar data. Within the previous data model, if we need to add a new group for some organisation we need to add a new table and change the major data model. After tweaking and redesigning our data model to meet our requirements we have the following model as shown in Fig 3.5 to fulfill most of our data storage and retrieval requirements for the GUI.

Now we will describe what was our first design and why there was a need to redesign that model. First of all we need to understand the basic structure of our system. ATLAS GUI is web based application which allows multiple organisations to make their profile and manage their template policies and evaluate their performance using ATLAS system. Each organisation has its own structure, some organisations may belong to bank sector, some belong to education, etc. Organisations can have different groups and each group can have different types of users who are allowed to perform different roles. For example, if the organisation is from the banking sector then there must be a user which deals with front desk known as front desk officer or cashier, his/her role will be different to a person who is working as teacher in some education sector organisation. So, every role/user will have their own attributes according to their affiliated domain/organisation.

Before the final version of our data model, we were creating separate table for each attribute of each user, so we can imagine the number of tables required to hold values for even one organisation and it was impossible to cover all attributes for all organisation beforehand. Also, the previous version of our data model can be expanded only vertically which means that we have to add new columns or tables to support new attributes, But our latest data model is able to expand horizontally, if we need to add new attributes of some new organisation, we typically just need to add more rows to existing tables. Consequently it does not require a schema change. Other benefits we get from the new design is that it is normalized; an attribute and its values can be modified within the GUI without repetition and/or errors.

Regarding the current data model for ATLAS GUI, Fig. 3.5 is a Unified Modeling Language (UML) diagram of the ATLAS GUI[1], where each class is backed by a table of the same name in a normalized relational database. The entry point to the data model is that data is captured and stored by Organisation. Organisation is a company which needs to write and evaluate their company policies using our ATLAS System. There is an "organisation" table that stores very basic information and allows its "Administrator" to create other users for that company and stores user related information in table named User which is again linked with Organisation. Following are the main tables, highlighted in fig 3.5 with title "tables to discuss", focusing on how to save the data for policies and properties files, tables which are shown in fig 3.5 are supporting tables for managing policies and properties files data.

Fig 3.6 shows the main relationships that must understand by the user in order to have clear idea about our system. In this fig 3.6 organisation main entities are saved in graph_entity table and values of all these entities are saved in graph_entity _value. This is done by the user who has administrator rights and following this structure we ensured that it will help policy editor to minimize the human errors while writing the policies using ATLAS GUI. Further in this fig 3.6 policy editor uses the values from graph_entity _value in dropdown options for subject, resources and actions to form a rule. This simple but very powerful structure reduces

[1]Note that the diagram has been "summarised" in the sense that the class fields are not displayed. Removing these details is intended to make the overall structure easier to see.

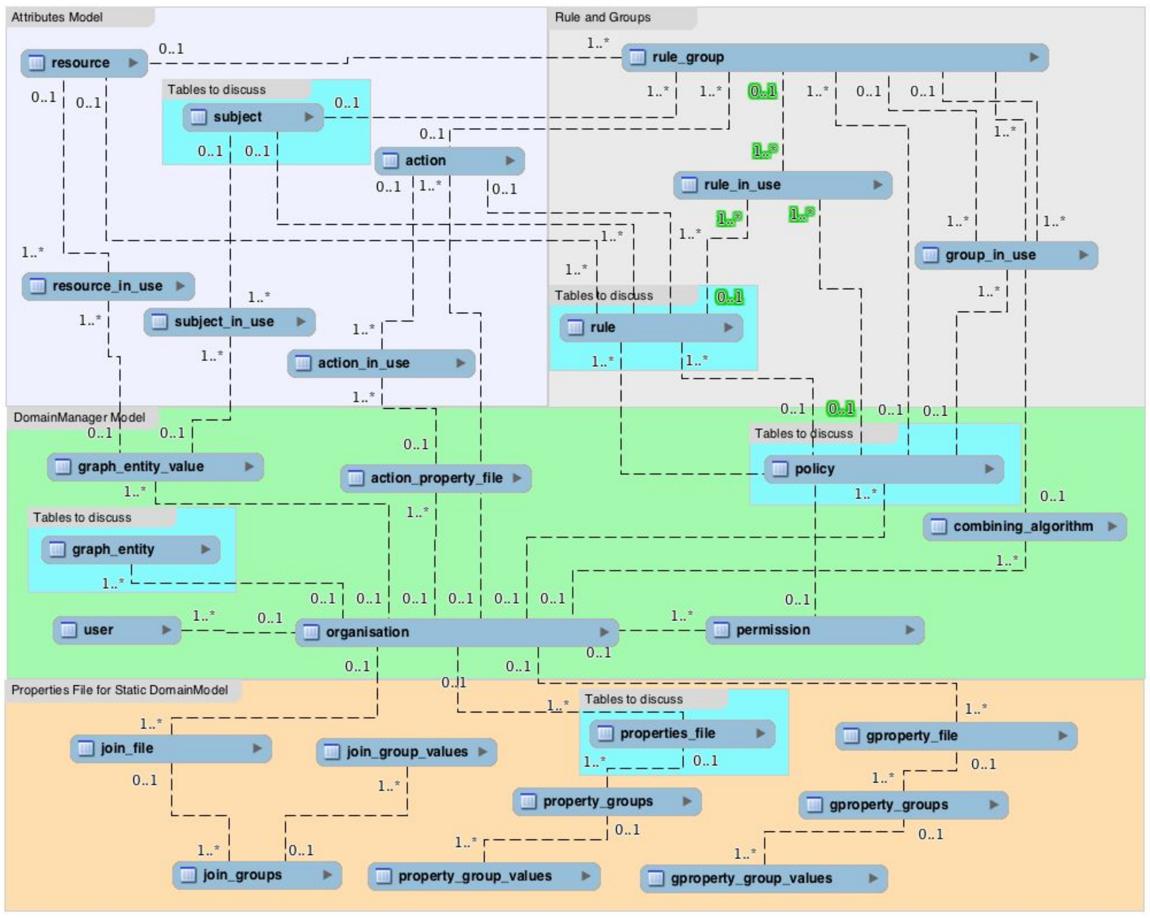


Fig. 3.5 UML Cluster Diagram

many problems and it is generic enough to incorporate different organisation structures using these two main tables because they are expanded horizontally for columns rather than vertical expansions to add new columns in future for organisation policies.

1. graph_entity
2. policy
3. rule
4. subject
5. properties_file

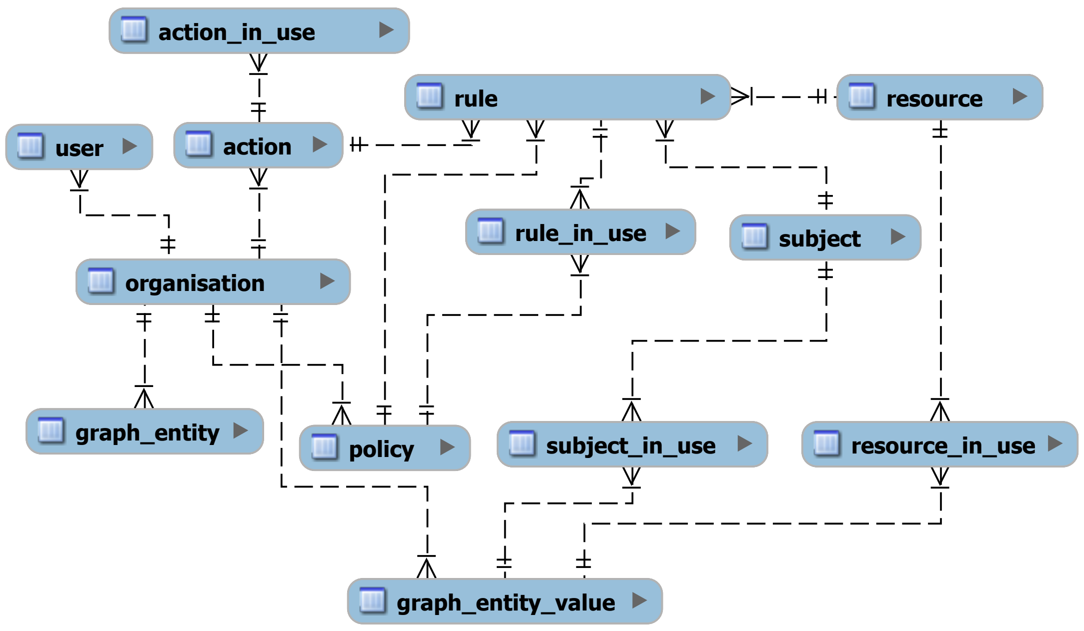


Fig. 3.6 Core tables

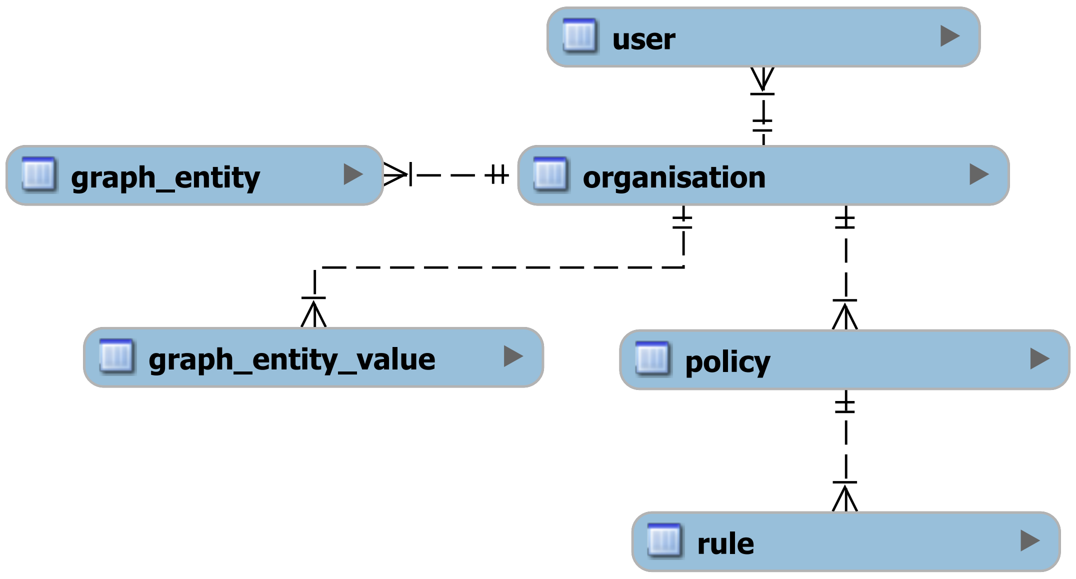


Fig. 3.7 Graph Entity relationship

3.3.1 Graph_Entity

Fig 3.7 represent the most important relationships among organisation class and other classes as Graph_entity table is core of organisation. It holds the structure for organisation, which includes hierarchy of organisation, roles of different users. Question arise that how it can hold different hierarchy for different organisations in a same table. So, here we redesigned our previous data model and introduced graph_entity table. Graph_entity have four columns i.e; “main_type”, “sub_type”, “value” and “organisation_id”. Graph_entity actually represents a graph or tree structure of any organisation up to two level of hierarchy. “main_type” stores parent of “sub_type” and values of “sub_type” is stored in “value” column. Organisation hierarchy starts with a “parent” node which contains the 1st level of organisation hierarchy. Child or sub-values of “sub_value” is defined in “value” column of graph_entity. Leaf node values of this hierarchy is stored in a table named “Graph_Entity_Value”. For example, values such as “low”,“medium”,“high” for AssetConfidentiality level are stored in “Graph_Entity_Value”. All dropdown values will be fetched from this table to populate a dropdown list that will be used to make rules and properties files.

3.3.2 Policy

ATLAS system is all about managing and evaluating policies and request, so this table holds the information for policy that user writes using ATLAS GUI module. The Policy table will hold very basic information including “policy name” its “description” and “organisation” that it belongs to. Policy is a combination of rules, groups, and groups of rules and groups combined together with combining algorithm to make a policy. Other details for policies are stored in different tables related to Policy. In Section 3.3.3 we will discuss how we are storing rules for a given policy.

3.3.3 Rule

Rule is a key component of any policy, a rule is composed of a name, its description (which is optional), reference to subject, resource, action, permission and a policy reference that contain this rule. Rule does not contain values for subject, action and resource, it contains the reference to all these values, because one rule can contain more than one subject, action and resource, so the values rule is using as subject are stored in a table named Subject, “subject_in_”, resources that rule is using are stored in Resource, “resource_in_” and similarly for actions that are used in rules are saved in a table named Action and “action_in_” used. We will discuss the structure of the Subject and “subject_in_use” Section 3.3.4 to give an idea of how things are stored for Rule in our data model.

3.3.4 Subject

If a rule contains at least one subject, then the detail for that subject will be stored in the Subject table. To create template policies, Subject has two types; one is “member” and the other is “organisation”. Each subject type has its own characteristics, and we can use these characteristics in rules as required. To explain the structure lets take an example of “member” as subject and we are using the member’s “function” and “level” in our rule. So, we will add only the values of “function” and “level” to “subject_in_use” table with reference of subject along with this. Table “subject_in_use” contains the reference to values that the “Administrator” added in 3.3.1, and a reference to the subject that is used in the rule. In similar fashion, resources and actions for rules are processed in the data model.

3.3.5 Properties_File

We are not only writing template policies using ATLAS GUI, but also ATLAS GUI enables us to write static model property files. For this purpose our data model contains three tables which are; “properties_file” itself, “property_group” and “property_group_values”. “property_file” contains the file name, property file type which identifies the type of file and organisation id to which that property file belongs. The entry in “property_file” allows the user to verify whether this type of property file exists or not. The property file is actually a list of groups used to create the instances of sample policies out of template policies by filling the template policies. “property_group” table stores each attribute of each group that is being used in the property file. It contains the number of instances for small, medium and large size policies that uses prefixes as mentioned with it in the entry. And the type with its value, which we have to include or omit from being used in group is stored in the “property_group_values” table. So, all these three tables enable users to save property files used to help DomainManager creating suites of policies, with each suite of policies having its own characteristics, from a common set of template policies. In the same manner we create group properties file, that is used to support DomainManager to combine attributes together to form an attribute group.

In Section 3.4 we will discuss why we have chosen play framework as our framework to work with and what are the pros and cons for this framework.

3.4 Framework and Technology

The first consideration was the selection of a suitable framework and technology for subsequent development. In modern frameworks, it is often possible to develop full applications based on a single file containing all configuration settings, for example application styling, data-layer controllers and front end coding could all be specified in a single configuration file. This will not effect the work flow, but ongoing maintenance can be difficult especially if numerous small changes are needed. Data persistence also needs to be considered.

We decided to build a web application instead of a desktop application, because ATLAS GUI users might wish to access their policies from anywhere.

We had many different programming languages as our option to start with, for example Javascript, Java and python. However the main language used by ATLAS components is Java, so we decided to stay with a JVM-based solution.

The next challenge was the selection of framework to build the ATLAS Web GUI. For this we compared many Java-based frameworks, especially the *Play Framework* and Spring. Both frameworks have their own advantages and disadvantages. Play is a light-weight, beginner-friendly web framework that includes a JavaScript library which is easy to learn and supports a component-oriented approach to developing web based applications. Play also uses Akka for concurrency, so it is incredibly scalable with high throughput. When coupled with the native integration for Akka actors, Play suits the development of high performance asynchronous applications ready for Big Data needs. Play also comes with the option to scaffold applications. It has a comprehensive ecosystem designed to increase developer productivity and shorten development times. On the other side Spring has an even more extensive ecosystem: the Spring Framework is well established and it benefits from tools like Roo and Spring Tool Suite etc. All Spring Maven dependencies are available in a public Maven repository. There are also 3rd-party solutions for Spring, such as MyEclipse which includes scaffolding capability for Spring Model View Controller (MVC) (Sukaj, 2015). However the Spring MVC architecture has a lot of layers and abstractions which can be hard to debug if problems arise. It is also highly dependant on the Spring core. It is a mature framework that has many ways to extend and configure it and this actually makes it more complex. By contrast, the Play framework is newer, less comprehensive but also easier to use. Since it is sufficient for ATLAS GUI development, we chose to use the Play framework.

According to research some of the main features of Play Framework that are worth mentioning are

Enhancement in productivity Developers can make a change, refresh the page, see the change. Hot reload for all Java code, templates, config changes, etc., allows developers to iterate much faster. This is available in many dynamic languages, but is more developed in Play than in other Java web frameworks.

Open Source framework Play is open source. So, we can see how everything works and extend it as needed. The Play Framework community is very active.

Support Typesafe provide commercial support.

Error handling support In dev mode, Play shows, for both compile and runtime errors, a) the error message, b) the file path, c) the line number, and d) the relevant code snippet all presented right in the browser. All this context makes the error easier to find and hopefully to fix. There is no need to dig through multiple log files (as in Tomcat) and there are far fewer incomprehensible, gigantic stacktraces (as in Spring).

Flexible Just about everything in Play is pluggable, configurable, and customizable.

Modern stack Play is an MVC stack on top of Netty and Akka and has built-in support for most tasks a developer needs in a modern web framework: Representational State Transfer (REST), JSON/XML handling, non-blocking I/O, WebSockets, asset compilation (CoffeeScript, less), ORM, NoSQL support, and so on.

We used the Eclipse Integrated Development Environment (IDE) as our tool to create the ATLAS GUI, because it has support to build Play framework applications. We used python2.7 to evaluate results from performance values stored by STACS in the results database and we have used MySQL 5.6.16 as our database for the ATLAS GUI data model. Section 3.5 shows what the ATLAS GUI looks like and how it works.

3.5 GUI Layouts

Recall that the ATLAS GUI has two major users; the role and responsibilities of these two users have been discuss in section 3.2 in detail. I Section 3.5.1 and Section 3.5.2 we will discuss using screen-shots and interaction diagrams for admin and policy author roles and responsibilities in detail respectively. We will describe how we mapped roles and responsibilities of both user in our ATLAS GUI to simplify writing policies and managing them and how to integrate the components of ATLAS using GUI.

ATLAS GUI is designed in a block pattern using different colors to represent different actions and features. We used this color grouping methodology for ease of use and to train users to perform actions quicker and accurately. We used the Bootstrap v3.3.0 (Bootstrap Team, 2015) *reactive* framework to enable mobile or tablet customized views for our ATLAS GUI. Bootstrap has very bright and informative button color schemes which is used in different contexts. For example, we used red colored buttons to delete some values from our system, because deleting some value accidentally might cause some serious after affects. We used green colored buttons when making progress, e.g., by saving some value, starting a new procedure or proceeding to the next step. The GUI major layouts are self-descriptive and consistent, so this leads the user to next step as seen in the following layouts.

In order to understand the needs from admin and employee perspective we have designed a diagram 3.8 which will represent the actions that both users will perform using GUI.

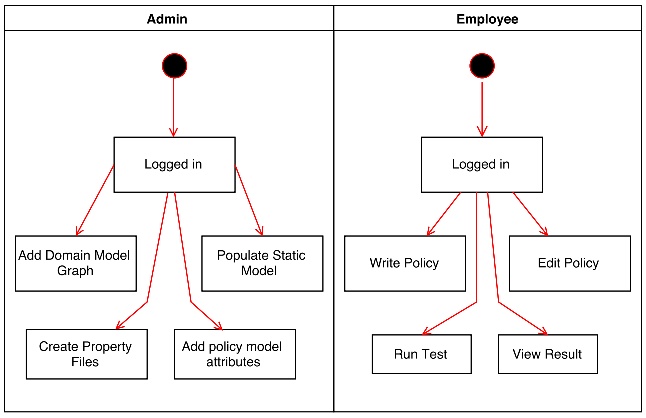


Fig. 3.8 Admin and Employee roles

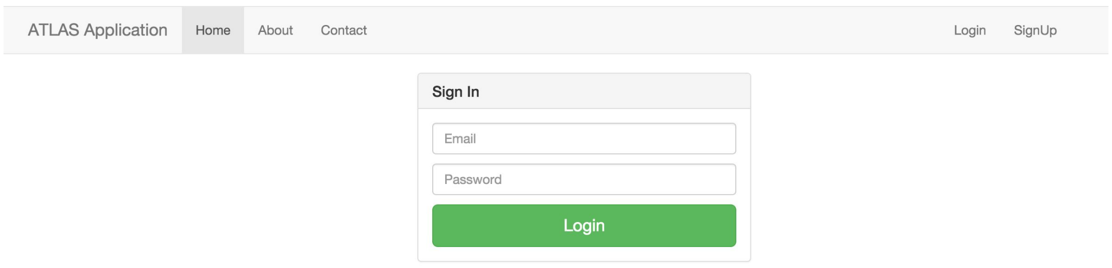


Fig. 3.9 login

3.5.1 Admin Role

First user of ATLAS GUI is admin, and admin has four major roles to perform using ATLAS GUI which are;

1. Manage DomainModel
2. Manage StaticModel
3. Manage PolicyModel
4. Generate Properties Files

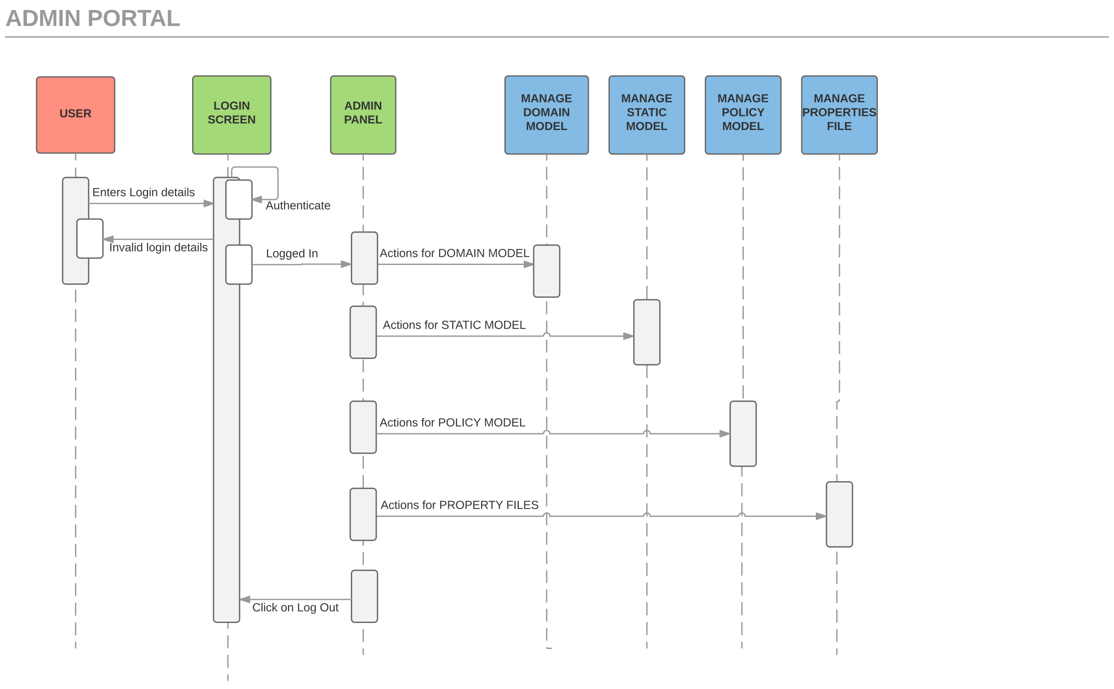


Fig. 3.10 Sequence diagram of actions performed by admin

Fig 3.10 represents the main flow of admin portal, how admin will interact with the ATLAS GUI to perform these operations. The Login Screen seen in fig 3.9 is our landing page, that is used by both admin and employee to log in. If the user enters valid login credentials then according to his/her role he/she will redirect to admin or policy author portal. In this section we are describing admin portal, so we assumed that admin logged in with valid credentials to perform various actions which can be seen in fig 3.11

Fig 3.11 is an admin control panel, which is used to initialize isolated systems of ATLAS application, that were initialized manually before ATLAS GUI. To initialize these systems,

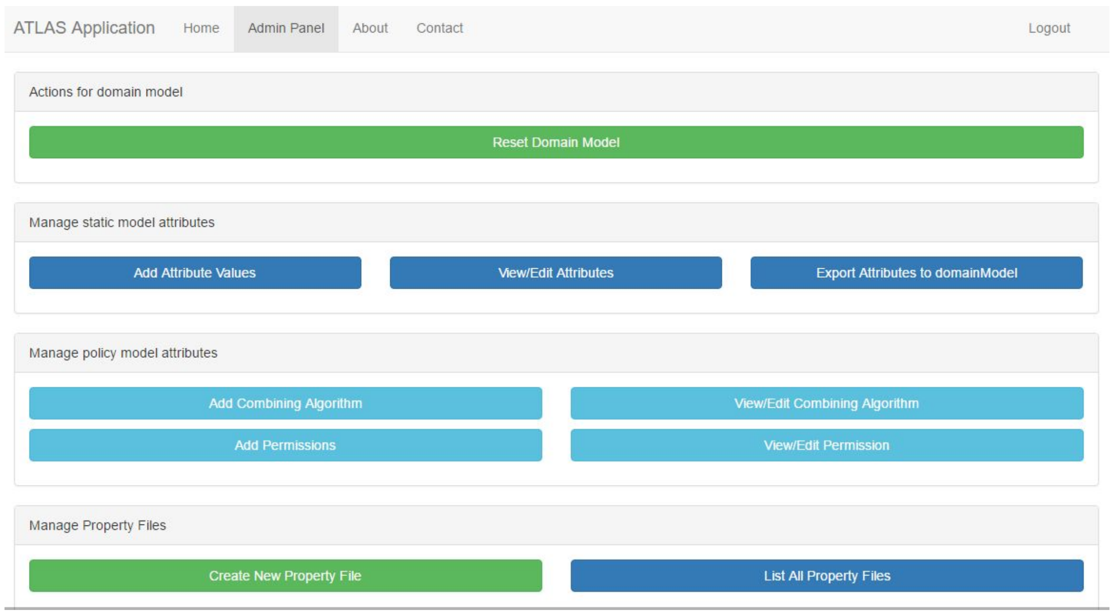


Fig. 3.11 Screen shot of admin control panel to perform various admin tasks

there are several operations we need to perform, as we can see in Fig 3.11 there are four sections in this layout and each section manages one system and configures it as required. Section 3.5.1.1 shows how to manage the domain model and initialize it with the company hierarchy and basic attributes of organisations, members and resources.

3.5.1.1 Manage DomainModel

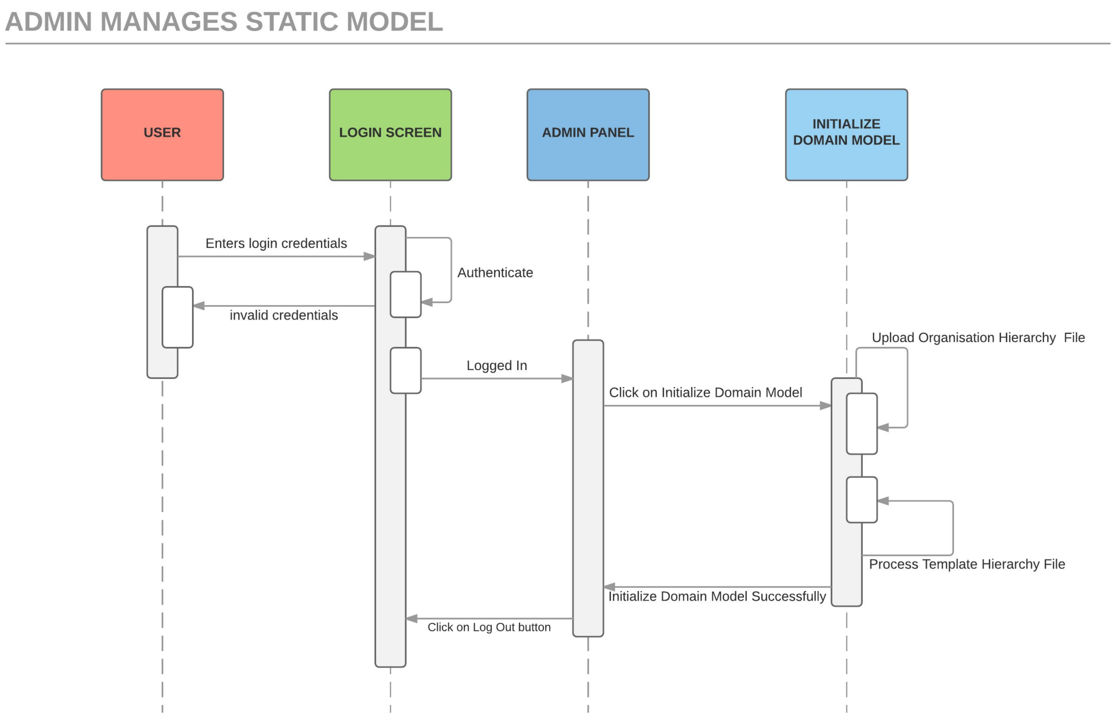

ADMIN MANAGES STATIC MODEL
USER
LOGIN SCREEN
ADMIN PANEL
INITIALIZE DOMAIN MODEL
Enters login credentials
Authenticate
invalid credentials
Logged In
Upload Organisation Hierarchy File
Click on Initialize Domain Model
Process Template Hierarchy File
Initialize Domain Model Successfully
Click on Log Out button

Fig. 3.12 Sequence Diagram to initialize DomainModel

Fig 3.12 represents the flow that the user has to follow to initialize DomainModel with company's hierarchy and as we can see the User first needs to log in with admin credentials and if login details are correct will be redirected to the admin panel, to click on initialize or reset DomainModel button which will take admin to next screen as shown in Fig 3.13.

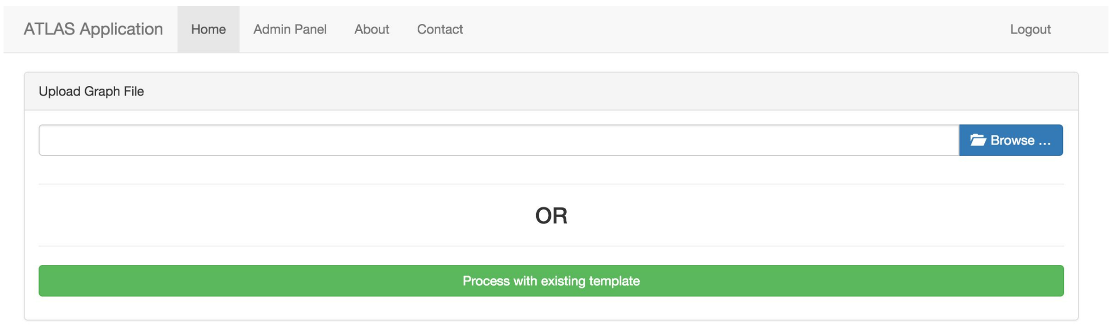


Fig. 3.13 Screen shot of ATLAS GUI to initialize structure of company

Fig 3.13 is designed to initialize the basic hierarchy of a company, i.e, what are the attributes in terms of subjects, resources and actions and it defines the type of subjects, resources and actions up to two levels. Every company must have some hierarchy defined for its company. As an example, subjects can either be a “member” or “organisation”. Next, the “member” can have attribute “role”, “function”, and “level”. This kind of hierarchy file is uploaded using the layout shown. The second option in this layout is to use ATLAS default hierarchy that contains very common attributes for most of the companies. Once the DomainModel is initialized with basic company hierarchy the next task for admin is to set the values for hierarchy attributes. These values will then be used to make the template policies, these template policies can be written by both policy authors and administrators. Fig 3.15 is designed to add attribute values and modify it, if there is a need to modify.

3.5.1.2 Manage StaticModel

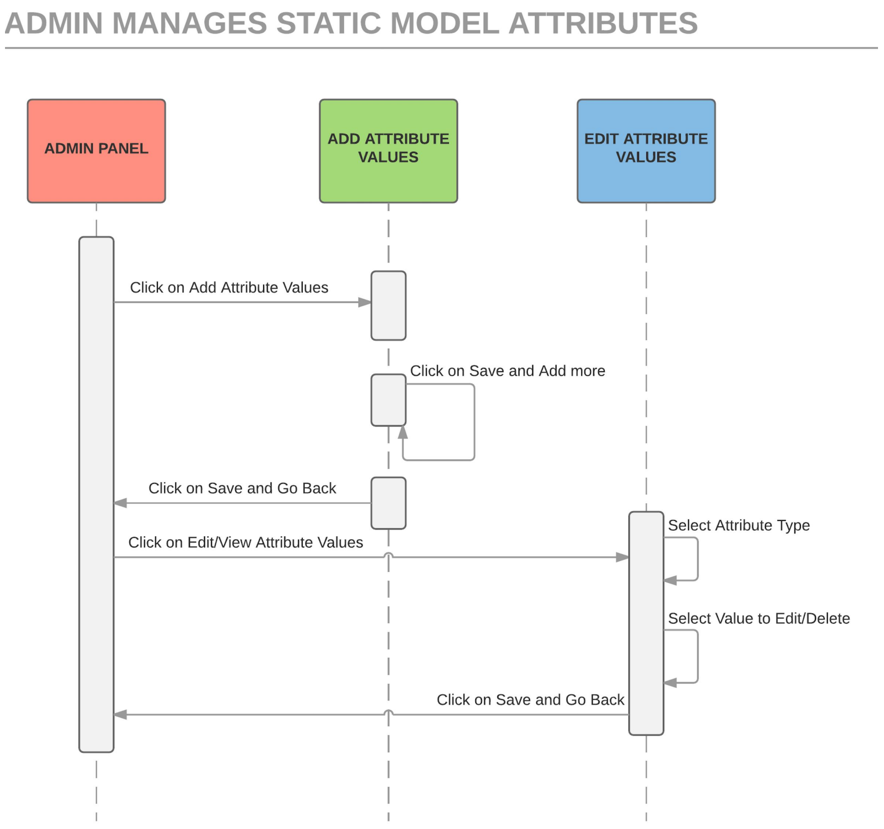
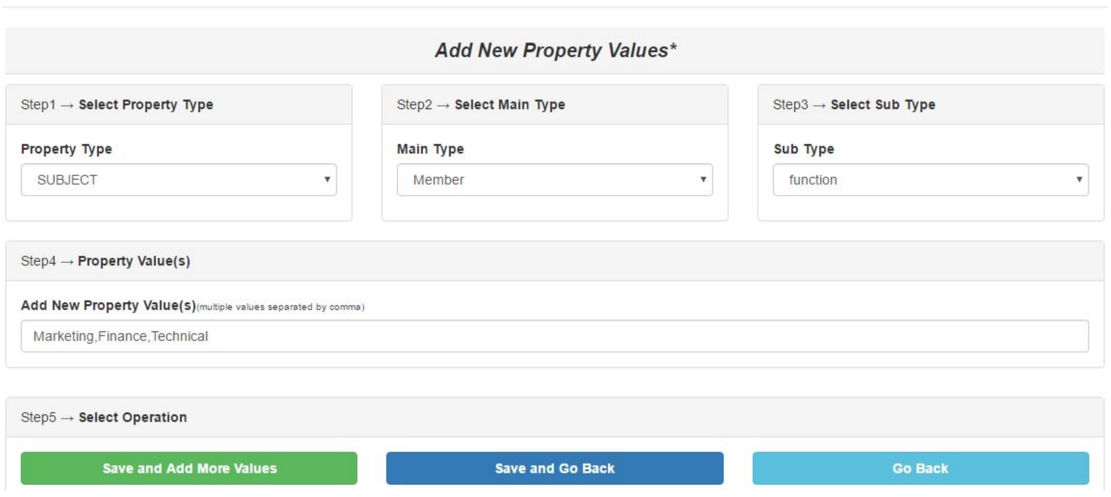

ADMIN MANAGES STATIC MODEL ATTRIBUTES
ADMIN PANEL
ADD ATTRIBUTE VALUES
EDIT ATTRIBUTE VALUES
Click on Add Attribute Values
Click on Save and Add more
Click on Save and Go Back
Click on Edit/View Attribute Values
Select Attribute Type
Select Value to Edit/Delete
Click on Save and Go Back

Fig. 3.14 Sequence diagram to add or modify attributes in StaticModel

To add new values for attributes of company hierarchy we will use layout as shown in Fig 3.15, as we already described that company hierarchy has two levels and values of leaf node for hierarchy is then added in to ATLAS system using above layout. In above layout, we can see that if users need to add any values to the system, they select “Property Type” i.e,


*Add New Property Values**

Step1 → **Select Property Type**

Property Type: SUBJECT

Step2 → **Select Main Type**

Main Type: Member

Step3 → **Select Sub Type**

Sub Type: function

Step4 → **Property Value(s)**

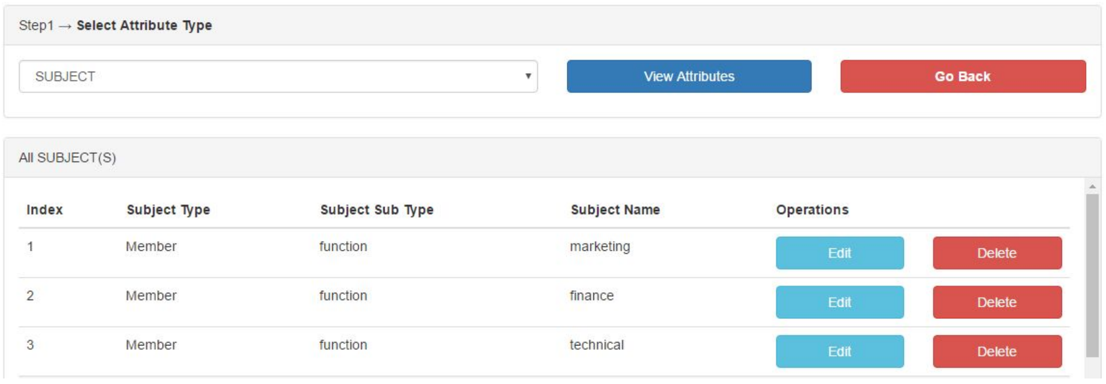
Add New Property Value(s)(multiple values separated by comma): Marketing,Finance,Technical

Step5 → **Select Operation**

Fig. 3.15 Screen shot of ATLAS GUI to add new attributes in StaticModel

the first level in company hierarchy, either to add some value related to subject or related to resource etc. Values will be selected using the “Property Type” drop down. We have selected “SUBJECT" as our “Property Type" value. After Step 1, the user will select “Main Type” of “Property Type” in step 2, i.e, if “SUBJECT” is the “Property Type”, in step 2 the choice is between the types of “SUBJECT” which in our case can either be “Member” or “Organisation”. Step 3 is dependent on the value of step 2, the values will change accordingly for step 3 on the basis of step 2 selection. In step 3 the user will choose the actual attribute whose values we are going to add using this layout. If the user selects “Member” from the step 2 dropdown, the choice is between “role”,“function” or “level” of “Member” in step 3, because these are the possible attributes that were uploaded while initializing the DomainModel in Fig 3.13. So, we have chosen “Function" sub-type in step 3. Step 4 is to add values for “Function" as sub-type. We have added “Marketing,Finance,Technical" as three “Function" values in step 4 separated by comma as delimiter. This method saves the time of users to re-do all steps from step 1 to step 3 in order to add values one by one.

After entering all the attributes using this layout, users can either add new attributes for other types by pressing “save and add more values” button or can simply press the “save and go back” button in order to save current values and return to the admin main page. The third button on this layout helps the user to go back to the admin panel without saving any values for attributes.



Index	Subject Type	Subject Sub Type	Subject Name	Operations	
1	Member	function	marketing	Edit	Delete
2	Member	function	finance	Edit	Delete
3	Member	function	technical	Edit	Delete

Fig. 3.16 Screen shot of ATLAS GUI to modify StaticModel values

All systems are designed in a way that once something is saved in a system, it can be modified later on if there comes a need. In order to fulfill this requirement we have designed a layout as shown in fig 3.16 which enable users to select “Property Type” from a dropdown and shows its dependent attributes and values in tabular form, with two options; one is to edit values of a leaf node attribute or to delete that leaf node attribute from the database. We can see in fig 3.16 the new values of “Subject" type attribute that we have entered in the database using the layout shown in fig 3.15.

3.5.1.3 Manage PolicyModel

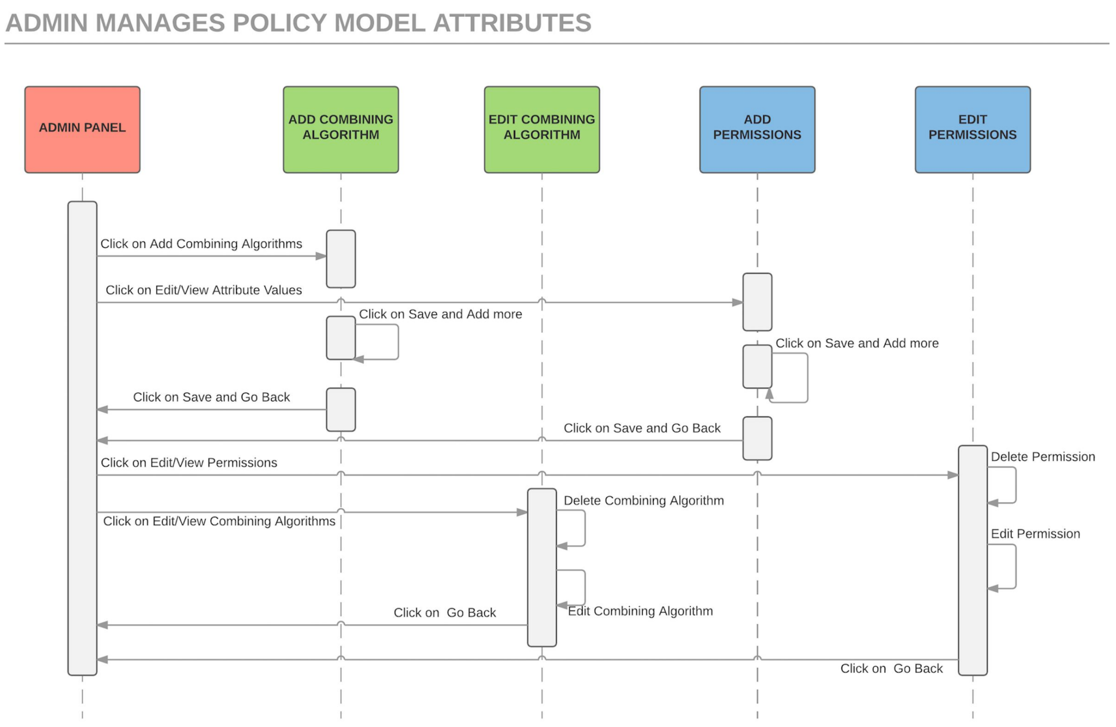
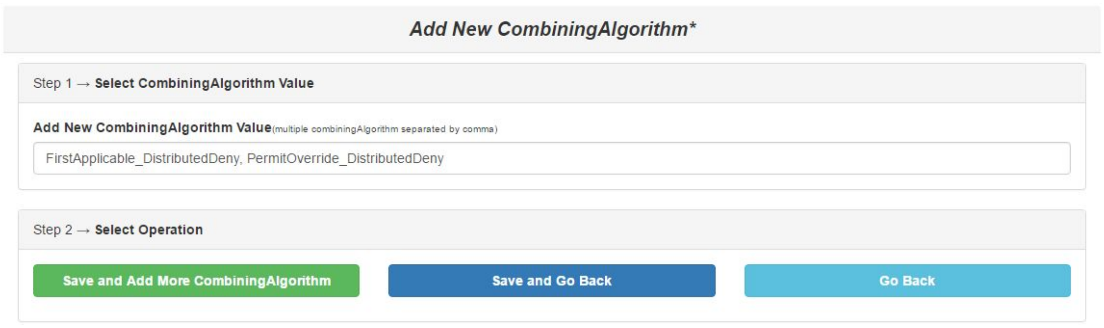

ADMIN MANAGES POLICY MODEL ATTRIBUTES
ADMIN PANEL
ADD COMBINING ALGORITHM
EDIT COMBINING ALGORITHM
ADD PERMISSIONS
EDIT PERMISSIONS
Click on Add Combining Algorithms
Click on Edit/View Attribute Values
Click on Save and Add more
Click on Save and Add more
Click on Save and Go Back
Click on Save and Go Back
Click on Edit/View Permissions
Delete Permission
Click on Edit/View Combining Algorithms
Delete Combining Algorithm
Edit Permission
Click on Go Back
Edit Combining Algorithm
Click on Go Back

Fig. 3.17 Sequence diagram to add or modify permissions and combining algorithms

Fig 3.17 shows the steps admin has to take, in order to add new combining algorithms, new permissions or to modify the existing combining algorithm or permissions. In the above sequence diagram, admin has to be logged in first in order to get access to the admin control panel and from there can perform operations to manage policyModel. Fig 3.18 and Fig 3.19 show how to use ATLAS GUI for adding new combining algorithms or new permissions respectively.


*Add New CombiningAlgorithm**

Step 1 → **Select CombiningAlgorithm Value**

Add New CombiningAlgorithm Value(multiple combiningAlgorithm separated by comma)

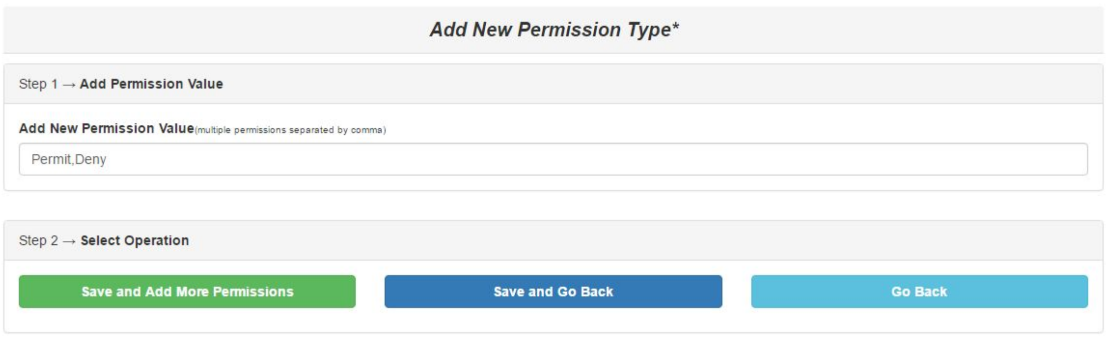
FirstApplicable_DistributedDeny, PermitOverride_DistributedDeny

Step 2 → **Select Operation**

Fig. 3.18 Screen shot of ATLAS GUI to add new combining algorithms

Policies are the combination of rules and rule groups. Each rule is made up of subject, resource and action to perform on that resource by that subject. Rules can form a group; combining algorithm bond rules together with particular semantics (Brossard, 2014).

To add these combining algorithm in the GUI, see fig 3.18. Admin just needs to write the name of combining algorithm in the given text field in step 1 and also can add more than one combining algorithm in step 1 using comma as delimiter to save time. In step 2, admin has three options for navigation. Option 1 is to “save and add more combining algorithm", option 2 is to just save current combining algorithms and go back and the last option is to go back to the admin panel without saving any combining algorithm. We have added “firstapplicable_distributeddeny" and “ permitoverride_distributeddeny" as two combining algorithm using comma as delimiter with the help of layout as shown in Fig 3.18.


*Add New Permission Type**

Step 1 → **Add Permission Value**

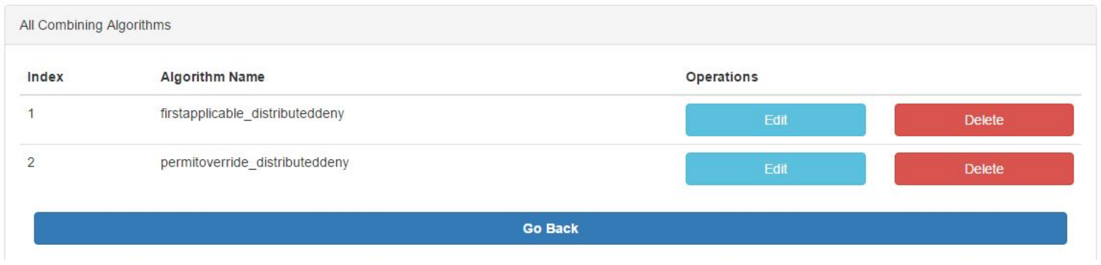
Add New Permission Value(multiple permissions separated by comma)

Permit,Deny

Step 2 → **Select Operation**

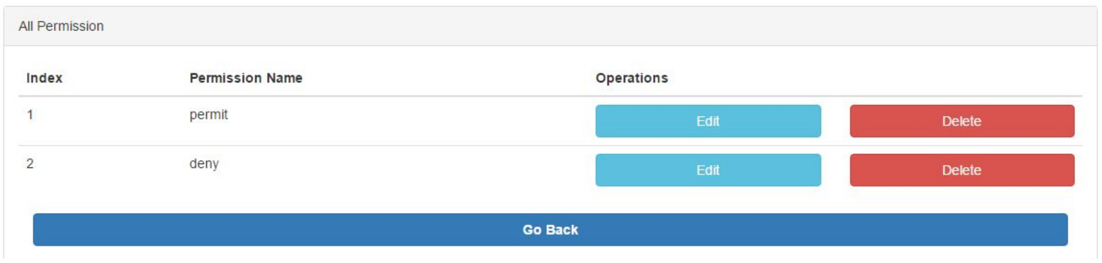
Fig. 3.19 Screen shot of ATLAS GUI to add new permissions

Fig 3.19 shows the similar layout used to add new permissions, see fig 3.19. We have added “permit" and “deny" as two very basic permissions using this layout.


All Combining Algorithms

Index	Algorithm Name	Operations	
1	firstapplicable_distributeddeny	Edit	Delete
2	permitoverride_distributeddeny	Edit	Delete

Fig. 3.20 Screen shot of ATLAS GUI to modify existing combining algorithms


All Permission

Index	Permission Name	Operations	
1	permit	Edit	Delete
2	deny	Edit	Delete

Fig. 3.21 Screen shot of ATLAS GUI to modify existing permissions

Fig 3.20 and Fig 3.21 layouts are used to update or delete existing combining algorithm and permissions respectively.

3.5.1.4 Manage Properties Files

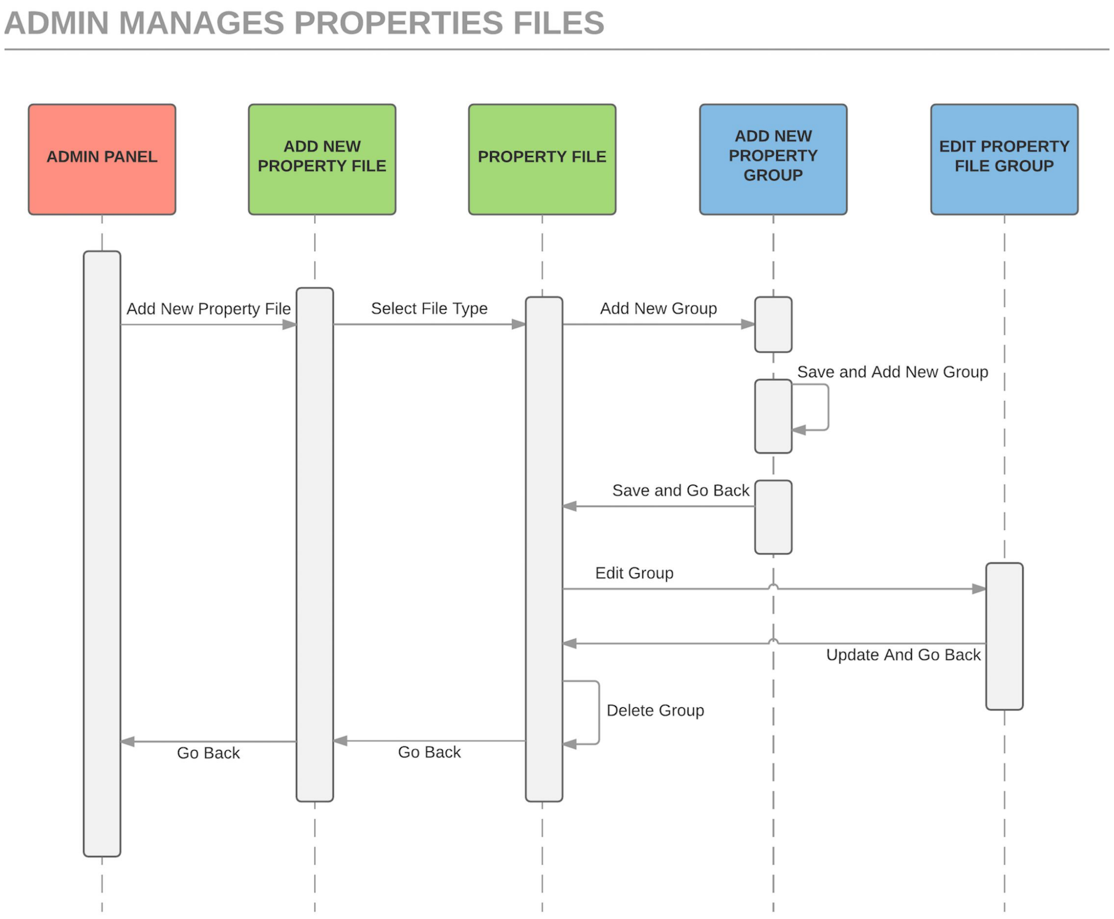
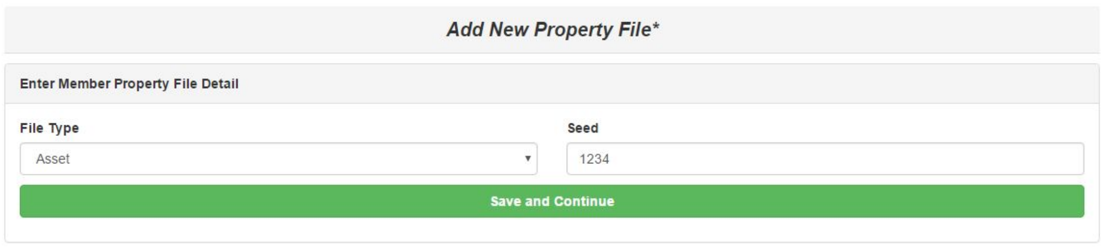


Fig. 3.22 Sequence diagram to manage property files using ATLAS GUI

In this section, Fig 3.22 represents the flow of adding and modifying properties files. We will discuss the properties files later in this section, but first we should explain how the ATLAS GUI works to create and manage properties files. Property files are the combination of groups of values and numbers.

In Fig 3.23 we will learn how to create a new property file using our ATLAS GUI.



*Add New Property File**

Enter Member Property File Detail

File Type: Asset

Seed: 1234

Fig. 3.23 Screen shot of ATLAS GUI to add new property file

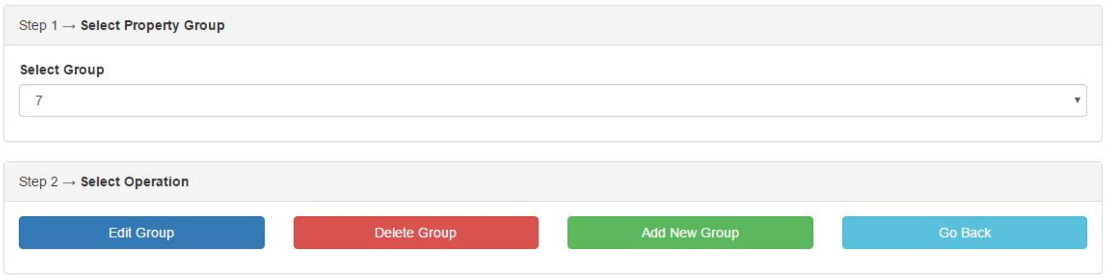
Property files are the helping files that DomainManager needs in order to evaluate the policy according to required specifications. Before ATLAS GUI these property files were created manually and thus had high chances of containing both logical and syntax errors. So, we have designed a layout which addresses this problem.

In order to generate a property file admin selects “Create New Property File” option from admin control panel, as shown in fig 3.11, and then will be redirected to a layout as shown in fig 3.23 where admin will select the “File Type" to generate. Input fields for entering seed number will appear on the basis of “File Type" depending ion whether it needs a “seed” value or not. After selection of “File Type", the admin will hit “save and continue” button which will take admin to a screen to create a new file selected from the dropdown list.

Assume the admin has selected “Asset" as a “File Type" from dropdown list and enters “1234" as its seed value in Fig 3.23. After saving “File Type" and its seed, the next screen as shown in fig 3.24 shows two possibilities. One is that it is a new file and so is empty. For this scenario, the admin will add new a group by clicking the “Add new Group" button and be redirected to the layout shown in fig 3.25

There are two types of properties files. The first type is a properties file for actions, resources and subjects, and the other type combines similar resources or subjects in to a group to apply conditions on all together. The next step is to add groups to properties files, for this we will use the layout of fig 3.24.

To generate property files we need to add group using Fig 3.24, To begin, there will be no group, so we need to add new group by pressing “Add New Group” Button in Step 2. This will redirect us to the layout in fig 3.25 to add a new group to the property file.

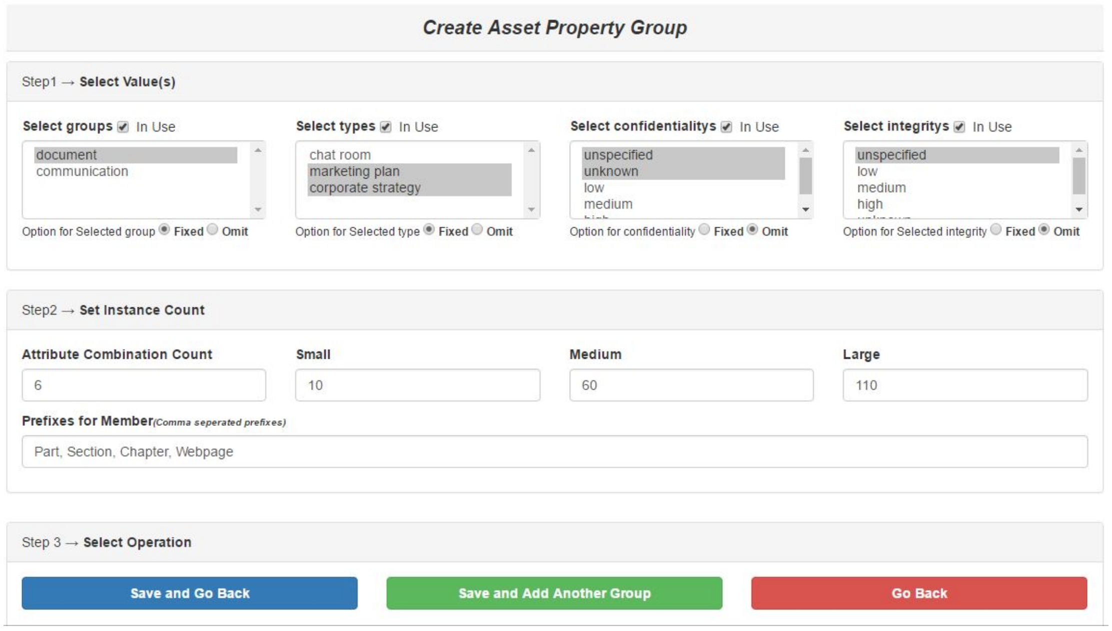

Step 1 → **Select Property Group**

Select Group

7

Step 2 → **Select Operation**

Fig. 3.24 Screen shot of ATLAS GUI to add or modify group


Create Asset Property Group

Step1 → **Select Value(s)**

Select groups ✓ In Use	**Select types** ✓ In Use	**Select confidentialitys** ✓ In Use	**Select integritys** ✓ In Use
document	chat room	unspecified	unspecified
communication	marketing plan	unknown	low
	corporate strategy	low	medium
		medium	high
Option for Selected group [✓ Fixed] Omit	Option for Selected type [✓ Fixed] Omit	Option for confidentiality Fixed [✓ Omit]	Option for Selected integrity Fixed [✓ Omit]

Step2 → **Set Instance Count**

Attribute Combination Count	**Small**	**Medium**	**Large**
6	10	60	110

Prefixes for Member*(Comma seperated prefixes)*

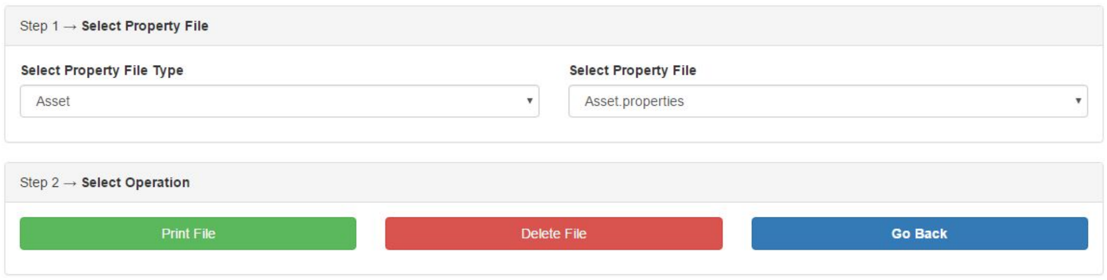
Part, Section, Chapter, Webpage

Step 3 → **Select Operation**

Fig. 3.25 Screen shot of ATLAS GUI to add new group into existing property file

Fig 3.25 shows how to add new groups by following three steps. The first step is to select values of different attributes for that property file, and an option for selected values is either to ignore it or to include it, by radio button selection. The second step is to set the count for attribute combination, number of instances to make for small, medium, large size requests, and comma separated prefixes that it will use as alias name while creating set of requests and policies. The last step is to to save the group and add more, or to save the group and go back or not to save for some reason and just go back. So, as we have seen in Fig 3.25 if the policy author selects “document" as a selected group option and from next column selected two values to include as group type which are “marketing plan" and “corporate strategy". After selecting group type values the policy author chose “unspecified" and “unknown" as the level of confidentiality to ignore for this group. The policy author has ignored the “unspecified" integrity for this group. So, the policy author has now selected all values from step1 and now in step2 employee entered “6" as number of attribute combination followed by “10",“60" and “110" as number of instances to make small,medium and large size requests respectively. To give some prefixes to member of this group in request, the policy author has filled values for prefixes as “Part",“Section",“Chapter" and“Webpage" separated by comma. After this step the policy needs to decide whether to save and exit, etc.

Once the selected property file has been saved, more groups can be added or existing groups modified as shown in fig 3.26. These are the use cases for admin to initialize the GUI


Step 1 → Select Property File

Select Property File Type: Asset

Select Property File: Asset.properties

Step 2 → Select Operation

Fig. 3.26 Screen shot of ATLAS GUI to select property file to perform action on it

database and thus to create policies and to manage them. Section 3.5.2 discusses how to use the ATLAS GUI for policy writing use cases.

3.5.2 Policy Author Role

Policy authors can do four different actions using ATLAS GUI, which are as follows:

1. Create New Policy
2. Manage Existing Policy
3. Run Policy
4. View Results

The policy author steps are shown in fig 3.27. Most commonly, authors use layouts like fig 3.30 and fig 3.31. We will see in fig 3.35 the process of initiating the evaluation of policy performance experiments. Having run performance experiments, the next step is to view the results. After analyzing these results, users might need to modify their template policies, see fig 3.34. We have already seen this execution cycle in Fig 1.3.

Fig 3.27 is an interaction diagram for employee use cases. When the user lands on ATLAS GUI, logs in and is redirected to the “employee” control panel, see fig 3.28.

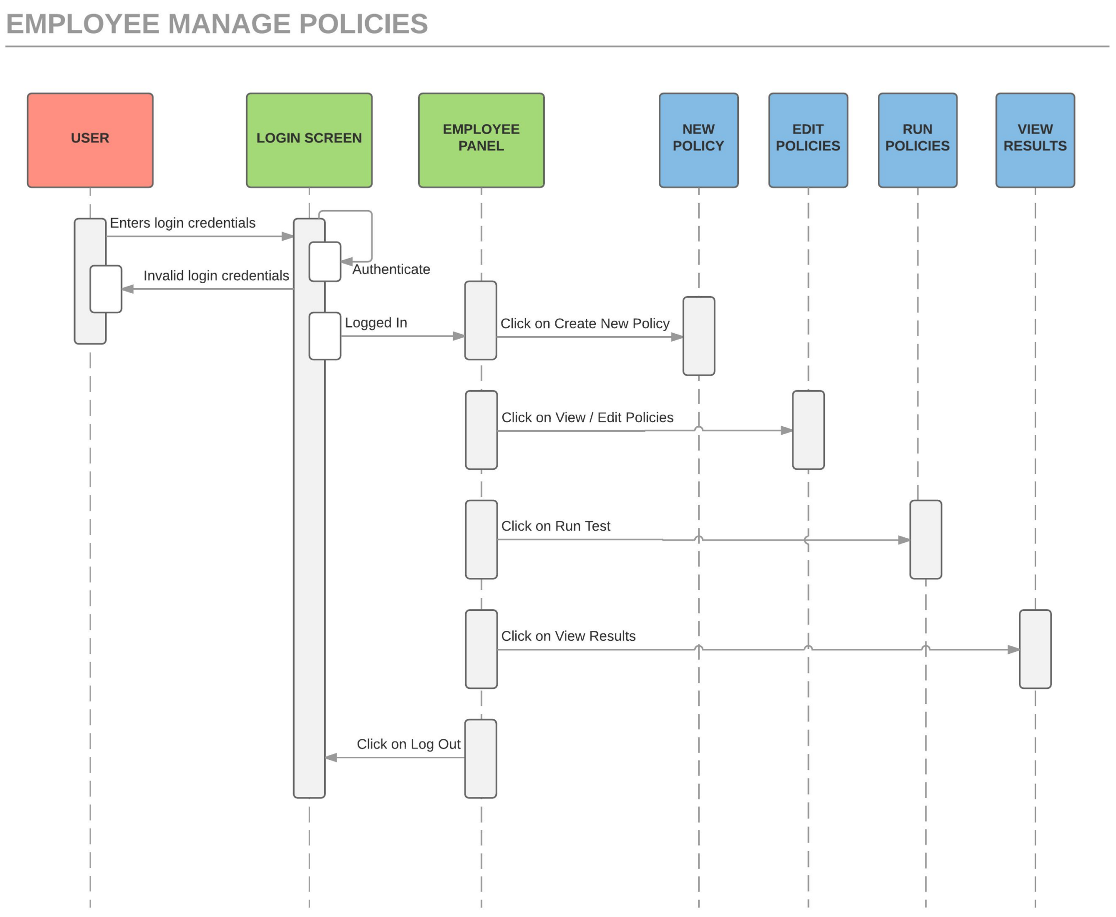


Fig. 3.27 Sequence diagram of ATLAS GUI to perform various actions as an employee

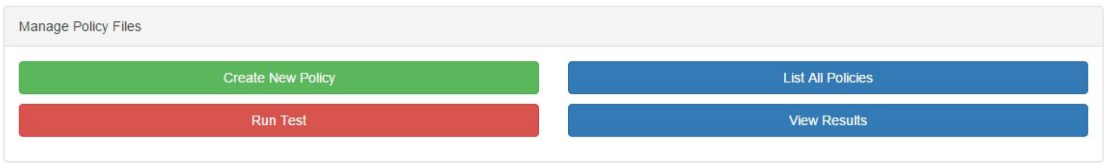


Fig. 3.28 Screen shot of ATLAS GUI as "employee" (policy author) control panel

3.5.2.1 Add New Policy

In fig 3.29, the user first needs to be logged in as an employee or admin. After successful login, the Policy author dashboard offers a "Create New Policy" button to start writing new policies using the "Add New Policy" layout, see fig 3.30. After selecting "save and continue", the policy author navigates to fig 3.31. After adding rules, the next screen is fig 3.32 to combine rules to make groups and rule groups to make new rule groups. Once satisfied with the template policy, the policy author can export it using "List all Policies" that navigates to

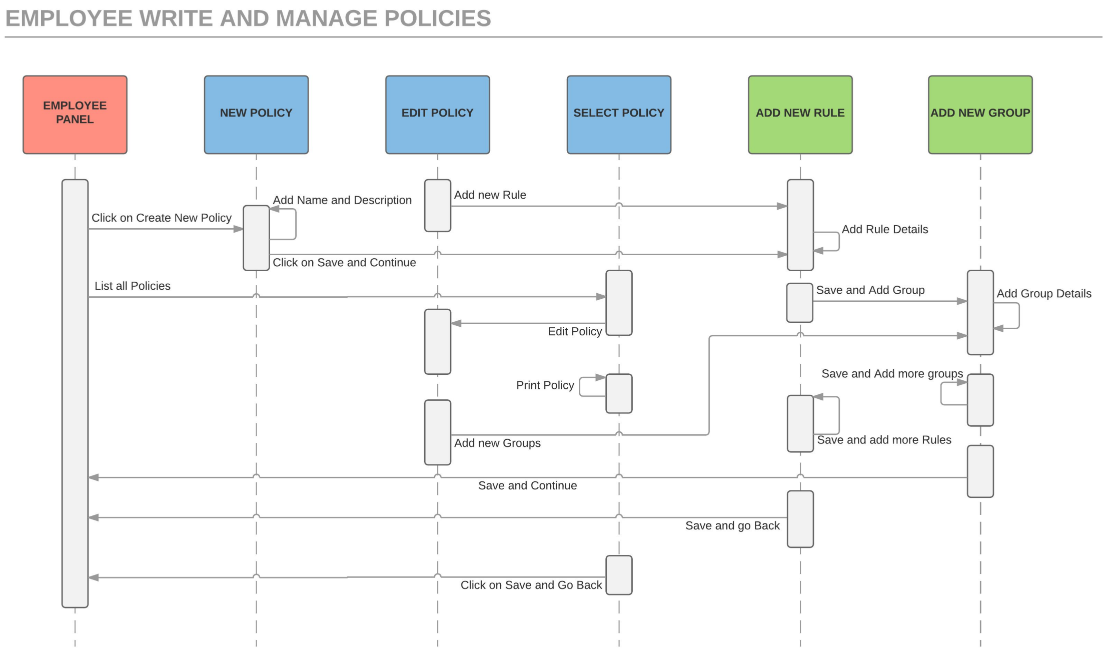
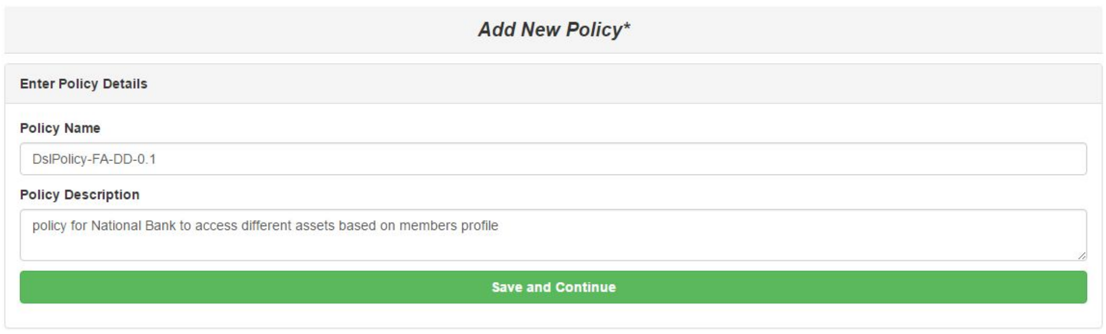


Fig. 3.29 Sequence diagram to write new policy using ATLAS GUI

fig 3.33. Otherwise, the policy author can modify the policy, or delete it if it‘s not needed anymore.


*Add New Policy**

Enter Policy Details

Policy Name

DslPolicy-FA-DD-0.1

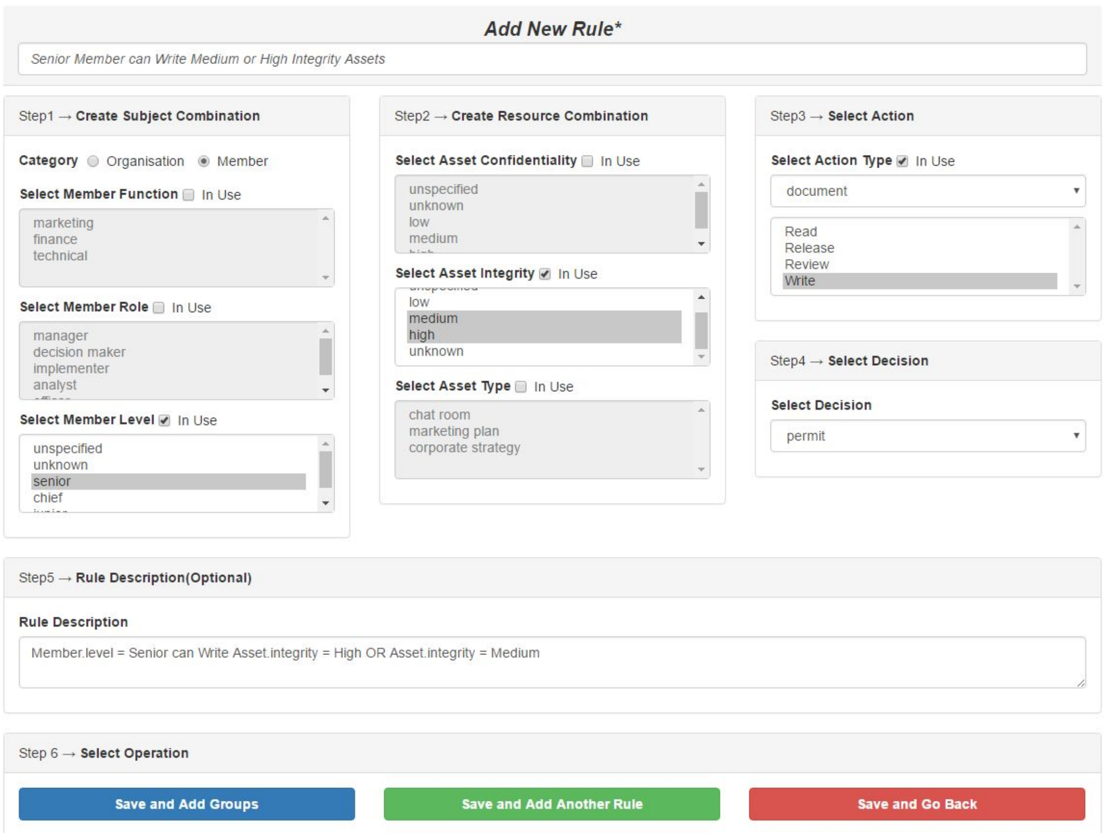
Policy Description

policy for National Bank to access different assets based on members profile

Fig. 3.30 Screen shot of ATLAS GUI to start writing new policy

To start the process, policy authors “Create new Policy” from Fig 3.28, then save name and description for that policy, see fig 3.30. The policy is still only a shell in that rules and/or rule groups still need to be added. As a concrete example, assume in fig 3.30 that the policy author has entered “DslPolicy-FA-DD-01" as a policy name along with description as shown in fig 3.30. After pressing the “Save and Continue" button the user is redirected to fig 3.31.

Fig 3.31 shows to add a new rule. This layout is logically divided into five steps, and designed in such a way that it is self-descriptive.



Add New Rule*

Senior Member can Write Medium or High Integrity Assets

Step1 → **Create Subject Combination**

Category ○ Organisation [✓ ● Member]

Select Member Function ☐ In Use
marketing
finance
technical

Select Member Role ☐ In Use
manager
decision maker
implementer
analyst

Select Member Level ☑ In Use
unspecified
unknown
[✓ senior]
chief

Step2 → **Create Resource Combination**

Select Asset Confidentiality ☐ In Use
unspecified
unknown
low
medium

Select Asset Integrity ☑ In Use
low
[✓ medium]
[✓ high]
unknown

Select Asset Type ☐ In Use
chat room
marketing plan
corporate strategy

Step3 → **Select Action**

Select Action Type ☑ In Use
document
Read
Release
Review
[✓ Write]

Step4 → **Select Decision**

Select Decision
permit

Step5 → **Rule Description(Optional)**

Rule Description
Member.level = Senior can Write Asset.integrity = High OR Asset.integrity = Medium

Step 6 → **Select Operation**

Fig. 3.31 Screen shot of ATLAS GUI to write new rule in policy

A rule is defined as “A target, an effect, a condition and (optionally) a set of obligations or advice. A component of a policy” (XACML-TC, 2005). So, it contains four basic components:

1. Subject
2. Resource
3. Action
4. Decision

The relational data model used by the GUI to represent these entities has been discussed in section 3.3. The five steps in fig 3.31 use these underlying tables.

1. Step1 is to create a “combination of subject attribute”. These are the attributes of subject that identifies it. For example, if subject is member, then every member will have it’s role, function and level. In fig 3.31 the user has entered the short description

of rule on the top and has selected “member" as “subject category" with only “member level" as member attribute option and selected “senior" level as a member that can perform an action according to this rule.

2. Step2 is to select “attributes for resource” on which we have to perform specific actions. In this screen shot the employee has selected two levels of integrity for resources i.e. either “medium" or “high" which means that if the resource has either integrity level it will be considered: “any medium or high level of integrity resources will be considered for this rule".

3. Step3 is to select “Action type” which is allowed to perform on selected resources in step2. Actions can be performed on resources consistent with the action type. In this example “Document" is the action type and “write" is the Action.

4. Step4 selects the decision for this particular rule, either to deny or permit if all conditions specified from step1 to step3 have been met. For this current example, the employee has selected a “permit” decision. Consequently this rule means: “Any member having senior level can write any documents having medium or high level of integrity".

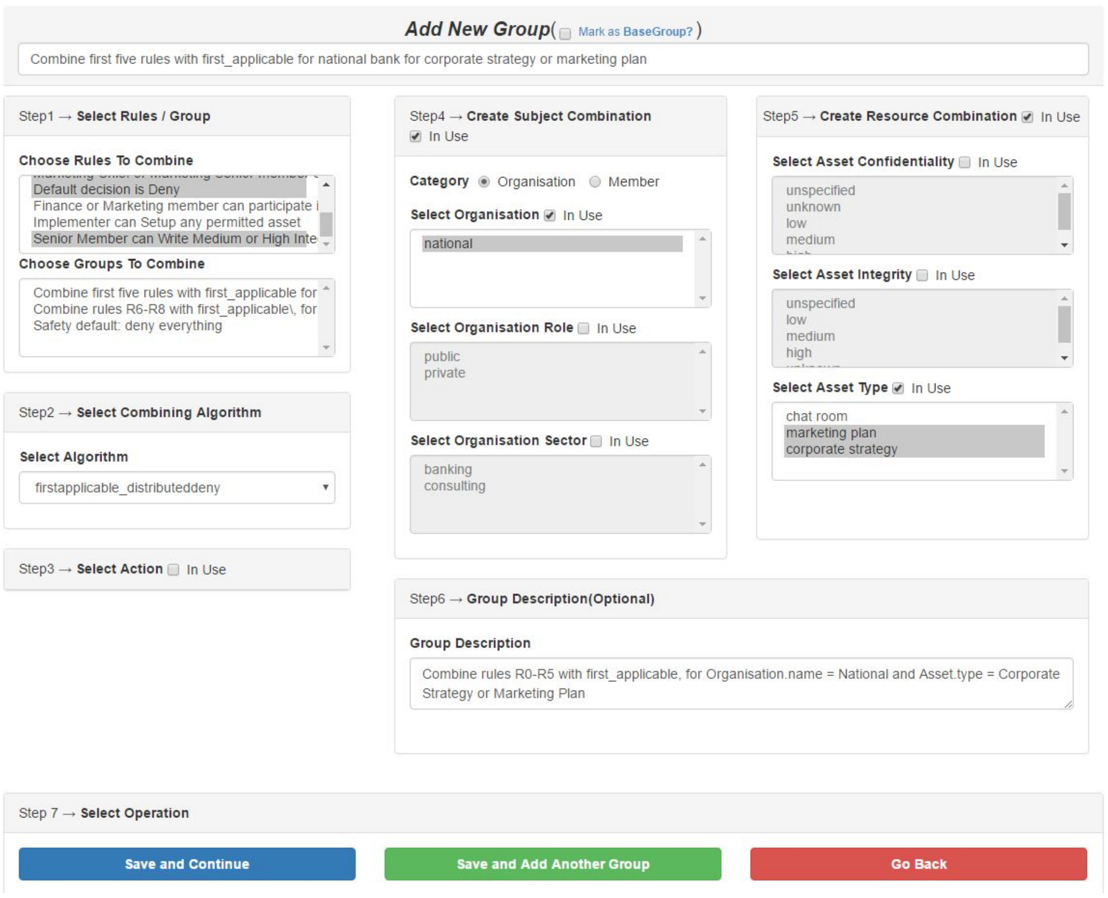
5. Step5 last but not the least, we have to add description(optional) which makes it easier for authors to scan and understand the rule.

Remember, the policy author can select more than one value or none for subject, resource and action according to need for rule.

After performing all these five steps, last step is to save this rule and navigate either to the screen shown in fig 3.32 to combine rules to form rule groups, to add new rules with this layout, or just to save the current rule in a policy and go back to the home page.

To add new rule groups to policies, we have designed the screen shown in fig 3.32. It is very similar to the layout for adding new rules to a policy, as conceptually the two procedures are very similar. The main addition is that one rule group has to be “baseGroup" and this baseGroup will be the entry point to start evaluating the policy.

Recall that the GUI needs to support modification and deletion as well as the addition of new entities, see fig 3.33. This layout can be used to edit existing policies, or to export selected policies in a file, or to delete it if there is no more need for that policy. When editing policies, users can add/remove rules and/or rule groups in an existing policy, or can update the description and/or name of the policy, see fig 3.34.



Add New Group(☐ Mark as BaseGroup?)

Combine first five rules with first_applicable for national bank for corporate strategy or marketing plan

Step1 → Select Rules / Group

Choose Rules To Combine

Default decision is Deny
Finance or Marketing member can participate i
Implementer can Setup any permitted asset
Senior Member can Write Medium or High Inte

Choose Groups To Combine

Combine first five rules with first_applicable for
Combine rules R6-R8 with first_applicable\, for
Safety default: deny everything

Step2 → Select Combining Algorithm

Select Algorithm: firstapplicable_distributeddeny

Step3 → Select Action ☐ In Use

Step4 → Create Subject Combination ✓ In Use

Category [✓ Organisation] Member

Select Organisation ✓ In Use: national

Select Organisation Role ☐ In Use: public, private

Select Organisation Sector ☐ In Use: banking, consulting

Step5 → Create Resource Combination ✓ In Use

Select Asset Confidentiality ☐ In Use: unspecified, unknown, low, medium

Select Asset Integrity ☐ In Use: unspecified, low, medium, high

Select Asset Type ✓ In Use: chat room, marketing plan, corporate strategy

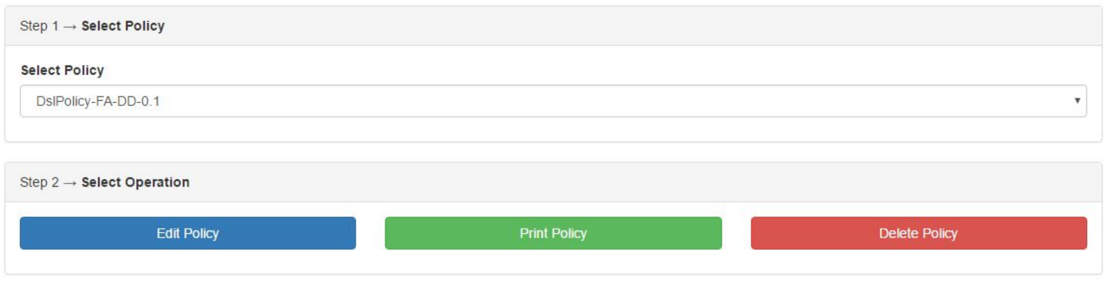
Step6 → Group Description(Optional)

Group Description

Combine rules R0-R5 with first_applicable, for Organisation.name = National and Asset.type = Corporate Strategy or Marketing Plan

Step 7 → Select Operation

Fig. 3.32 Screen shot of ATLAS GUI to form rule groups by combining them



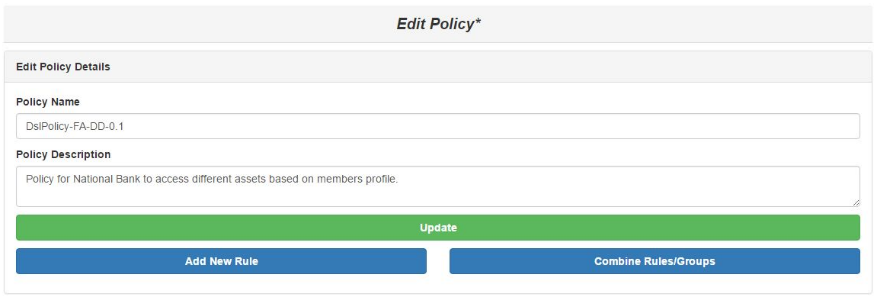
Step 1 → Select Policy

Select Policy: DslPolicy-FA-DD-0.1

Step 2 → Select Operation

Fig. 3.33 Screen shot of ATLAS GUI to perform operations on selected policy

Fig 3.35 layout shows how to run the test for a selected experimental configuration. Users need to “Select policy” from the dropdown, and choose XACML language version.



Edit Policy*

Edit Policy Details

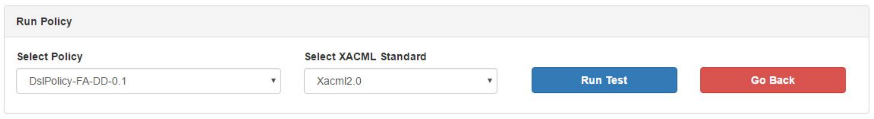
Policy Name

DslPolicy-FA-DD-0.1

Policy Description

Policy for National Bank to access different assets based on members profile.

Fig. 3.34 Screen shot of ATLAS GUI to modify existing policy



Run Policy

Select Policy: DslPolicy-FA-DD-0.1

Select XACML Standard: Xacml2.0

Fig. 3.35 Screen shot of ATLAS GUI to run existing policy on system

3.6 Summary

We have shown how configuring DomainManager is complex. Even with the ATLAS GUI, many steps are needed, but there is less scope for error because rich context is available at each step. And also there is less error because of schema that we have designed to hold the values and entries in database that we have discussed in section 3.3. While designing the GUI we have kept the concepts of Human Computer Interaction (HCI) in our mind in order to facilitate both admin and employee to write and manage policies with less effort and more accuracy in terms of time and learning. The GUI exports ATLAS configuration files in the expected format. If the GUI were not available, these configuration files would need to be edited by hand, and this presents a very steep learning curve to novice ATLAS users. Furthermore, *preliminary* steps have been made regarding how to use the GUI as a dashboard for running ATLAS-based experiments.

Chapter 4

Extending DomainManager and STACS

4.1 Introduction

In Chapter 3 we focused on how to make policy editing easier, as this would help security administrators to have more confidence when engaging in ever more complex performance experiments. In this chapter we indicate how ATLAS can be extended to open up the possibility of new types of performance experiment.

Our main focus for the research in this chapter is how to write, manage and evaluate policies in different policy writing standards and to evaluate policies using different implementations of PDPs. For evaluating the performance of policies, Butler et al. (2010) developed a testbed called STACS. STACS can be used to evaluate the sets of policies and requests generated by DomainManager. DomainManager (Butler and Jennings, 2015) takes template policies and supplementary configuration files as its input to generate sets of policies and requests. Performance experiments run for long times (often hours) and the policies were written and evaluated according to the (XACML-TC, 2005) Standard. XACML3.0 is an improved version of XACML2.0 standard with many new features in it, which we will discuss later in this chapter. After introducing the option to generate XACML 3.0 policies, we can then identify the performance gain or loss in evaluating the requests written for XACML3.0 policies.

In upcoming sections, we will show how we integrated Balana PDP with the existing STACS testbed in order to evaluate policies and request written in XACML3.0 standard. Balana PDP is an open source XACML3.0 implementation, which allows the user to evaluate requests having the optional Multiple Decision Profile. Also, we will highlight some of the differences between XACML2.0 and XACML3.0, and how we enabled DomainManager to export sets of policies and requests in XACML3.0 standard. So, in this chapter our main focus will be on;

1. Core differences between XACML2.0 and XACML3.0
2. Understanding how to add XACML3.0 export functionality to DomainManager
3. Integrating BalanaPDP in STACS (the performance testbed) to evaluate policies and requests written in XACML3.0

4.2 Evolution of XACML

4.2.1 Structure of XACML Policy Elements

Following are three basic elements of XACML:

1. Policy set
2. Policy
3. Rule

A policy set is a top level element in XACML policy schema and it can contain any number of policy elements and policy set elements. A policy can contain any number of rule elements according to OASIS XACML standard defined in (XACML-TC, 2005). Policies, policy sets and rules can contain Targets, which can contain subjects, resources, environments, and actions.

- **Subject** element is the entity requesting access. A subject has one or more attributes.
- **Resource** element is a data, service or system component. A resource has one or more attributes.
- **Action** element defines the type of access requested on the resource. Actions have one or more attributes.
- **Environment** element can optionally provide additional information.

XACML provides a "target", which is basically a set of simplified conditions for the subject, resource, and action that must be met for a policy set, policy, or rule to apply to a given request. Once a policy or policy set is found to apply to a given request, its rules are evaluated to determine the access decision and response. The target information also provides a way to index policies, which is useful if you need to store many policies and then quickly sift through them to find which of them apply. It works when a request to access that service arrives, the PDP will know where to look for policies that might apply to this request because the policies are indexed based on their target constraints.

The elements above are the basic blocks to form XACML based policies. We will see in section 4.2.2 why there comes a need to introduce a new standard for XACML, and the differences between XACML2.0 and XACML3.0 standard, which is our main focus for this chapter.

4.2.2 Need for the introduction of XACML2.0

XACML-TC (2005) explained the need of introducing XACML2.0 to the market;

> "Because the "economics of scale" have driven computing platform vendors to develop products with very generalized functionality, so that they can be used in the widest possible range of situations. "Out of the box", these products have the maximum possible privilege for accessing data and executing software, so that they can be used in as many application environments as possible, including those with the most permissive security policies. In the more common case of a relatively restrictive security policy, the platform's inherent privileges must be constrained, by configuration.
>
> The security policy of a large enterprise has many elements and many points of enforcement. Elements of policy may be managed by the Information Systems department, by Human Resources, by the Legal department and by the Finance department. And the policy may be enforced by the extra-net, mail, Wide Area Network (WAN) and remote-access systems; platforms which inherently implement a permissive security policy. The current practice is to manage the configuration of each point of enforcement independently in order to implement the security policy as accurately as possible. Consequently, it is an expensive and unreliable proposition to modify the security policy. And, it is virtually impossible to obtain a consolidated view of the safeguards in effect throughout the enterprise to enforce the policy. At the same time, there is increasing pressure on corporate and government executives from consumers, shareholders and regulators to demonstrate "best practice" in the protection of the information assets of the enterprise and its customers.

> For these reasons, there is a pressing need for a common language for expressing security policy. If implemented throughout an enterprise, a common policy language allows the enterprise to manage the enforcement of all the elements of its security policy in all the components of its information systems. Managing security policy may include some or all of the following steps: writing, reviewing, testing, approving, issuing, combining, analyzing, modifying, withdrawing, and enforcing policy.“

XACML-TC (2005) also explained why XML was selected to write such kinds of security policy;

> XML is a natural choice as the basis for the common security-policy language, due to the ease with which its syntax and semantics can be extended to accommodate the unique requirements of this application, and the widespread support that it enjoys from all the main platform and tool vendors.

XACML-TC (2013) was introduced as an enhanced version of XACML standard, XACML3.0 has many new features with improvement in several aspects which we will discuss in Section 4.2.3

4.2.3 XACML3 vs XACML2

XACML2.0 standard is in use since 2005, and many organisations write their access control policies in XACML2.0 for example, companies like PayPal, Bank of America and many more organisations (XACML3.0-specification-draft) are using XACML2.0. It is not easy for organisations to move their policies from XACML2.0 to XACML3.0 because the policies need to be rewritten and enterprises are not sure yet about the performance enhancement and complexities of migrating their policies from XACML2.0 to XACML3.0.

The following are some major enhancements and new features highlighted in (XACML3.0 vs XACML2.0);

- Custom Categories
- Targets
- Multiple Decision Profile

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Request  xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
            access_control-xacml-2.0-context-schema-os.xsd">
        <Subject>
            <Attribute
                  AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                  DataType="http://www.w3.org/2001/XMLSchema#string">
                <AttributeValue>
                  Dr. James Thomas
                </AttributeValue>
            </Attribute>
        </Subject>
        <Resource>
            <Attribute
                  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                  DataType="http://www.w3.org/2001/XMLSchema#anyURI">
                <AttributeValue>
                  http://medicalrec.com/record/patient/Pattrick
                </AttributeValue>
            </Attribute>
        </Resource>
        <Action>
            <Attribute
                  AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                  DataType="http://www.w3.org/2001/XMLSchema#string">
                <AttributeValue>
                  read
                </AttributeValue>
            </Attribute>
        </Action>
        <Environment/>
</Request>
```

Fig. 4.1 Request written in XACML2.0 standard

4.2.3.1 Custom Categories

Before XACML3.0 attributes were defined as a set of categories, which includes subject, resource, action and environment. So, XACML2.0 can have only four fixed categories for targets. These fixed categories are hard-coded elements and if we had to introduce a new category in our target there is no way of doing that using XACML2.0 standard. Also, each category has its own element tag in which information for that category was stored using XACML2.0, For example, if we have to write a request in which a subject asks to perform some action on certain resource, that request will be written using XACML2.0 standard as shown in fig 4.1, where we can see that request written in XACML2.0 has three element tags starting with categories name as “subject” and same goes for “resource” and “action”.

In XACML3.0, users are given the option to create their own custom categories. XACML3.0 generalized categories with element “category”. By this change, users can now add more categories as required to express their policies: they are not bound to use only the pre-defined categories. In fig 4.2 the same request is re-written using XACML3.0 standard and we can see clearly the difference of writing request in both XACML2.0 and XACML3.0. This new generalization gives the user more control and freedom to write policies and requests.

The difference in category specification is the major structural difference in writing policies and requests but there are others.

```
<?xml version="1.0" encoding="utf-8"?>
<Request xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
  http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
  xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Attributes
  Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute
    AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id">
      <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">
        Dr. James Thomas
      </AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
    <Attribute
    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
      <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#anyURI">
        http://medicalrec.com/record/patient/Pattrick
      </AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
    <Attribute
    AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
      <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">
      read
      </AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment" />
</Request>
```

Fig. 4.2 Request written in XACML3.0 standard

4.2.3.2 Target

In XACML2.0, a target groups attributes of the same category together under elements that reflect that category e.g. <Resources> and <Resource>. There is an “or” (disjunctive) relationship between attributes of the same category, and the top-level category elements are combined using an “and” (conjunctive) relationship.

In XACML3.0 they removed the disjunctive and conjunctive function of the category (e.g. <Resources>) elements and introduces the “AnyOf” and “AllOf” elements. The target element still bears the conjunctive function though. XACML2.0 had already introduced and defined the any-of and all-off functions but simply they did not have the equivalent schema elements: they were implicit and interpreted as such by PDPs. We can refer to table 4.1 to explain a new feature that has been introduced in XACML3.0 related to Targets. If there are two separate rules combined in a single rule as shown in last row of table, XACML2.0 can not support this to evaluate, but XACML3.0 supports to evaluate using a single Target. Specially it is helpful when there are two different actions perform by two different subjects on single resource then XACML3.0 helps us to evaluate such kind of rule using Targets.

Rule	XACML2.0	XACML3.0
Doctor view medical record	Yes	Yes
Doctor or nurse view medical record	Yes	Yes
Doctor or nurse view or edit medical record	Yes	Yes
Doctor view medical record or nurse edit medical record	No	Yes

Table 4.1 Target in XACML3.0 and XACML2.0

4.2.3.3 Multiple Decision Profile

In XACML2.0 we have a feature called Multiple Resource Profile (MRP) which allowed multiple requests for multiple resources only, for example, “Can I view resource 1 and can I view resource 2?”. This example allows multiple resources to have same action to be performed by the same subject. This type of request can be processed using a XACML2.0 feature called MRP. Also it is used to get a single response to a request for an entire hierarchy as mentioned in (Multiple resource profile of XACML2.0). This MRP feature groups all requests together which have same subject and action performed on different resources.

XACML3.0 introduced MDP as a generalisation of the MRP used in XACML2.0 MDP not only allows multiple resources to be grouped together but also it allows any category to be grouped together. For example, “Can I view and edit resource 1?” to which the PDP can reply “Permit” to “view” and “Deny” to “edit”.

So, by comparing MRP and MDP, we can clearly see the advantage of MDP over MRP, as it allows multiple conditions for any category. Based on this new enhanced feature we have some interesting findings of how PDP performs when it has to evaluate requests with MDP enabled PDP implementation in Chapter 5.

So far, we have discussed and compared some new features and enhancements of XACML3.0 with XACML2.0 standard. We need to add XACML3.0 support in our existing ATLAS system, so DomainManager will be able to generate sets of policies and requests based on XACML3.0 standard. To evaluate all those sets of requests with given policies in XACML3.0 standard we integrated BalanaPDP, which is the implementation of PDP forked from SunXACML PDP to give support for XACML3.0 based request evaluation. We will discuss how we enhanced the DomainManager to generate new sets of policies and requests based on XACML3.0 standard in section 4.3. We will discuss the integration of BalanaPDP in our existing STACS system in section 4.4.

4.3 DomainManager

DomainManager is the first application in the ATLAS platform toolchain. It takes a template policy and a set of property files and uses them to generate sets of policies and requests as shown in fig 4.3. The existing DomainManager implementation produces sets of policies and requests in XACML2.0 standard only. However it was designed with a set of interfaces to support other hierarchical policy languages. We have used this feature to enhance DomainManager to generate sets of policies and requests in XACML3.0 format. It was not an easy task to integrate both standards in one application, because of the structural differences. However the existing DomainManager presented in (Butler and Jennings, 2015) was a good basis because of its open, extensible architecture.

DomainManager was shown as the initial component in ATLAS in Fig 1.3. In Fig 4.3, DomainManager processes the property files and template policies given to it as input, in a process called "Facade". "Facade" sets up the basic details to generate a graph for policies and creates instance of `neo4j` graph based on the template policy specified by the user. Then it fills the graph nodes and relationships with content for generating policies and requests. If the user decides to generate policies for XACML2.0 it will follow the path labeled with a green colored line and box and with a blue color if it needs to generate policies and requests in XACML3.0 format.

To enable support for generating XACML3.0 from DomainManager, we use the same toolset as was used to generate XACML2.0: we use (Eclipse Foundation, 2005) to generate JAXB classes for the XACML3.0 schema. OASIS XACML-TC (2010) provides the structure of XACML3 policies and requests in "xsd" format, which is used as an input to Eclipselink MOXY to generate the JAXB domain classes for XACML 3.0 policies and requests. Given a policy in the internal representation used by DomainManager, it needs to be exported in a form suitable for use by XACML (2.0 and now also XACML 3.0) PDPs. To achieve this, DomainManager recursively descends through the internal policy representation, transforming its data to fit into the different hierarchical metamodel of XACML (2.0 and now also XACML 3.0). We can then use the annotations in the JAXB domain classes to specify how to unbind/serialize the XACML (2.0 and now also XACML 3.0) *objects* to XACML (2.0 and now also 3.0) *XML text*. Note that the extension of DomainManager relates to the ability to export policies and requests in XACML 3.0 format; this was not possible previously, when the only "option" was XACML 2.0 format.

The features we needed to add to DomainManager to support XACML 3.0 policies and requests were as follows. We created XACML 3.0 versions of the classes used as facades to the set of XACML 3.0 domain classes generated by Eclipselink MOXY. We also created XACML 3.0 versions of the classes used to translate from the internal property graph model of DomainManager to the XACML 3.0 structures expected by the XACML 3.0 facade classes. We also extended some of the existing DomainManager control classes to accept language version as an argument, and to choose the language serialisation specified by the new control argument.

Note that the basic structure of DomainManager remained unchanged, and the availability and use of common interfaces limited the scale and scope of the changes to relatively few classes. However, we did find that the more general XACML 3.0 underlying model made the mapping from the internal DomainManager graph model to the XACML 3.0 metamodel a lot more challenging than it was for XACML 2.0.

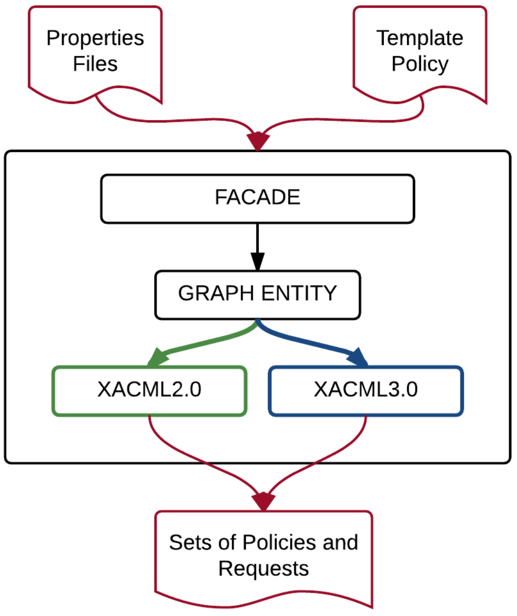


Fig. 4.3 Internal Working of DomainManager

4.4 STACS

(Butler et al., 2011) describes the STACS testbed, based on the testbed proposal in (Butler et al., 2010), in order to enable performance experiments to be carried out under controlled conditions. STACS takes sets of policies and requests as its input from DomainManager and evaluates each request based on he available policies and saves its decision and time taken for evaluation in a database for subsequent analysis. Fig 4.4 indicates how STACS works internally. Note that STACS allows multiple implementations of PDPs to evaluate (different flavours of) policies and requests. Before our work, STACS evaluated policies and requests based on the XACML2.0 standard only so the existing adapters were for SunXACML1.2, SunXACML2.0 and EnterpriseXACML PDPs which are able to evaluate XACML2.0 policies and requests.

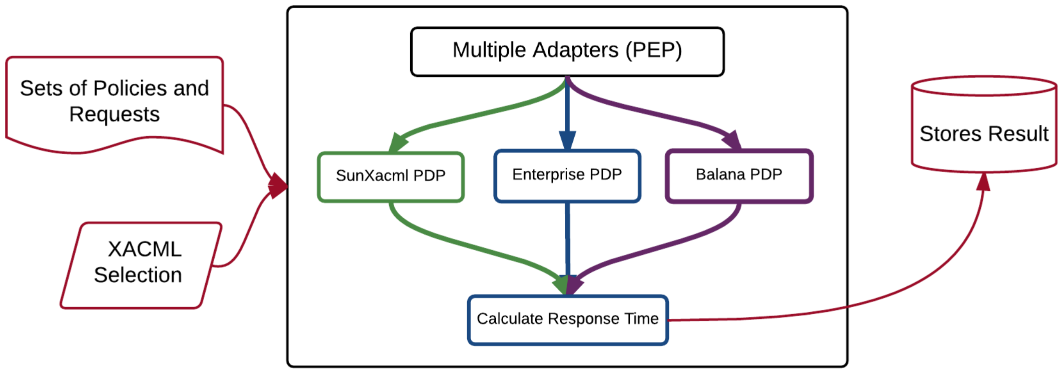


Fig. 4.4 Internal Working of STACS

We have enabled the use of XACML3.0 in existing STACS testbed, by integrating a new PDP named “Balana”, It is the latest open source XACML implementation based on SunXACML. Currently Balana supports both XACML2.0 and XACML3.0 (Pathberiya, 2012). STACS is a platform which allows PEP adapters to be added in order to accommodate different implementations of PDP.

In that regard, we added an adapter for the “Balana” PDP in order to evaluate requests based on the XACML3.0 standard. The new adapter for Balana is a class with the same external interface as that offered by the adapters for each of the existing XACML 2.0 PDPs available to STACS. Consequently, it is possible to conduct performance experiments involving Balana in the same way as that used for testing other PDPs. It is even possible to conduct comparative experiments where Balana operates on XACML 3.0 policies and requests while another PDP operates on XACML 2.0 policies and requests.

One of the reason of selecting “Balana” PDP as XACML3.0 based PDP is that it is open source and evaluation of MDP can be made optional. It works very similarly to the existing PDPs already integrated in STACS, so writing an adapter for it is relatively easy. When incorporated in STACS, it takes sets of policies and requests, evaluates each request and saves the service time in the results database used for all results in STACS. A graphical representation of the STACS system is shown in fig 4.4.

4.5 Summary

In this chapter we have studied and compared the similarities and dissimilarities in two standards of XACML for writing policies. The structure of XACML policy elements have been highlighted in Section 4.2.1. We have discussed the new features that have been introduced in XACML3.0 in section 4.2.3: introducing custom categories, targets, and multiple decision profile are some major enhancements that we have seen in XACML3.0. After discussing these new features and improvements in XACML3.0, we highlighted in Section 4.3 the work presented in (Butler and Jennings, 2015) that described `DomainManager`. We extended `DomainManager` to export XACML policies and requests as an option, when only XACML 2.0 policies and requests were exported previously.

We also added an adapter for BalanaPDP and thus a means of using STACS to run performance experiments comparing XACML 2.0 and XACML 3.0, see Chapter 5. The extension to ATLAS that we have made are consistent with Research Question **RQ2** mentioned in chapter 1.

Chapter 5

Experimental Analysis

5.1 Introduction

ATLAS GUI was designed in order to write and manage template policies and to integrate all existing applications of the ATLAS System. We have enabled support for generating XACML3.0 policies and requests in DomainManager and integrated Balana PDP in STACS in order to evaluate sets of requests and policies generated by DomainManager for both XACML2.0 and XACML3.0 standards. Researchers have compared the performance of evaluating the XACML2.0 based requests on different implementation of XACML2.0 PDPs as mentioned in Butler et al. (2011). XACML3.0 contains some new features and structural changes compared to XACML2.0, which we discussed in section 4.2.3. In chapter 4 we enhanced ATLAS to support experiments involving both XACML3.0 and XACML2.0 PDPs, policies and requests.

In section 4.2.3.1 we discussed enhancements in XACML3.0 to add new custom categories without having to create new tags for these categories. This enhancement generalized the elements and may save a little time in processing different element tags. We will examine this assumption later in this chapter. Also, a new feature called MDP was introduced which allows multiple requests for any category in request to be grouped together with separate decisions for each. For example, "Can I view *and* edit resource 1?". By looking at this feature we can check whether there is some impact on performance while evaluating the requests.

Ngo et al. (2013) optimized the XACML policy evaluation to gain performance improvement in policies, written in XACML3.0 standard. But no one has yet compared the performance of XACML3.0 vs XACML2.0 standard policies and requests with implementation of different PDPs. So, we conducted experiments to determine whether there is any impact on performance on the basis of which standard (XACML2.0 or XACML3.0) the policies are written. Also we conducted experiments using different implementations of PDPs to determine whether this is also an important factor.

5.2 Experimental Methods

We compared the performance of XACML2.0 and XACML3.0 using SunXACML and Balana PDP. We used improved STACS testbed that was discussed in section 4.4. We recorded response time for each request to determine its decision, and processed the records to measure the performance of XACML3.0 and XACML2.0 in various combinations. We compared their performances on the basis of;

1. Different implementation of PDPs: SunXACML PDP vs BALANA PDP
2. Different standards of XACML: XACML2.0 vs XACML3.0
3. Different nature of request complexities. i.e, FINE1 and FINEALL. FINE1 can contain at most one *instance-level* condition whereas FINEALL can contain multiple *instance-level* conditions in each slot. For example, a FINE1 request condition might be `Member.name = Alice` whereas a FINEALL (composite) request condition might be `Member.name = Alice OR Member.name = Bob OR Member.name = Carol OR ...`. The corresponding policy condition might have an *attribute-level* (COARSE) condition like `Member.level = Senior AND Member.department = Finance` which resolves to a list of persons (hence instances) satisfying this condition.
4. Different combining algorithms Brossard (2014) i.e, FirstApplicable and PermitOverride in order to see if there is any performance impact by using different algorithms to combine the decisions produced by different children of a parent policy (or policy set) into a single decision that the given policy will return to its own parent.
5. Different sizes for request, generally we had three request sizes; i) small with 100 elements in it, ii) medium with 1,000 elements and iii) large with 10,000 elements per requests.

To perform experiments we used a MacBook with an Intel Core i7 processor running at 2.8GHz speed with 16GB of memory. We repeated each measurement 9 times and in each iteration there were 3 sizes small, medium, and large having 100, 1,000, and 10,000 elements in each request respectively with two kind of complexities. FINE1 requests contain a single attribute value in each attribute and FINEALL requests can contain multiple values for each attribute in a request.

5.3 Results

We collected data by running experiments and saved the response time and decision for each request in a database. We labeled the raw data using five key factors identified in section 5.2. We compared the performance of XACML3.0 vs XACML2.0 requests based on different PDP implementations as our first indicator which is described in detail in section 5.3.1. On the basis of PDP comparisons we expanded our findings to compare the ratio of performance effects by replacing XACML3.0 with XACML2.0 in section 5.3.2. We discovered that XACML2.0 requests cumulative response time is almost the same on both (SunXACML and Balana) PDP implementations. We then aligned our key performance indicator to compare the effects of request complexity in section 5.3.3, which shows us that the performance of XACML2.0 is almost same in both complexities i.e requests having FINE1 or having FINEALL complexities but on the other hand, XACML3.0 with requests having FINEALL complexity take much longer time to make a decision than equivalent requests having FINE1 complexity in XACML3.0.

By contrast, the differences between XACML2.0 and XACML3.0 performance were negligible for requests having FINE1 complexity. After applying three different kinds of filtering, we were able to identify the reason of such performance degradation when evaluating XACML3.0 requests on Balana PDP. We have different combining algorithms Brossard (2014) in order to combine decisions of multiple requests. We compared the results of XACML3.0 based requests using different combining algorithms in section 5.3.4 and were able exclude the combining algorithms as a contributory factor to the performance differences. Therefore, by a process of elimination, we were able to determine that the performance degradation is because of FINEALL type request complexity while evaluating XACML3.0 requests on Balana PDP. The final step in our investigation was to compare performance by request size, i.e small, medium, large sized requests. We have expanded FINE1 and FINEALL complexity results on the basis of request size in section 5.3.5 and on the basis of that we have seen no performance impact while evaluating XACML3.0 based FINE1 requests across request size, but on the other hand in Fig 5.5b we have seen superlinear performance degradation to evaluate large size XACML3.0 requests having FINEALL complexity.

So, we have discovered very interesting findings by drilling down from the overall results, excluding possible explanations as we went. We believe the performance degradation in

XACML3.0 requests are because of the MDP feature which is enabled in Balana PDP. By checking the code, we discovered the PDP then iterates over the requests to collect and refine the decision for multiple requests.

5.3.1 PDPs Comparison

Fig. 5.1 compares the service times of Balana and SunXACML PDP. It shows that XACML3.0 requests in Balana PDP took almost 60ms to decide its decision as permit for cumulative requests, and 40ms for evaluating cumulative requests with decision as deny. By contrast, the cumulative XACML2.0 requests in Balana PDP and SunXACML PDP evaluated in almost 2.0ms for both permit and deny decisions.

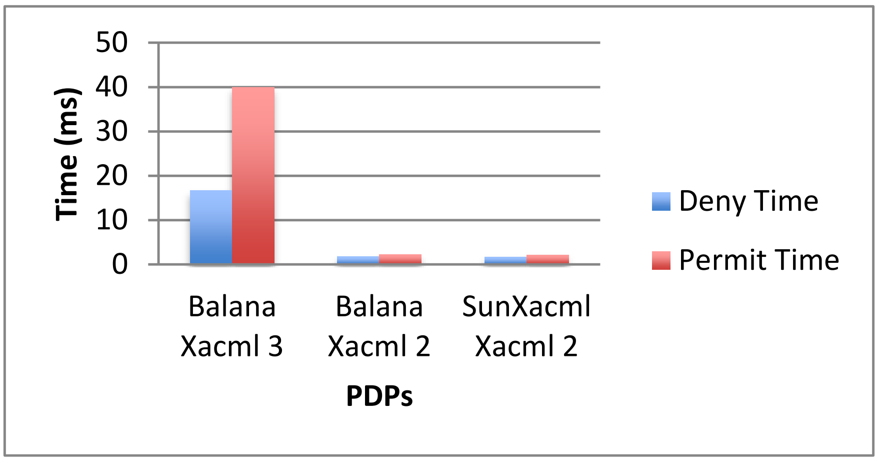


Fig. 5.1 Evaluation time of Balana Vs SunXACML PDP

5.3.2 XACML 2.0 vs. XACML 3.0 Comparison

Fig. 5.2 shows the performance improvement arising from replacing XACML3.0 with XACML2.0 in Balana PDP. It shows that the evaluation time for permit decision increases around 17 times whereas processing time for deny decision increases around 10 times. XACML2.0 performance in Balana and SunXACML PDP is almost the same in relation to both permit and deny decision.

So, as we can see there is no significant difference in performance between Balana and SunXACML PDP for processing XACML2.0 requests, because Balana PDP is forked from SunXACML PDP 1.2 and our attention changes to finding what makes XACML2.0 more

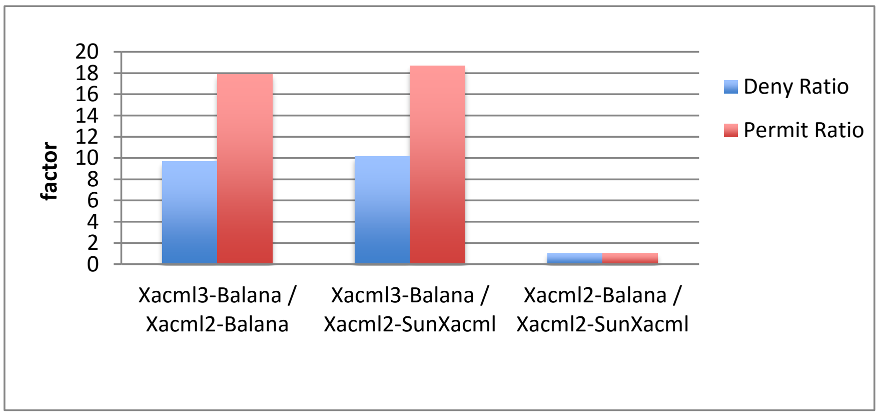
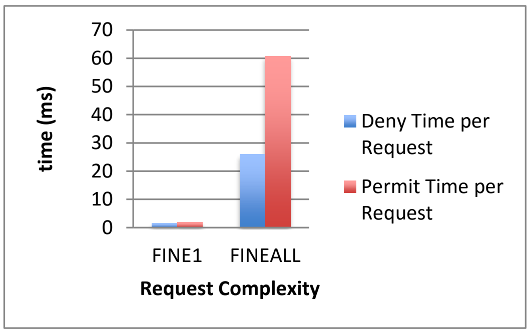


Fig. 5.2 XACML3.0 to XACML2.0 performance ratio across PDPs

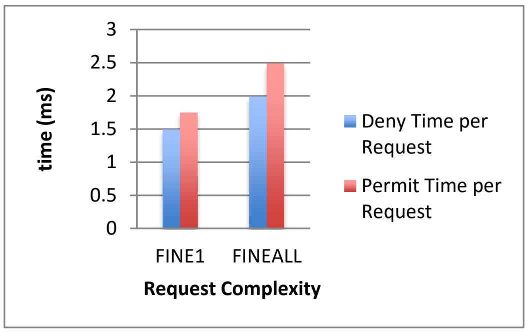
efficient when it comes to evaluating requests in Balana and SunXACML PDP over evaluating XACML3.0 using Balana PDP.

5.3.3 Complexity Comparison

Fig. 5.3 shows service time results based on complexity of requests written in XACML2.0 vs XACML3.0, either permit or deny with two kind of requests complexities. One is FINE1 (single condition requests) and the other is FINEALL (multiple condition requests)



(a) XACML3.0 requests



(b) XACML2.0 requests

Fig. 5.3 Request Complexity Comparison: XACML2.0 vs XACML 3.0

Fig. 5.3a shows that Balana PDP took around 61ms to evaluate XACML3.0 requests having FINEALL complexity and around 27ms to make Deny decisions. For XACML3.0 requests having FINE1 complexity Balana PDP takes around 1.91 and 1.61ms for permit and deny decisions respectively.

Fig. 5.3b shows that Balana PDP took around 2.48ms to evaluate XACML2.0 requests having FINEALL complexity and around 1.98ms to make Deny decisions. For XACML2.0

requests having FINE1 complexity the Balana PDP takes around 1.74 and 1.48ms for permit and deny decisions respectively.

So, as we can see clearly, in Fig. 5.3a and Fig. 5.3b, that XACML3.0 FINEALL complexity requests take much more time to make a decision compared to XACML2.0 FINEALL and FINE1 requests. So we have a deeper look into XACML3.0 FINEALL requests to determine the reason why it takes so much time to make a decision.

5.3.4 Combining Algorithm Comparison

Performance tests were run with two different complexity levels of requests and along with combining algorithms within these complexity levels to validate whether the large performance drop is because of combining algorithm or not.

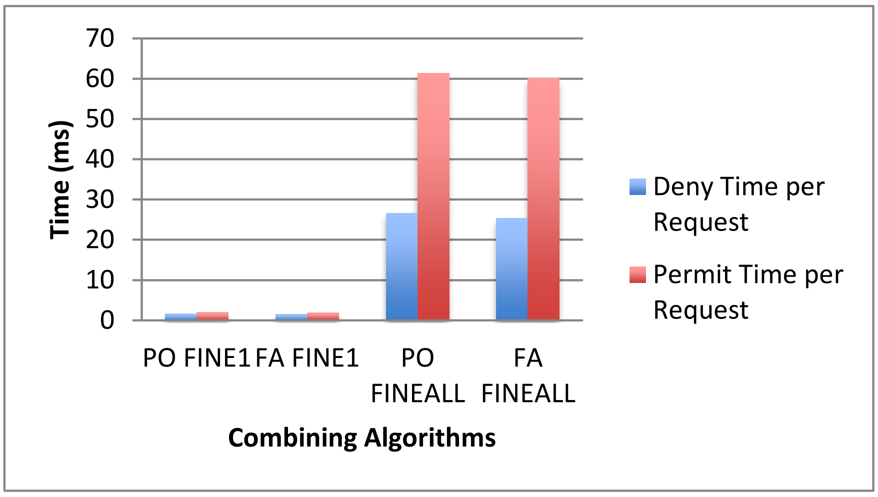
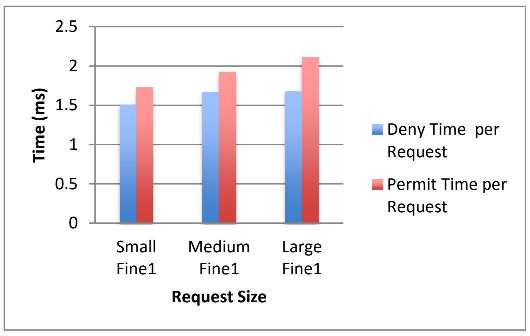


Fig. 5.4 Combining algorithms comparison with request complexity in XACML3.0

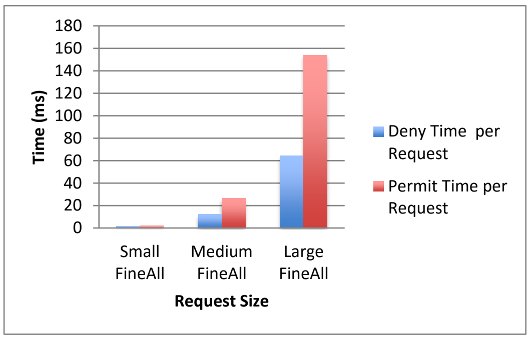
Fig. 5.4 shows that combining algorithms have no significant effect on performance while evaluating XACML3.0 requests having FINE1 and FINEALL complexities. Indeed, Permit-Override(PO) and First-Applicable(FA) combining algorithms take 1.67 and 1.96ms for decision as deny and permit respectively under FINE1 complexity. On the other hand, requests having FINEALL complexity take almost 26.65 and 61.00ms for deny and permit decisions respectively. This indicates that we need to investigate another factor.

5.3.5 Request Size Comparison

Fig. 5.5 shows results of XACML3.0 requests, by decision (either permit or deny) and with two kind of requests complexities: FINE1 and FINEALL.



(a) XACML3.0 FINE1 requests



(b) XACML3.0 FINEALL requests

Fig. 5.5 Request Complexity Comparison: XACML3.0 FINE1 vs FINEALL

Fig. 5.5a shows that Balana PDP took on average 1.5, 1.6 and 1.67ms to decide *deny* decision for XACML3.0 small, medium, and large size requests having FINE1 complexity respectively. And it took 1.7, 1.9, and 2.1ms for Balana PDP to decide *permit* decision for XACML3.0 small, medium, and large size requests having FINE1 complexity respectively.

Fig. 5.5b shows that Balana PDP took on average 1.4, 12.18 and 64.25ms to decide *deny* decision for XACML3.0 small, medium, and large size requests having FINEALL complexity respectively. And it took 1.8, 26.57, and 153.82ms for Balana PDP to decide *permit* decision for XACML3.0 small, medium, and large size requests having FINEALL complexity respectively.

So, as we can see clearly, in Fig. 5.5a and Fig. 5.5b, the performance of XACML3.0 with FINE1 complexity requests is not affected by the size of those requests for both deny and permit decisions. But, we can see certain difference in Large request size having FINEALL complexity for XACML3.0. As, small size requests took 1.4 and 1.8ms to deny and permit requests respectively with FINEALL complexity, and for large size requests it took Balana PDP 64.25 and 152.82ms to evaluate FINEALL complexity XACML3.0 requests. which is almost 38.5 and 73.24 times slower for evaluating Large size requests to make deny and permit decision having FINEALL complexities respectively.

The interaction between request complexity and domain size is because, as the domain size increases, FINEALL requests will have more request conditions. For example, in a small domain perhaps only two persons satisfy `Member.level = Senior AND Member.department = Finance`, so the FINE1 request condition would be either `Member.name = Alice` or `Member.name = Bob` and the FINEALL request condition would be `Member.name = Alice OR Member.name = Bob`. In a large domain, perhaps fifty persons satisfy the same attribute-level condition (`Member.level = Senior AND Member.department = Finance`) so the FINE1 condition would look much the same as the small domain, but the FINEALL condition would look like `Member.name = Alice01 OR Member.name = Alice02 OR ... OR Member.name = Alice50`. With XACML 2.0's MRP, a single response decision is given: PERMIT or DENY to all persons. With XACML 3.0's MDP, a response decision is given to *each* subcondition, e.g., `Alice01` receives PERMIT while `Alice02` receives DENY. Clearly, MRP does not always require that all subconditions are evaluated to return the *overall* decision, but MDP does not have that advantage. Consequently, a PDP operating in XACML 3.0 mode can be outperformed by a PDP operating in XACML 2.0 mode because of this inherent difference in how multiple conditions are handled by the two XACML standards. By using DomainManager and STACS to conduct the experiments and collect the service times for analysis, we were able to drill down to find the conditions where the performance differences arise, and then were able to pinpoint the relevant corresponding source code in Balana (a XACML 3.0-compliant PDP) and SunXACML (a XACML 2.0-compliant PDP) to confirm our findings regarding the cause of the performance differences. It should be noted that the OASIS XACML committee probably introduced MDP in place of MRP in an attempt to make policies easier to specify. However, they did not have the benefit of ATLAS to understand the possible performance implications of this change.

5.3.6 Normalized Service Response Time

We have seen in Fig. 5.1 the overall evaluation performance in Balana and SunXACML PDP using different XACML standards, i.e XACML2.0 and XACML3.0. Using this plot we investigated several factors that might have caused such behavior and identified the root cause of such performance differences in fig 5.5b.

Up till now we have shown the individual comparison of different key factors that might affect the performance of requests evaluation, now we collect the results in a single table 5.1 to break down the performance degradation that we have seen in fig 5.5b. We normalized the service response time based on the shortest time that was measured, i.e., evaluating a XACML2.0 request with a Deny decision having FINE1 complexity in SunXACML PDP highlighted with green in table 5.1. The minimum and maximum service time values were 1.4 (for this set of conditions) and 60.8ms (for evaluating a XACML3.0 request with a Permit decision having FINEALL complexity in Balana PDP) respectively. After normalization we have identified XACML3.0 FINEALL based requests evaluated in BalanaPDP as two outliers highlighted with red color in table 5.1 taken 44.8 and 19.2 times as long as the best case for permit and deny responses respectively. The root cause for such performance degradation has been discussed above in Section 5.3.5.

Normalized Average Service Response Time				
PDP	XACML version	Complexity	Decision	Normalized Service Time (ms)
Balana	XACML3	Fine1	Permit	1.4
			Deny	1.2
		FINEALL	Permit	44.8
			Deny	19.2
	XACML2	Fine1	Permit	1.3
			Deny	1.1
		FINEALL	Permit	1.8
			Deny	1.5
SunXACML	XACML2	Fine1	Permit	1.2
			Deny	1.0
		FINEALL	Permit	1.8
			Deny	1.4

Table 5.1 Overall Normalized Average Service Response Time of Requests

Chapter 6

Conclusion and Future Work

6.1 Conclusion

By introducing ATLAS GUI and integrating it with the overall ATLAS System, it enabled the support of writing and managing template policies for organisations. We believe that, by making the context of each editing step clearer, it is much easier for policy authors. Chapter 3 describes how this functionality was added. This work was motivated by our efforts to address Research Question **RQ1** as defined in Section 1.3.

In response to Research Question **RQ2**, we have integrated XACML3.0 support in existing applications of ATLAS System. This enables ATLAS users to conduct more varied performance comparisons than before. Examples of such comparisons include

- Other things being equal, will “upgrading” to XACML 3.0 from XACML 2.0 help or hinder performance?
- Different PDP improvement proposals have been proposed, and implementations have been offered either for XACML 2.0 or XACML 3.0. Which of these proposals result in better performance, independent of the choice of XACML language variant?
- What are the best practices for organizing/structuring policies, and do they differ between XACML 2.0 and XACML 3.0?

Our major performance finding is that the MDP feature introduced with XACML 3.0 is potentially expensive in relation to performance. Consequently policy evaluation should be configured in such a way that the adverse effects of this new feature are reduced. For example, the trade-off between batching related requests together, versus sending them as individual requests in sequence, can be investigated using ATLAS.

The differences between MRP and MDP relate not just to performance. There is a very important semantic difference between then: MDP can return many decisions but with MRP a single overall decision is returned. Depending on the intentions of the policy, either approach might suit. This is an example where converting from XACML 2.0 to XACML 3.0 syntax might introduce unwanted semantic differences, in addition to performance differences. Although it is not discussed in this dissertation, ATLAS also records the decisions arising from each policy-request evaluation, and not just the service times, so simple semantic comparisons are also possible.

6.2 Future work and recommendations

In future the GUI could be extended to look more professional with different add-ons to make it easier to use in large organizations. For example, it would be good to integrate a) with Lightweight Directory Access Protocol (LDAP) stores to provide a source of organisation and/or member data, and b) to integrate with suitable content management systems to provide a source of resource data. Therefore, instead of administrators having to define manually the company hierarchies and resource characteristics, they could be collected directly, in real time, from the authoritative systems of record.

A further enhancement to the GUI would be to able to control the operation of ATLAS itself. At present, this is done by editing script files directly. In future these scripts could be replaced by alternative, more web-friendly specifications such as Ansible playbooks. There might then be scope to share control operations with the GUI, since it uses open web standards. Indeed, performance investigations such as the one that led to the discovery of the influence of MDP would be greatly eased if users had easy access to the powerful analysis of the PARPACS component of ATLAS, which currently is more suited to statistically expert users.

The investigation of XACML2.0 versus XACML3.0 identified an interesting performance difference. However, many supplementary questions can be asked, starting with those outlined in Section 6.1.

Bibliography

Jason Barron. *Policy Authoring and Analysis Processes for Federation Policies*. PhD thesis, School of Science and Computing, Waterford Institute of Technology, Waterford, Ireland, 2013. URL http://repository.wit.ie/id/eprint/2763.

Bootstrap Team. Bootstrap a framework for developing responsive, mobile first projects on the web, May 2015. URL http://getbootstrap.com.

David Brossard. Understanding XACML combining algorithms, Jul 2014. URL http://www.axiomatics.com/blog/entry/understanding-xacml-combining-algorithms.html.

Bernard Butler and Brendan Jennings. Measurement and Prediction of Access Control Policy Evaluation Performance. *Network and Service Management, IEEE Transactions on*, 12(4): 526–539, 2015. doi: 10.1109/TNSM.2015.2486519.

Bernard Butler, Brendan Jennings, and Dmitri Botvich. XACML Policy Performance Evaluation Using a Flexible Load Testing Framework. In *Proc. 17th ACM Conference on Computer and Communications Security (CCS 2010)*, pages 648–650. ACM, 2010. ISBN 978-1-4503-0245-6. doi: http://doi.acm.org/10.1145/1866307.1866385. URL http://repository.wit.ie/1698/1/extendedAbstract-butler.pdf. Short paper.

Bernard Butler, Brendan Jennings, and Dmitri Botvich. An experimental testbed to predict the performance of XACML Policy Decision Points. In *Proc. IM 2011 - TechSessions*, May 2011. doi: 10.1109/INM.2011.5990711. URL http://repository.wit.ie/1652/1/im2011bbv2_0-CAMERAREADY-XPRS.pdf.

David Chadwick, Gansen Zhao, Sassa Otenko, Romain Laborde, Linying Su, and Tuan Anh Nguyen. PERMIS: a modular authorization infrastructure. *Concurr. Comput. : Pract. Exper.*, 20:1341–1357, August 2008. ISSN 1532-0626. doi: 10.1002/cpe.v20:11. URL http://portal.acm.org/citation.cfm?id=1399603.1399607.

Alberto De la Rosa Algarín, Timoteus B. Ziminski, Steven A. Demurjian, Yaira K. Rivera Sánchez, and Robert Kuykendall. *Generating XACML Enforcement Policies for Role-Based Access Control of XML Documents*, pages 21–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-662-44300-2. doi: 10.1007/978-3-662-44300-2_2. URL http://dx.doi.org/10.1007/978-3-662-44300-2_2.

Eclipse Foundation. Eclipselink MOXY, 2005. URL http://www.eclipse.org/eclipselink/#moxy.

David Ferraiolo, Ramaswamy Chandramouli, Rick Kuhn, and Vincent Hu. Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC). In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, ABAC '16, pages 13–24, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4079-3. doi: 10.1145/2875491.2875496. URL http://doi.acm.org/10.1145/2875491.2875496.

Leigh Griffin, Bernard Butler, Eamonn de Leastar, Brendan Jennings, and Dmitri Botvich. On the performance of access control policy evaluation. In *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY 2012)*, pages 25–32. IEEE, July 2012. doi: http://dx.doi.org/10.1109/POLICY.2012.15. URL http://repository.wit.ie/1739/.

Xin Jin, Ram Krishnan, and Ravi Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In Nora Cuppens-Boulahia, FrÃ©dÃ©ric Cuppens, and Joaquin Garcia-Alfaro, editors, *Proceedings of the 26th Annual IFIP WG 11.3 conference on Data and Applications Security and Privacy*, volume 7371 of *Lecture Notes in Computer Science*, pages 41–55. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-31539-8. doi: 10.1007/978-3-642-31540-4_4. URL http://dx.doi.org/10.1007/978-3-642-31540-4_4.

Mathias Kohler and Achim D. Brucker. Caching Strategies: An Empirical Evaluation. In *International Workshop on Security Measurements and Metrics (MetriSec)*, pages 1–8. ACM Press, New York, NY, USA, 2010. ISBN 978-1-4503-0340-8. doi: 10.1145/1853919.1853930. URL http://www.brucker.ch/bibliography/abstract/kohler.ea-caching-2010.

Butler W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8:18–24, January 1974. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/775265.775268.

Reuven M. Lerner. At the Forge: Redis. *Linux J.*, 2010(197), September 2010. ISSN 1075-3583. URL http://dl.acm.org/citation.cfm?id=1883519.1883524.

Alex X. Liu, Fei Chen, JeeHyun Hwang, and Tao Xie. Xengine: a fast and scalable XACML policy evaluation engine. In *Proc. ACM SIGMETRICS international conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2008)*, pages 265–276. ACM, 2008. ISBN 978-1-60558-005-0. doi: http://doi.acm.org/10.1145/1375457.1375488.

A.X. Liu, Fei Chen, JeeHyun Hwang, and Tao Xie. Designing Fast and Scalable XACML Policy Evaluation Engines. *Computers, IEEE Transactions on*, 60(12):1802 –1817, dec. 2011. ISSN 0018-9340. doi: 10.1109/TC.2010.274.

S. Marouf, M. Shehab, A. Squicciarini, and S. Sundareswaran. Adaptive Reordering and Clustering-Based Framework for Efficient XACML Policy Evaluation. *IEEE Transactions on Services Computing*, 4(4):300–313, 2011. ISSN 1939-1374. doi: 10.1109/TSC.2010.28. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5467030.

Philip L. Miseldine. Automated XACML policy reconfiguration for evaluation optimisation. In *Proc. Fourth international workshop on Software Engineering for Secure Systems (SESS '08)*, pages 1–8. ACM, 2008. ISBN 978-1-60558-042-5. doi: 10.1145/1370905.1370906. URL http://portal.acm.org/citation.cfm?id=1370906.

Multiple resource profile of XACML2.0. Multiple resource profile of XACML2.0, Feb 2005. URL http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-mult-profile-spec-os.pdf.

Canh Ngo, Marc X Makkes, Yuri Demchenko, and Cees de Laat. Multi-data-types Interval Decision Diagrams for XACML Evaluation Engine. In *Proc. 11th International Conference on Privacy, Security and Trust (PST 2013)*, 10–12 July 2013. URL https://staff.science.uva.nl/t.c.ngo/publications/201301-midd.pdf.

OASIS XACML-TC. XACML 3.0 core schema, 2010. URL https://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd.

Ömer Malik Ilhan, Dirk Thatmann, and A. Küpper. A Performance Analysis of the XACML Decision Process and the Impact of Caching. In *2015 11th International Conference on Signal-Image Technology Internet-Based Systems (SITIS)*, pages 216–223, Nov 2015. doi: 10.1109/SITIS.2015.83.

Asela Pathberiya. Balana the open source XACML3.0 implementation, Aug 2012. URL http://xacmlinfo.org/2012/08/16/balana-the-open-source-xacml-3-0-implementation/.

Santiago Pina Ros, Mario Lischka, and Félix Gómez Mármol. Graph-based XACML evaluation. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 83–92, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1295-0. doi: 10.1145/2295136.2295153. URL http://doi.acm.org/10.1145/2295136.2295153.

Margaret Rouse. XACML (eXtensible Access Control Markup Language) definition, Sep 2005. URL http://searchcio.techtarget.com/definition/XACML.

Pierangela Samarati and Sabrina De Capitani di Vimercati. Access Control: Policies, Models, and Mechanisms. pages 137–196, 2001. URL http://dl.acm.org/citation.cfm?id=646206.683112.

Bernard Stepien, Amy Felty, and Stan Matwin. A Non-technical User-Oriented Display Notation for XACML Conditions. In Gilbert Babin, Peter Kropf, and Michael Weiss, editors, *E-Technologies: Innovation in an Open World*, volume 26 of *Lecture Notes in Business Information Processing*, pages 53–64. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-01186-3. doi: 10.1007/978-3-642-01187-0_5. URL http://dx.doi.org/10.1007/978-3-642-01187-0_5.

Endi Sukaj. choice for best JVM web frameworks, May 2015. URL http://www.slant.co/topics/439/~jvm-web-framework.

Fatih Turkmen and Bruno Crispo. Performance evaluation of XACML PDP implementations. In *Proc. 2008 ACM workshop on Secure Web Services (SWS '08)*, pages 37–44. ACM, 2008. ISBN 978-1-60558-292-4. doi: http://doi.acm.org/10.1145/1456492.1456499.

OASIS XACML-TC. eXtensible Access Control Markup Language(xacml) version 2.0, Feb 2005. URL https://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

OASIS XACML-TC. eXtensible Access Control Markup Language, January 2013. URL http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

XACML1.0-draft. Xacml (eXtensible Access Control Markup Language) definition, Apr 2009. URL http://xml.coverpages.org/XACML-v30-HierarchicalResourceProfile-WD7.pdf.

XACML3.0-specification-draft. Xacml (eXtensible Access Control Markup Language) definition, Jul 2011. URL http://securesoftwaredev.com/2011/07/21/xacml-3-0-is-now-committee-specification-draft/.

XACML3.0 vs XACML2.0. Difference between XACML2.0 and XACML3.0, Apr 2012. URL https://wiki.oasis-open.org/xacml/DifferencesBetweenXACML2.0AndXACML3.0.

List of Acronyms

MVC

Model View Controller. 33, 34

O

OASIS

Organization for the Advancement of Structured Information Standards. 10, 17, 55

P

P2P

Peer-to-Peer. 17

PAP

Policy Administration Point. 12

PARPACS

Performance Analysis, Reporting and Prediction of Access Control Systems. 6, 75

PDP

Policy Decision Point. v, vii, 2, 6, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 54, 55, 60, 62, 63, 65, 66, 67, 68, 69, 70, 71, 72

PEP

Policy Enforcement Point. 12, 14

PIP

Policy Information Point. 12

R

RBAC

Role-Based Access Control. 9, 10

REST

Representational State Transfer. 34

S

SAML

Security Assertion Markup Language. 11

SFI

Science Foundation Ireland. v

SOA

Service Oriented Architecture. 17

STACS

Scalability Testbed for Access Control Systems. vii, 2, 4, 5, 6, 18, 21, 23, 34, 54, 60, 62, 63, 65, 66

T

TSSG

Telecommunications Software and Systems Group. v, vii

U

UML

Unified Modeling Language. 28

W

WAN

Wide Area Network. 56

X

XACML

eXtensible Access Control Markup Language. xiii, 2, 4, 6, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 51, 55, 56, 57, 63, 65, 66, 72

XACML2.0

eXtensible Access Control Markup Language Version 2.0. vii, xiv, 5, 7, 19, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 65, 66, 67, 68, 69, 70, 72, 75

XACML3.0

eXtensible Access Control Markup Language Version 3.0. vii, xiv, 4, 5, 6, 7, 19, 54, 56, 57, 58, 59, 60, 61, 63, 65, 66, 67, 68, 69, 70, 71, 72, 74, 75

XML

Extensible Markup Language. 10, 19, 21, 22, 57