A Modularity and Extensibility Analysis on Authorization Frameworks

E. M. Guerra, J. O. Silva, and C. T. Fernandes

Abstract — **Authorization in its most basic form can be reduced to a simple question: "May a subject X access an object Y?" The attempt to implement an adequate response to this authorization question has produced many access control models and mechanisms. The development of the authorization mechanisms usually employs frameworks, which usually implements one access control model, as a way of reusing larger portions of software. However, some authorization requirements, present on recent applications, have demanded for software systems to be able to handle security policies of multiple access control models. Industry has resolved this problem in a pragmatic way, by using the framework to solve part of the problem, and mingling business and the remaining authorization concerns into the code. The main goal of this paper is to present a comparative analysis between the existing frameworks developed either within the academic and industry environments. This analysis uses a motivating example to present the main industry frameworks and consider the fulfillment of modularity, extensibility and granularity requirements facing its suitability for the existing access control models. This analysis included the Esfinge Guardian framework, which is an open source framework developed by the authors that provides mechanisms that allows its extension to implement and combine different authorization models.**

Keywords— **Software architecture, Access control, Authorization, Metadata-based frameworks, Decoupling, Metadata, Security, Software development, Software engineering.**

I. Introduction

ACCESS control is usually referred to as a broader term that includes authentication and authorization procedures. The former can be defined as a procedure that confirms if the subject is who it claims to be. The latter can be defined as a procedure that verifies if the subject has the right privileges to access a certain object. Even though both types of access control procedures are interesting to investigate, the focus of this work is on the analysis of authorization mechanisms architectures.

During our research, we noticed that many of the existing access control mechanisms used for developing industry applications tend to offer more features for authentication, limiting the authorization procedures to fewer ones, usually bounded to authorization based on roles. While it is understandable for a great amount of effort to be used to prevent the entrance of intruders into a system, it is still very important to control the authorization concerns. Analogously, a person can be allowed (authenticated) to enter a building, but it is still very important to control which floors or rooms this person is allowed (authorized) to go and under what circumstances.

Online systems can be cited as an environment in which the importance of access control has greatly increased. Software has been increasingly being made available through web services, requiring the control of authorization aspects of how these services are going to be consumed.

Since the very early days of Software Engineering, mechanisms have been developed on software systems to provide effective authorization procedures. However, mingling the implementation of the authorization rules with business concerns has proven to be ineffective regarding some software design principles, such as modularity, extensibility, reuse, cohesion, code readability, and testability. The use of the traditional object-oriented paradigm alone does not solve the issue adequately, mainly because authorization has a crosscutting nature.

The use of authorization frameworks are one of the current academic and industry answers to this issue, because they allow different levels of code reuse, extensibility, and modularity. For an object-oriented developer, the usage of these authorization frameworks implies in a learning process of how to use each one of them. From the point of view of the framework developer, it is vital that the application developer can use their features without major complications, being able to focus mainly on business tasks.

Additionally, the choice of an authorization framework ordinarily implies that it will be bound to some specific access control model, and once this choice is made, it becomes difficult to incorporate new authorization requirements belonging to others access control models. In frameworks that uses a more granular access control mechanism, another problem happen in parts of the system where more simple models could be used, because it is bounded to an access control model that is more complex than necessary

In this matter, the existing authorization frameworks underachieve requirements of separating business from authorization concerns appropriately. As a consequence, developers have to improvise and craft solutions for more

E. M. Guerra, Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, SP, Brasil, eduardo.guerra@inpe.br

J. O. Silva, Pontificia Universidade Católica de São Paulo (PUC-SP), São Paulo, SP, Brasil, silva.o.jefferson@gmail.com

C. T. Fernandes, Instituto Tecnológico da Aeronáutica (ITA), São José dos Campos, SP, Brasil, clovis@ita.br

complex authorization scenarios [1], or to use complex rules definition in simple scenarios making its management unnecessarily hard. In practice, it leads to applications where the authorization is coupled to the business code, a clearly undesirable situation [1]. Additionally, the existing authorization mechanisms leave developed applications coupled to their architecture, technology, or access control models, making the resulting application – once coded – more difficult to evolve [10].

The goal of this paper is to present a comparative analysis on existing authorization frameworks, focusing on modularity, granularity and extensibility requirements. As part of this work contribution, it presents development-guiding principles that can be used to perform the design and analysis of such kind of framework. Additionally, this paper presents a solution implemented by the authors called Esfinge Guardian, which is a framework that allows its extension to implement different access control models and particular authorization requirements, not coupled to any particular technology or architecture.

This research work can be considered an expansion of one previous work of ours [30], where Esfinge Guardian was presented, however this one has a strong focus on the frameworks comparison analysis. More specifically: (*i*) we provide more information on the theoretic background presenting the access control models $UCON_{ABC}$ and RAdAC; (*ii*) we establish development guiding principles for extensible and decoupled authorization models; (*iii*) we expand the Section on the use of the Esfinge Guardian; (*iv*) we propose a development guiding principles to provide a baseline for comparison analyses among authorization frameworks, specifically on extensibility and decoupling features.

This work is organized as follows: Section II provides the theoretic background on access control models and frameworks. Section III formalizes the problems of the current authorization framework implementations. Section IV schematizes a motivating authorization scenario. Section V presents how each of the main authorization solutions implements the access control policies of Section IV. Section VI makes a deeper analysis on academic and industry authorization frameworks, highlighting differences related to modularity and extensibility. Section VII concludes the paper, highlighting the main points and contributions of this work.

II. Authorization in Access Control Frameworks

One way to understand authorization is as the accurate management of three parameters: subject, resource, and privileges [1]. That is, for a resource to be accessed, a subject must have the right privileges. In fact, authorization is the process that ensures that resources are only made available to authorized subjects, and the selection of the privileges that each subject should have brings the notion of access control policies [3].

Access control policies are a key concept in the construction of an access control mechanism. They are here defined as "high-level requirements that specify how access is managed and who may access information under what circumstances" [2]. One example is: "Only account managers can credit money into a client's account". Clearly, computers cannot understand policies as high-level requirements. When translated into a format that programs can understand, authorization policies become digital policies [8].

The literature makes a distinction about the moments of creation and use of authorization policies. Privilege management is the process that creates and manages attributes and policies that are used by the access control [2]. Access control is the process responsible for the enforcement of policies and rules [4].

The effective enforcement of policies usually requires a mechanism, due to the complexity involved. Mechanisms must enforce system policies for every subject request to protected resources [16]. However, in many scenarios, in order to implement a mechanism, it is necessary to firstly design a model [1]. It is reasonable to think that a mechanism depend on a model. More precisely, access control models are mathematical formalizations of the security properties of a system, which are used to describe and, in some cases to prove these properties [1]. Models enter to bridge the gap between policies and mechanisms [2].

In application development, authorization mechanisms are many times implemented as frameworks, which can be considered as incomplete pieces of software with some special points that can be specialized to add application-specific behavior [5]. Frameworks' extension points are called hot spots [6], which are the points that applications use for customization. Each kind of behavior that a framework can execute is called variability [6].

Frameworks that base their logic decisions on the class metadata that they are working with are called metadata-based frameworks [5]. In this type of framework, a class needs to contain additional metadata so that the framework can consume, process, and make the decision for which variability to follow. In the context of object-oriented programming, metadata is information about the program structure itself such as classes, methods, and attributes [5].

Metadata-based frameworks' decoupled approach has represented an important facet in the reduction of coupling between the framework and business application concerns. As a general rule, it can be said that the more decoupled an approach the more general the types of algorithms it can execute [5]. This kind of framework can be applied for crosscutting functionality [28], such as authorization.

The main variability that access control frameworks need to handle is what rules should be enforced in each point of the system. The access control mechanism should also be able to prevent the execution of the functionality when the authorization is not confirmed. Some frameworks use metadata as an approach to configure the security rules related to a code element, such as a class or a method [6].

Different needs and contexts have led the development of many access control models and a plethora of access control mechanisms [16]. For brevity, this work only discusses some of the classical methods of authorization, but the architectural

model here presented is general enough to contain other authorization methods. Authorization methods refer to both access control mechanisms and access control models [2]. The following sub-sections provide a brief view about the classical access control methods implemented by the main security frameworks.

A. Identity-based Access Control (IBAC)

Due to the immense diversity of access control models, some works condense many access models into a category called IBAC [2][3][4], which despite essentially different among themselves, they share in common that privileges are somehow associated to the identity of the subjects.

In terms of policy enforcement, IBAC mechanisms tend to be relatively simple, as long as they handle simple policies [2]. Their drawback is when the number of resources grows too much [9], because it became problematic to privilege management. In a company of thousands of employees, it is difficult to centrally manage the creation and attribution of privileges for this huge number of resources. Scalability problems, like the previous example, were among the main reasons why it was advocated for the adoption of RBAC worldwide [7].

However, this access control model is suitable for applications where the privilege management is distributed among the users. For instance, when users own resources and can control the access to them. Nevertheless, nowadays social networks' access control model fit into this category, where a user can define who is allowed to access each of her resources [24][25], such as files, photos and information.

B. Role-based Access Control (RBAC)

RBAC introduces the concept of accessing resources mediated by roles. A role is a set of related privileges, normally equivalent to a function performed by someone in an enterprise organization. Instead of having the privileges bounded directly to subjects, they are attributed to roles [10]. Roles are attributed to subjects. The inclusion of this level of indirection immensely facilitates privilege attribution, for all a privilege administrator has to do is to set a person up with a role.

Although very efficient in the representation of hierarchies, such as companies and organizations, RBAC presents difficulties in the representation of other contexts. As an example, consider a global organization with branches in many countries. It could be necessary to divide the Information Technology team into multiple sub-teams, each team in one country administrating local resources. The creation of the one role Administrator for all sub-teams would not be adequate since each sub-team must only have access to their local resources. This is known as the least privilege principle [8]. One solution adopted by companies is to create as many roles as the number of sub-teams. However, this practice may lead into the role explosion problem in some cases, when the number of roles to be created is too numerous [9].

Another issue is that RBAC is not much adaptable to situations that demand change according to dynamic factors. The case of a hospital system illustrates the point. Information about patients is confidential by law and ethical reasons. Only the designated medical doctor must have access to the patient information. However, there are cases in which other doctors must attend to the patient for a matter of urgency. In these cases, doctors must have access to the patient information, but RBAC does not inherently deals with contextual and dynamic authorizations.

In fact, the literature does document RBAC variations built for dealing with dynamic situations such as Rule-based Access Control model (RuBAC) [3]. RuBAC is essentially RBAC that makes use of rules to create and manage roles. However, there are those who consider that these types of adaptation change the essence of RBAC, turning it into ABAC in disguise [2].

These sorts of situations – beyond the proposed scope of RBAC – are normally resolved in industry by embedding the additional access control policies into the application code [11][12][13].

C. Attribute-based Access Control (ABAC)

ABAC introduces the notion of access control based on the attributes of the subject, environment, and resources [14]. It still does not have a formal definition and its description can differ in the access control literature. For our purposes, ABAC will be defined as "access control based on attributes and policies. Attributes are distinguishable characteristics of users or resources, conditions defined by an authority, or aspects of the environment, and policies specify how to use attributes to determine whether to grant or deny an access request" [2].

Because it is based on the attributes of authorization entities, ABAC is generally said to be a fine-grained access control. It also includes the environment as part of the authorization, allowing rules to depend on other factors. It is worth mentioning that whatever access control can be defined with IBAC or RBAC can also be defined with ABAC [2].

Consider a hypothetical fine-grained policy: "Only account managers of level 2 can give credit to their clients during the working time". The problem with fine-grained policies is that they do not fit into the categories of IBAC and RBAC models.

ABAC mechanisms do not need to know the subject identity to authorize an operation. Instead, they rely on the attributes that the request proves to have [15]. In the case of the previous example policy, the request to give credit operation must contain that the requester is a level-2-manager, and the request time to be within the working time.

The ABAC's downside is that its fine-grained management increases considerably the complexity in the management of authorization policies [15], demanding a great effort to define and maintain the semantics of attributes in the enterprise.

D. Policy-based Access Control (PBAC)

Although PBAC [3] is often cited as a different access control model, it is essentially ABAC with a few differences [16]. The question about why it was necessary to create a slightly different model than ABAC naturally arises. The reason is that using ABAC in its pure form does not offer any

means of standardization in the communication of the attributes [18].

Consider an attribute called organization-name. It may happen that in one company the value of this attribute is "Aeronautical Institute of Technology", while in the other it could be "A.I.T.". Another issue is when enterprises use the same attribute name for different things, introducing the problem of name collapse [3].

In order to standardize the communication of attributes, OASIS' has created a standard named eXtensible Access Control Markup Language (XACML) [17], which is a general purpose language in XML for the declaration and communication of digital access control policies. Since then, XACML has become the de facto standard for writing fine-grained access control applications [18].

E. $UCON_{ABC}$

Traditionally, access control models focus on protecting resources on the server side and do not deal with client-side controls for locally stored digital information. Additionally, the advent of public-key infrastructure has allowed the authorization of subjects using models categorized as trust management [35]. In many cases, trust management utilizes subject properties for authorization in the form of digital credentials or certificates.

Usage Control (UCON) is a notion, a conceptual framework, introduced to be comprehensive enough to encompass traditional access control, trust management, and DRM [34] The term has some connotations, which reference [33] present them: "*In the DRM context, it conveys the sense that digital content is provided for use of the end-user's system, but the provider would like to retain some control over what the user does with the bits. In the privacy context the situation is reversed. It is the end-user who often provides personal information to a service provider, and would like to control how the service provider can use that information. Sometimes the personal information is provided by a third-party originator, say a health-care provider, but the individual, called 'identifiee', to whom it pertains, would nevertheless like to exercise control over its use. Usage also has a connotation of duration, so the access may continue for some time.*"

We can see some new concerns for authorizations. One example for access control in the DRM context is the re-distribution of a music file (e.g.: MP3) once it has been bought from a service provider. The service provider may be able to retain the right of distribution from the end user. One example for access control with duration, suppose an application that must control the use of prepaid mobile phone. In this case, even if the subject (user) is authorized to complete the phone call, the application must continuously check if the subject still has the credits for continuing the call.

Park and Sandhu [33] not only use the concept of Authorization (A), but also introduce oBligations (B), and Conditions (C), integrating them into the conceptual framework, forming the $UCON_{ABC}$ access control model. Obligations are requirements that have to be fulfilled for allowing access. Conditions are environmental or system requirements – related to resources – that have to be satisfied for access.

F. Risk-Adaptive Access Control (RAdAC)

RAdAC is an emerging access control model that takes into account risks to grant resources, being used basically in contexts that demand large-scale computing. The RAdAC model represents the cutting-edge model envisioned for the new contexts of grid and cloud computing [31].

In a world that is each day more interconnected, a differentiation in the access control must be made beyond roles, attributes and identities [32]. Risks must be taken into account. An example is the netbanking services that we do customarily. The risks of accessing the netbanking services from a trusted PC are different from the ones we take on accessing the same services from an untrusted PC.

RAdAC is still a very recent model that needs much research on it. Hu *et al* [3] have proposed a formal framework – at a policy layer – in terms of components and their interactions to develop abstract models for RAdAC.

G. Hybrid Authorization Models

Despite each model focus on solving the authorization problem for a given scenario, real applications can have needs that are not solved by a single model. In these cases, it is important to combine models in order to fulfill the authorization requirements.

For instance, imagine a military application where each user has a role in a military organization, but the documents also have a sensitivity that requires a certain privilege level from the subject. In this scenario, in order to access a given document, the user should have the appropriate role, the document should be related to the organization where he is allocated and he should have the minimum privilege level. Based on this example, it is possible to see that different models can be appropriate for different authorization requirements.

In such hybrid scenarios, by using a more restrictive model, such as RBAC, it does not cover all the authorization requirements. However, a more general model, such as ABAC, can be hard to manage for rules that fit better on other models. A possible solution in such scenarios is to combine authorization models, using each ones for the scope where it is more appropriate. In [29] there is an example of an authorization model that combines characteristics of RBAC and ABAC, creating what it calls a Contextual Authorization Model.

III. Problems in Existing Authorization Frameworks

The basic premise of access control mechanisms is that authorizations can be enforced in terms of subjects accessing protected resources in a particular environment. In the application development world, access control mechanisms are many times implemented as frameworks. It is noteworthy that the main security framework developers already provide

them as metadata-based frameworks.

Although there is not much debate about the importance of authorization, there is not still a general solution that decouples business from authorization concerns, except for simpler authorization policies. For more complex ones, the existing security frameworks fall short on offering tools that can be used declaratively, necessarily forcing developers to craft solutions tangling business with authorization codes.

The existing authorization frameworks offer rudimentary coding tools – or none at all – for software customization such as Spring Framework, Java EE, and Axiomatics XACML. These frameworks are each restricted, in the best cases, to a few access control models – usually RBAC –, but still far from providing means for extending to other authorization models. In other words, they have little or non-existent extensibility. The work of building an exhaustively complete access control mechanism that comprised all the possible user needs would be an impossible one. Therefore, extensibility must have a high priority in the design of an authorization framework architectural model.

To our best knowledge, there are not works that research why access control developers do not invest more in ABAC systems. However, a NIST report mentions that it is because its many-to-many relationships are difficult to represent [2]. It also states that the lack of more complex mechanisms maintains enterprises using RBAC solutions, leaving the ABAC ones on the horizon for most organizations.

Another issue about some current authorization solutions is in the technology dependence for its instantiation. Usually, these frameworks are coupled to some specific architecture or other frameworks. For instance, Java EE authorization solution can only be used in application containers and Spring Security can only be applied to objects managed by the Spring Frameworks, which can limit its application to a small set of applications.

An general architectural model for authorization frameworks is important because it provides ways to adequately separate concerns such as code tangling and technology dependence, representing an important step in the move from programmatic solutions to declarative ones. If a framework can include other framework solutions as its own, we say that such a framework is extensible. If a framework can be plugged in applications independently from its architecture and from the frameworks that it uses, we say that such a framework is technology independent.

IV. A MOTIVATING AUTHORIZATION SCENARIO

This Section schematizes a reasonable access control scenario and it aims to show – at the next Section – how each access control mechanism implements access control policies. This approach helps to create a common baseline for comparison of access control solutions and to create a more concrete view of each implementation.

A. The Scenario Access Control Policy

Consider the following hypothetical access control policy:

"*Any management position can oversee the operations performed by any of its subordinates, but must be restrained of overseeing the operations of their peers, their peers subordinates, and any superior position in the bank management hierarchy. In addition, the oversee operations function must only be accessed from within the perimeters of the bank facilities*."

B. Bank Career Hierarchy

For a richer comparison scenario, Fig. 1 defines a career bank hierarchy. This organizational chart is hypothetical but we consider it to be within the limits of reasonable.

According to the bank access control policy, a manager can oversee operations of Clerks and Officers as long as they are their direct subordinates, but cannot oversee operations of other Managers, Senior Managers or any other position above in the hierarchy. Also, all accesses must be made within the bank facilities.

C. Access Control Policy Rationale

This authorization scenario is composed of elements belonging to different access control models. For instance, the organizational chart is made of roles, each having a set of operations that they can execute. This indicates the use of RBAC.

In addition, the access control policy mentions the oversee operation, which has hierarchical features, which makes the policy in compliance with the MAC model [23] or at least with some type of hierarchical RBAC.

Finally, by limiting the execution of oversee the subordinate operations to the inside of the bank facilities, the access control policy uses elements of the ABAC/PBAC model, because it depends on the access context and not only on the subject and object. The requirement that restrict this authorization to its subordinates also is related to this access control model.

In this fashion, despite the apparent simplicity, the access control policy can be considered a complex one, from the point of view of the modularization of authorization concerns.

V. HOW EACH AUTHORIZATION FRAMEWORK IMPLEMENTS THE SCENARIO ACCESS CONTROL POLICY?

The objective of this Section is to present how each of the main existing solutions implements the access control policy presented in Section IV. Since our focus is on solutions ready to be used on industry projects, we are going to focus on mature solutions that are accessible to use.

We have selected three of the main security industry frameworks, that is: Java EE Security Framework; Spring Security; and Axiomatics XACML. For each one of them we present a possible implementation of the security policy of Section IV. The Esfinge Guardian framework is also included for comparison. Despite Esfinge Guardian is proposed by the authors of this work, it is important to highlight that it is open-source, have a comprehensive tutorial and a good automated test coverage, being ready to be used on real applications.

For better visualization, as a convention, we stress the authorization related code in bold on the code examples.

Java EE Security

The Java EE platform is the current industry standard for building enterprise Java applications. It defines an API that aims to simplify enterprise code development and in the meantime to robust Authentication, Authorization, Confidentiality, Non-Repudiation, Auditing, and Quality of Service. The Java EE security model has the means to secure the Web Tier, Enterprise JavaBean (EJB) Tier, and the Enterprise Information System (EIS) Tier. For the sake of comparison, this research only analyzes authorization in service tiers (i.e. EJB tier).

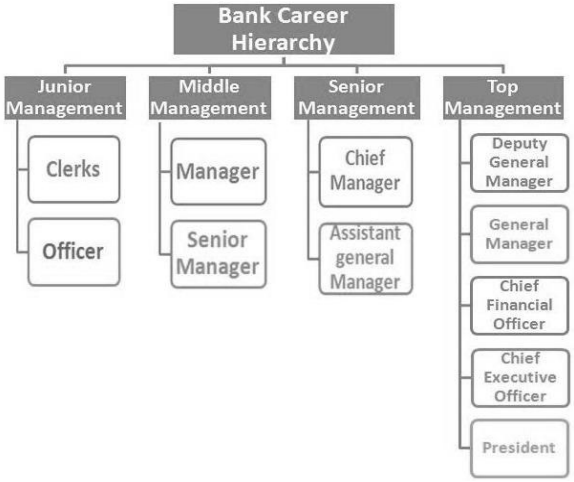

Figure 1. A hypothetical bank career hierarchy

The Java EE platform provides two ways of securing an application: declaratively and programmatically. There is in fact a recommendation in their tutorial for the declarative security, which can be done via XML descriptors files or via framework annotations.

The Java EE reference access control model is the RBAC. However, for anything with more complexity, they offer programmatic security. Listing 1 exemplifies a possible implementation of the bank authorization policy of Section IV. Java EE offers the `@DeclareRoles` annotation for specifying the management roles.

In the actual implementation, this line contains all the roles of the bank hierarchy. However, the only method made available by the platform is the `isCallerInRole()`, that the developer can use when s/he wants to discover is a certain subject belong to a known role. Since the authorization of the calling subject is dynamically relative to its own role, this method does not solve the issue alone. That is the reason why another proprietary method called `canOverseeRole()` is used. This method does not only has the responsibility of discovering which is the role of the calling subject, but also to determine its own role is equal, lower, or higher than the object's role (the other employee).

```
@Stateless
@DeclareRoles("{MANAGER,CLERK,OFFICER,…}")
public class EmployeeServImpl implements EmployeeServ{

  @Resource SessionContext ctx;

  public EmplOps[] overseeAllOps(
          EmplInfo info, CallerLocation cl) {
    boolean canOversee = canOverseeRole(ctx,info);
    boolean isSub =
       info.isSubordinate(ctx.getCallerPrincipal());
    if (!canOversee || !isSub || !cl.isInside()){
      throw new SecurityException(…);
    }
    EmplOps[] empops = //logic for retrieving data
    return empops;
  }

  private boolean canOverseeRole(
      SessionContext ctx, EmplInfo info) {
    //find user role using ctx.isCallerInRole(String)
    //return if this role can oversee the employee role
  }
}
```

Listing 1. A possible authorization policy implementation in Java EE 6.

Another verification that needs to be performed according to the requirements is if the employee is a subordinate from the current user. In Java EE platform, the method `getCallerPrincipal()` can be used to retrieve the current user registered in the session. This information can be used as a parameter to perform this check, which also needs to be done declaratively.

Spring Security

Spring Security is a popular security framework that has the same goals as Java EE, except that it is much more modular and lighter. Spring Security is designed to handle authentication and authorization requirements. In their tutorial [20], they cite four types of security concerns that the framework addresses: (*i*) authentication; (*ii*) web request security; (*iii*) service layer; and (*iv*) domain object security. In this research we focus on (*iii*) and (*iv*) security concern.

Considering the extensibility and modularity capabilities of each framework, Spring Security 3.X represents an enhancement when compared to Java EE 6. Although heavily based on the RBAC model, the Spring framework offers the possibility of accessing the application beans declaratively, through the use of authorization annotations.

As can be seen in Listing 2, by implementing the spring interface `PermissionEvaluator` it is possible to decouple authorization code from business code, except for the framework annotation declared on the business method, such as shown in Listing 3. There is also some configuration to bind these classes in the XML descriptors that we have omitted.

```
public class BankPermissionEvaluator implements
                              PermissionEvaluator {
  @Override
  public boolean hasPermission(Authentication auth,
      Object uid, Object pid) {
    return isHierarchyCompliant(auth, uid, pid)
      && isWithinFacilities(auth, uid)
      && isSubordinate(auth, uid, pid);
  }
  private boolean isHierarchyCompliant(
      Authentication auth, Object uid, Object pid) {
    allow = /* allow based on hierarchy */
    return allow;
  }
```

```
  private boolean isWithinFacilities(Authentication auth,
      Object uid) {
    boolean allow = /*allow based on location*/
    return allow;
  }
  private boolean isSubordinate(Authentication auth,
      Object uid, Object pid) {
    allow = /* allow if he is a subordinate */
    return allow;
  }
}
```

Listing 2. A possible implementation of the authorization policy using Spring Security.

```
@PreAuthorize("hasPermission(#subject, #target) and
        hasPermission(#subject, #cl)")
public EmplOps[] overseeAllOps(EmplInfo target,
      CallerLocation cl)
  EmplOps[] empops = //logic for retrieving data
  return empops;
}
```

Listing 3. Business method protected using Spring Security.

This is how the authorization would work. When business method `overseeAllOps()` is invoked, the framework would intercept the operation using aspect-orientation [26], and redirect the flow to the code presented in Listing 3. The access would be granted only if the method `hasPermission()` present on `BankPermissionEvaluator` return true.

Axiomatics XACML

Axiomatics XACML is the current most popular XACML access control platform, and it significantly facilitates the development of fine-grained applications [37]. It standardizes three essential aspects of the authorization process: policy language; XACML request/response protocol; and reference architecture. Axiomatics XACML can be seen as an implementation of the ABAC model, or more precisely, of the PBAC model.

One of the main advantages of this mechanism is the structured standardized use of external authorization. By providing a standardized language for writing authorization policies, it is possible to apply the mechanism into multiple tiers, having only one authorization policy description. This increases the separation of concerns, therefore augmenting flexibility.

The XACML language can be very fine-grained, being able to express a significant amount of authorization scenarios and access control models. One example is that the language can express hierarchical relationships between roles – unlike the other previous presented solutions. Listing 4 presents a code snippet that uses Axiomatics XACML to send a request to evaluate an authorization rule of the defined policy.

```
public boolean isEmployeeAllowed(EmplIinfo info,
    CallerLocation cl) throw SecurityException {
  try{
    //Create the connection to the service;
    ConnectionInterface pep = new MetroPEPModule();
    Properties config = new Properties();
    config.load(new FileInputStream(
      new File("connection.properties")));
    pep.setupConnection(config);

    //create XACML request
    SimpleRequestWrapper r = new SimpleRequestWrapper(4);
    r.addSubjectAttribute(URI.create("location", cl));
    r.addSubjectAttribute(URI.create("role",
      subj.role()));
    r.addActionAttribute(URI.create("action-id",
      "read"));
    r.addResourceAttribute(URI.create("resource",
      info.role()));

    //Send the request and handle response
    SimpleResponseWrapper resp = pep.evaluate(r);
    return resp.isPermit();
  } catch(Exception e) {
    throw new SecurityException(e);
  }
}
```

Listing 4. Business method protected using Spring Security.

Listing 5 shows one possible implementation of the business method, using solely Axiomatics XACML. It is important to note that this solution does not define how the authorization mechanism is plugged in the application.

```
public EmplOps[] overseeAllOps( EmplInfo info,
    CallerLocation cl) throw SecurityException {

  boolean allow = isEmployeeAllowed(info, cl);
  if(allow){
    EmplOps[] empops = //logic for retrieving data
    return empops;
  } else {
    throw new SecurityException("Access Denied");
  }
}
```

Listing 5. A possible implementation of the business method using Axiomatics XACML.

Esfinge Guardian

The Esfinge Guardian framework is an extensible authorization framework, fully capable of being used in the development of any business application. Among its benefits we can include the complete separation of business and authorization code.

The Esfinge Guardian framework can be seen from at least two perspectives. Since the framework completely separates business from authorization concerns, the implementation of the authorization logic can be delegated to experienced developers, usually the ones with technology and business domain background for creating and implementing business security rules. On the other hand, once the framework has been extended, it can be used by the other members of the development team, which can be composed of less experienced people.

Esfinge Guardian provides the application developer with tools for attacking traditional development problems without compromising its simplicity. Two design decisions are responsible for the simplicity of the framework: (*i*) Esfinge Guardian is a metadata-based framework that allows metadata schema extension, fully capable of adapting its internal algorithm based on the declared metadata associated with the protected operations [30]; and (*ii*) the use of Domain Annotations [21][22] allows the abstraction of complex

authorization policies, factoring them with business domain terminology.

The Esfinge Guardian framework contains ready-to-use metadata elements and component implementations for some of the classical access control models: RBAC, ABAC, and MAC. These implementations can be used and combined to represent an expressive number of authorization scenarios. Nevertheless, the framework can be easily extended to implement new authorization models that can be plugged in an application through custom metadata elements, working the same way as the existing implementations.

As with the previous authorization frameworks, we start showing the class that contains the authorization code, which can be seen in Listing 6. Esfinge Guardian links an authorization annotation, `@RespectHierarchy`, to an Authorizer class, `HierarchyAuthorizer`. This binding is done by using the annotation `@AuthorizerClass` in the definition of the authorization annotation, as presented in Listing 7.

```
public class HierarchyAuthorizer implements
    Authorizer<RespectHierarchy> {
  public Boolean authorize( AuthorizationContext ctx,
      RespectHierarchy rh) {
    Set<String> roles = ctx.subject("roles");
    //retrieve other relevant information from ctx
    return //hierarchy authorization logic;
  }
}
```

Listing 6. A possible implementation of the hierarchy authorization policy using Esfinge Guardian.

```
// Retention and ElementType suppressed
@AuthorizerClass(HierarchyAuthorizer.class)
public @interface RespectHierarchy {
}
```

Listing 7. Binding authorization annotation with the respective implementation.

A similar structure composed by an annotation and an authorizer class can be created to define the other rules from the bank authorization policy. For brevity, the code for the other annotations, `@WithingHQ` and `@SubodinateOnly`, and their respective authorizers are omitted. These three authorizer annotations can be added to a business method it is compliant with the authorization policy. Listing 8 presents a possible implementation of the business code.

```
@RespectHierarchy
@WithinHQ
@SubodinateOnly
public EmplOps[] overseeAllOps(EmplInfo info,
    CallerLocation cl) {
  EmplOps[] empops = //logic for retrieving data
  return empops;
}
```

Listing 8. A possible implementation of the business method using Esfinge Guardian.

In the described context, what Esfinge Guardian does is: (*i*) to intercept transparently the `overseeAllOps()` method call; *(ii)* to recognize that `@RespectHierarchy`, `@WithinHQ` and `@SubodinateOnly` are authorization annotations; (*iii*) to populate the authorization context with subject, object and environment information; (i*v*) to execute the authorization logic to verify if the security conditions to execute the method are satisfied; and, finally, *(v)* proceed or not with the method execution according to the result. The authorization context populator, the authorization logic and its representation on annotations are framework hot spots, meaning that they are extensible and can be adapted according to the software system needs.

Further details on how to use Esfinge Guardian framework, or on its respective architectural model, which lays the theoretical foundations for the framework, can be found in our previous works [30] [36].

VI. ANALYZING MODARILARITY AND EXTENSIBILITY FEATURES OF AUTHORIZATION FRAMEWORKS

The purpose of this section is to offer a deeper analysis on academic and industry authorization frameworks, highlighting the main differences between them, specially related to modularity and extensibility.

Subsection A proposes a comparison baseline for comparing extensibility and modularity of authorization frameworks. Subsection B focuses on the analysis of industry frameworks and, finally, Subsection C, of academic frameworks.

A. Common Comparison Baseline

There are a considerable number of authorization solutions in the industry and in the academic world. We have selected three of the main industry and academic authorization frameworks for the Java platform. However, for other languages and platforms there are other solutions with a similar approach.

One issue is how to establish a baseline for a proper comparison among authorization frameworks. This is important because authorization frameworks are designed with different purposes, making necessary to establish the points of comparison in order to reach a fair conclusion.

Table I proposes some requirements that must be taken into account in the authorization frameworks' design. They intend to establish requirements of extensibility and modularity that are desirable on authorization frameworks [5]. It is important to highlight that this analysis focus on extensibility and modularity aspects only, disregarding other equally important quality attributes, necessary for choosing security frameworks in real projects.

TABLE I
FRAMEWORKS DESIGN REQUIREMENTS

Req Id	Description
REQ01	Authorization frameworks must provide a way for granting authorization in a fine-grained level, considering the three basic authorization entities: subject; resource; environment
REQ02	The authorization mechanism must be able to transparently intercept the subject's requests to the resources
REQ03	Authorization concerns must be completely modularized into specific isolated units
REQ04	Authorization rules cannot depend on the location of the access data for authorization
REQ05	The intersection of authorization and business concerns must be declarative, and related to the business domain

B. Industry Authorization Frameworks

This subsection presents the analysis of each industry framework presented before guided by the requirements presented on table I. The code examples presented on Section V are referenced to exemplify some points.

Java EE

The Java EE Security framework specification does make a recommendation for declarative security instead of the programmatic approach [19]. But, it is not possible to avoid the programmatic approach, except for systems with classic RBAC authorization policies. The scenario authorization policy illustrates this point: it presents a hierarchical form of RBAC combined with geolocation based component.

We present a more systematic analysis as follows:

- REQ01 is not satisfied because Java EE does not inherently offer any means for authorizations to be in the fine-grained level.
- REQ02 is not satisfied because Java EE does not provide transparent interception. Instead, the developer has to explicitly call the security procedure, or to embed in the business code. Listing 1 is an example.
- REQ03 is not satisfied because the authorization code mingles with business code in non-trivial scenarios. Listing 1 is an example.
- REQ04 is not satisfied because there is no structured way of obtaining authorization context data in different places such as files, databases, transactions objects, session objects etc. The developer has to embed this logic into the business code as well.
- REQ05 is not satisfied because all the authorization code that is written into the business code adds nothing to the business domain itself. This is also a result of coupling the framework authorization code to the business class.

Spring Security

Spring Security offer means to express authorization policies for the ABAC model, due to the use of expression languages in its annotations. More precisely, the use of expression languages allows a form of RuBAC model [3], which is essentially a simplified version of ABAC.

We present a more systematic analysis as follows:

- REQ01 is partially satisfied because even though Spring Security is able to work in the fine-grained level theoretically, the mechanism does not scale well when the amount of individual items to be protected is large [45].
- REQ02 is satisfied because Spring Security understands that the target method in the resource is protected, freeing the developer from having to explicitly call the mechanism.
- REQ03 is partially satisfied because Spring Security authorization annotations are related to the framework, not to the business itself. A complex authorization policy could make the annotations hard to read and maintain.
- REQ04 is partially satisfied. Spring Security has the concept of `Evaluators` interface, which are called in custom authorization implementations. In the best scenario, a concrete implementation of this interface could be used to search for authorization context data, which can even be injected by the Spring framework. Especially for data passed as a parameter for the method, it would imply that the search for authorization context data would mingle with authorization logic itself.
- REQ05 is satisfied because there are a considerable number of scenarios in which the Spring Security authorization annotations can carry meaning to the business, adopting some best practices on how to write the annotation.

Axiomatics XACML

Although this architecture is good for handling fine-grained authorizations, when it comes to modularity there are not much tools beyond those already provided by object-oriented programming [37].

- REQ01 is fully satisfied. XACML is currently the best mechanism for fine-grained authorization nowadays.
- REQ02 is not satisfied because the developer has to explicitly make the authorization verification request.
- REQ03 is not satisfied because in the best scenario the authorization code would be modularized into a separate method, but an explicit call would have to be made to it for the authorization to take place.
- REQ04 is partially satisfied. The authorization data can be obtained from multiple places, however the software that are instantiating the framework is responsible to retrieve this data.
- REQ05 is not satisfied because the intersection point between authorization and business code is an embedded method call to the authorization code. This requirement is about declarative, cohesive intersection with business data such as by using domain annotations.

Esfinge Guardian

Esfinge Guardian has been developed from the start with these requirements as a guide.

We present a more systematic analysis as follows:

- REQ01 is satisfied because Esfinge Guardian is capable of operating in the fine-grained level in multiple scales.
- REQ02 is satisfied because Esfinge Guardian offers not only a mechanism for transparent interception, but can also be extended to use a different one.
- REQ03 is satisfied because authorization and business concerns are completely modularized.
- REQ04 is satisfied because it has a component type called `Populator`, which allow authorization data to be retrieved by the framework from anywhere. The framework provides some implementations of this kind of component, but it can also be extended by the application to implement custom populators.
- REQ05 is satisfied because of its support for domain annotations, which should be created with business meaning. Exemplifying this practice, the annotations `@RespectHierarchy`, `@SubordinateOnly` and `@WithinHQ` created on the example are not related to Esfinge Guardian, but to the application business.

Table II summarizes the analysis.

TABLE II
SUMMARY OF AUTHORIZATION FRAMEWORK COMPARISON

Req Id	Related Topic	Esfinge Guardian	Java EE 6	Spring Security	Axio-matics XACML
REQ01	Fine-grained capability	Complete	None	Partial	Complete
REQ02	Transparent interception	Complete	None	Complete	None
REQ03	Modularization of authorization concerns	Complete	None	Partial	None
REQ04	Data location independence	Complete	None	Partial	Partial
REQ05	Cohesion with the business domain	Complete	None	Complete	None

Industry authorization frameworks

C. Academic Authorization Frameworks

Sirbi and Kulkarni [38] present a discussion on the modularization of security concerns combining the Aspect-Oriented Programming (AOP) paradigm [26][27] with the Spring Security framework. Even though the authors recognize the importance of separation of concerns in their work, they focus on showing techniques on the implementation level, detailing how to combine AOP with Spring Security. However, their approach is representative of other solutions based on AOP [39][40], which covers modularization of crosscutting concerns (REQ03), transparent interception mechanism (REQ02), and it also offers a simple form of RuBAC (REQ01). The other architectural requirements presented in Table I are not covered. AOP's interception mechanism is based on the selection of joins points by the pointcuts. Join points are well defined points in the execution of a system such as method execution, method call, attribute read, and attribute write. In the case of Spring AOP, a join point is always equivalent to a method execution. Pointcut is the mechanism that specifies which join points will link aspects and classes. The Spring AOP interception mechanism in the business layer is inherently fragile because it is mostly based on method signatures [41]. That means, if a business method signature X changes, there is no feedback mechanism informing that the modularized crosscutting concern is not being considered in X anymore [30]. Another point in this work is that it is coupled to the Spring Security framework.

Camargo [43] propose an implementation of authentication and authorization concerns in AspectJ, aiming to make them reusable for web applications based on the MVC pattern and the Struts framework. The authors have implemented various levels of authorization: class, method, and attribute level, implementing the RuBAC model. Basically, the same arguments presented for the research of Sirbi and Kulkarni [38] apply for the work of Camargo [43], being coupled to the Apache Struts and restricted to web applications.

Welch and Stroud [42] propose an architectural model for modularizing security concerns using reflective security architecture for distributed computing. They compare a third-party application secured through inheritance and the proxy pattern with a re-engineered version that uses bytecode manipulation, obtaining a code reduction and a degree of separation of concerns that is not complete. They do not provide an access control model, but focus on presenting the technique they used for the separation. However, it should have the granularity of ABAC (REQ01). This inference is necessary because we had not access to their code. Extensibility aspects are not considered, nor cohesion with business domain (REQ05). An interesting point is that they critique the use of the Proxy Pattern, which is one of the interception mechanism used by Esfinge Guardian. The authors argue that applications that rely on this pattern for interception are subject to the bypass problem, which is a variant of the confinement problem [44]. In a complex application, it is always possible that an instance of a proxied class returned by a method invocation might not be replaced with an instance of its proxy. The unwrapped instance would bypass the proxy.

VII. CONCLUSION

This paper is an extension of the one previous work of ours [30]. We provide some theoretical background, discussing the main access control models in use nowadays. In this research, we add a discussion on the RAdAC and $UCON_{ABC}$ models. In addition, we present a discussion about the current problems in the existing authorization frameworks.

A motivating authorization scenario is proposed as a baseline for the comparisons on the rest of the work. Despite contrived, we believe that the proposed authorization scenario is a reasonable one for the comparisons.

We propose an implementation of the authorization policy for each one of the main authorization industry frameworks along with Esfinge Guardian. For each framework, we tried to use the best resources made available. In the case of other approaches for implementation, an extension of this analysis can be made considering the same requirements.

We reserve a Section for analyzing the implementation decisions: strengths and shortcomings, focusing on extensibility and modularity aspects. For a fairer comparison, we propose some development guiding requirements, which must be taken into account in the development of authorization frameworks. Some academic authorization frameworks are also analyzed.

The overall development time and authorization management effort might potentially be reduced, because of the complexity reduction in the use of the authorization rules constructs, and due to the increased semantic cohesion created by the use of domain annotations.

This paper is useful for software architects, framework developers, and software developers in general, by allowing the creation of more decoupled and extensible authorization solutions. Software architects could benefit from the Esfinge Guardian Architectural Model by instantiating a version of the architecture suitable for the enterprise needs. Framework developers could benefit by extending or re-creating the Esfinge Guardian in another language or platform. Finally, software developers in general could benefit from the understanding of the techniques involved in a framework development.

REFERENCES

[1] E. BERTINO; B. CATANIA; E. FERRARI; P. PERLASCA, "A logical framework for reasoning about access control models." *ACM Transactions on Information and System Security*, v. 6, no. 1, pp. 71-127, 2003.

[2] PRIVILEGE MANAGEMENT CONFERENCE COLLABORATION TEAM. A report on the privilege (access) management workshop. Washington, DC: NIST, 2010. (NIST-IR-7657).

[3] Hu, V. C., Ferraiolo, D. F., Kuhn D. R.: Assessment of Access Control (NIST-IR-7316). Gaithersburg, MD (2006)

[4] Hu, V. C., Scarfone, K.: Guidelines for Access Control System Evaluation Metrics NIST-IR-7874. Gaithersburg, MD (2012)

[5] Eduardo Guerra, Felipe Alves, Uirá Kulesza, Clovis Fernandes, A reference architecture for organizing the internal structure of metadata-based frameworks, Journal of Systems and Software, Volume 86, Issue 5, May 2013, Pages 1239-1256.

[6] Fayad, M., Schmidt, D. C., Johnson, R. E.: Building application frameworks: object-oriented foundations of framework design. In: Building application frameworks: object-oriented foundations of framework design, New York, Wiley, 55-83 (1999)

[7] Ferraiolo, D., Kuhn R., Chandramoulli, R.: Role-based access control. Artech House (2007)

[8] Ferraiolo, D., Kuhn, R.: Role-based Access Controls. In: Proceedings of 15th NIST-NCSC National Computer Security Conference, Baltimore, MD, 554-563 (1992).

[9] Elliott, A. A., Knight, G. S.: Role Explosion: Acknowledging the Problem. In: Proceedings of the 2010 International Conference on Software Engineering Research & Practice. (2010)

[10] Sandhu, R., Ferraiolo, D.F., Kuhn, D.R.: The NIST Model for Role-Based Access Control: Toward a Unified Standard. In: 5th ACM Workshop Role-Based Access Control. pp. 47–63. (2000).

[11] Probst, S., Kung, J.: The need for declarative security mechanisms. In: Proceedings of 30th Euromicro Conference, pp. 526- 531 (2004)

[12] Merz, M.: Enabling declarative security through the use of Java Data Objects. In: Journal of Science of Computer Programming, V. 70, n. 2-3, pp. 208-220 (2008)

[13] Bartsch, S.: Authorization Enforcement Usability Case Study. In: ESSoS'11: Proceedings of the Third international conference on Engineering secure software and systems, pp. 209-220 (2011)

[14] Hai-bo, S., Fan, H.: An Attribute-Based Access Control Model for Web Services. In: PDCAT '06. Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies, pp.74-79 (2006)

[15] Peng, J., Yang, F.: Description Logic Modeling of Temporal Attribute-Based Access Control. In: ICCE '06. First International Conference on Communications and Electronics, pp.414-418 (2006)

[16] Hsieh, G., Foster, K., Emamali, G., Patrick, G., Marvel, L.: Using XACML for Embedded and Fine-Grained Access Control Policy. In: ARES '09 International Conference, pp.462-468 (2009)

[17] XACML: eXtensible Access Control Markup Language (XACML), Version 3.0, Committee Specification 01. http://docs.oasisopen.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf (2011)

[18] Bo, L, Nan, Z., Kun, G., Kai, C.: An XACML Policy Generating Method Based on Policy View. ICPCA 2008: 3rd International Confer. on Pervasive Computing and Applications, v.1, pp.295-301 (2008)

[19] Java EE: Java Enterprise Edition Tutorial 6. http://docs.oracle.com/javaee/6/tutorial/doc/bnbyl.html (2013).

[20] Spring Framework: Spring Source Community. http://www.springsource.org/ (2013)

[21] Perillo, J., Guerra, E., Silva, J., Silveira, F., Fernandes, C.: Metadata Modularization Using Domain Annotations. In: Workshop on Assessment of Contemporary Modularization Techniques. V. 3, Orlando (2009)

[22] Perillo, J., Guerra, E., Fernandes, C.: Daileon-A Tool for Enabling Domain Annotations. In: RAM-SE '09: Proceedings of the Workshop on AOP and Meta-Data for Software Evolution, n. 7 (2009)

[23] Trusted Computer System Evaluation Criteria (Orange Book), Department of Defense. http://csrc.nist.gov/publications/history/dod85.pdf (1985)

[24] Sayaf, R., Clarke D.: Access Control Models for Online Social Networks. In: Social Network Engineering for Secure Web Data and Services, (2012)

[25] R. Sayaf. Access control for online social networks - research summary. In: For your eyes only conference. Brussels. (2012)

[26] Ribeiro, M., Dosea, M., Bonifácio, R., Neto, A. C., Borba, P., Soares, S.: Analyzing Class and Crosscutting Modularity Structure Matrixes. In Proceedings of the 21th Brazilian Symposium on Software Engineering (SBES) (2007)

[27] Neto, A. C., Ribeiro, M., Dósea, M., Bonifácio, R., Borba, P., Soares, S.: Semantic Dependencies and Modularity of Aspect-Oriented Software. In: Workshop on Assessment of Contemporary Modularization Techniques (2007)

[28] Guerra, Eduardo, Buarque, Eduardo, Fernandes, Clovis, Silveira, Fábio (2013) A Flexible Model for Crosscutting Metadata-Based Frameworks. Computational Science and Its Applications – ICCSA 2013, Lecture Notes in Computer Science, V 7972, 391-407.

[29] Motta, G.H.M.B.; Furuie, S.S., "A contextual role-based access control authorization model for electronic patient record," *Information Technology in Biomedicine, IEEE Transactions on* , vol.7, no.3, pp.202,207, Sept. 2003

[30] Silva, J., Guerra, E., Fernandes, C.: An Extensible and Decoupled Architectural Model for Authorization Frameworks. In: Murgante, B., Misra, S., Carlini, M., Torre, C.M., Nguyen, H.-Q., Taniar, D., Apduhan, B.O., Gervasi, O. (eds.) ICCSA 2013, Part IV. LNCS, vol. 7974, pp. 614–628. Springer, Heidelberg (2013)

[31] Kandala, S.; Sandhu, R.; Bhamidipati, V., "An Attribute Based Framework for Risk-Adaptive Access Control Models," Availability, Reliability and Security (ARES), 2011 Sixth International Conference on , vol., no., pp.236,241, 22-26 Aug. 2011

[32] Ferreira, A.; Chadwick, D.; Farinha, P.; Correia, R.; Gansen Zao; Chilro, R.; Antunes, L., "How to Securely Break into RBAC: The BTG-RBAC Model," Computer Security Applications Conference, 2009. ACSAC '09. Annual , vol., no., pp.23,31, 7-11 Dec. 2009

[33] PARK, J.; SANDHU, R. The $UCON_{ABC}$ usage control model. ACM Transactions on Information System Security, v. 0, n. 0, February, 2004.

[34] Yonggang Ding; Junhua Zou, "DRM Application in $UCON_{ABC}$," Advanced Software Engineering and Its Applications, 2008. ASEA 2008 , vol., no., pp.182,185, 13-15 Dec. 2008

[35] Srijith K. Nair, Andrew S. Tanenbaum, Gabriela Gheorghe, and Bruno Crispo. 2008. Enforcing DRM policies across applications. In Proceedings of the 8th ACM workshop on Digital rights management (DRM '08). ACM, New York, NY, USA, 87-94.

[36] Silva, J. O. An Architectural Model for Access Control Frameworks Extensible for Different Authorization. São José dos Campos, 2013. Master's Thesis 114f.

[37] Rissanen E, Brossard D, Slabbert A Distributed access control management—a xacml-based approach. In: ICSOC-servicewave. Springer, Berlin, 2009

[38] Sirbi, K.; Kulkarni, P. J. Modularization of enterprise application security through Spring AOP. International Journal of Computer Science & Communication, v. 1, n. 2, p. 227-231, 2010.

[39] Fernandez, L. L.; Carrillo, M. G.; Pelaez, J.; Fernandez, F. A declarative authentication and authorization framework for convergent IMS/Web application servers based on aspect oriented code injection. In: IMSAA INTERNATIONAL CONFERENCE ON INTERNET MULTIMEDIA SERVICES ARCHITECTURE AND APPLICATIONS, 2, 2008, Bangalore. Proceedings… Bangalore: IMSAA, 2008. p. 1-6.

[40] HAI-BO, S. A semantic and attribute-based framework for web services access control. In: ISA INTERNATIONAL WORKSHOP ON INTELLIGENT SYSTEMS AND APPLICATIONS, 2, 2010, Wuhan. Proceedings… Wuhan: ISA, 2010, p.1-4.

[41] Silva, J. Frameworks orientados a aspectos baseados em metadados. São José dos Campos: Aeronautics Institute of Technology (ITA), 2008.

[42] Welch, I. S.; Stroud, R. J. Re-engineering security as a crosscutting concern. The Computer Journal, v. 46, n. 5, p. 578-589, 2003.

[43] Camargo, V. V. Frameworks transversais: definições, classificações, arquitetura e utilização em um processo de desenvolvimento de software. 2006. PhD's Thesis in Computing Science – University of São Paulo, São Carlos, 2006.

[44] Lampson, B. W. A note on the confinement problem. Communications of ACM. v. 16, n. 10, p. 613–615, October, 1973.

[45] LU, Peng; YIN, Zhao-lin. Analysis and extension of authentication and authorization of Acegi security framework on spring [J]. Computer Engineering and Design, v. 6, p. 022, 2007.

Eduardo Martins Guerra received a PhD in Computer and Electronic Engineering from Aeronautics Institute of Technology - ITA in 2010. He has been working for National Institute for Space Research since 2013 where he is an Associate Researcher. His current research interests include Software Engineering, Framework Development and Application Security.

Jefferson O. Silva received a Master's degree from Aeronautics Institute of Technology - ITA in 2013. He has been working for Pontifícia Universidade Católica de São Paulo - PUC-SP since 2011. He is currently a doctoral student in Instituto de Matemática e Estatística in Universidade de São Paulo - IME-USP. His current research interests include Social Computing, Software Engineering, and Computer Security.

Clovis Torres Fernandes received a PhD in Computer Science from Pontifícia Universidade Católica of Rio de Janeiro – PUC/Rio in 1992. He has been working for Instituto Tecnológico de Aeronáutica – ITA since 1980 where he is an Associate Professor and Director of LAI – Learning and Interaction Laboratory in the Computer Science Department. His current research interests include Software Engineering, Computers and Education and Computer Security.