Federated Access Management

for Collaborative Environments

by

Carlos Ernesto Rubio Medrano

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved November 2016 by the
Graduate Supervisory Committee:

Gail-Joon Ahn, Chair
Adam Doupé
Ziming Zhao
Raghu T. Santanam
Dijiang Huang

ARIZONA STATE UNIVERSITY

December 2016

ABSTRACT

Access control has been historically recognized as an effective technique for ensuring that computer systems preserve important security properties. Recently, attribute-based access control (ABAC) has emerged as a new paradigm to provide access mediation by leveraging the concept of attributes: observable properties that become relevant under a certain security context and are exhibited by the entities normally involved in the mediation process, namely, end-users and protected resources. Also recently, independently-run organizations from the private and public sectors have recognized the benefits of engaging in multi-disciplinary research collaborations that involve sharing sensitive proprietary resources such as scientific data, networking capabilities and computation time and have recognized ABAC as the paradigm that suits their needs for restricting the way such resources are to be shared with each other. In such a setting, a robust yet flexible access mediation scheme is crucial to guarantee participants are granted access to such resources in a safe and secure manner. However, no consensus exists either in the literature with respect to a formal model that clearly defines the way the components depicted in ABAC should interact with each other, so that the rigorous study of security properties to be effectively pursued. This dissertation proposes an approach tailored to provide a well-defined and formal definition of ABAC, including a description on how attributes exhibited by different independent organizations are to be leveraged for mediating access to shared resources, by allowing for collaborating parties to engage in federations for the specification, discovery, evaluation and communication of attributes, policies, and access mediation decisions. In addition, a software assurance framework is introduced to support the correct construction of enforcement mechanisms implementing our approach by leveraging validation and verification techniques based on software assertions, namely, design by contract (DBC) and behavioral interface spec-

ification languages (BISL). Finally, this dissertation also proposes a distributed trust framework that allows for exchanging recommendations on the perceived reputations of members of our proposed federations, in such a way that the level of trust of previously-unknown participants can be properly assessed for the purposes of access mediation.

Para mi familia: la felicidad esta en el camino, no en el destino...

ACKNOWLEDGMENTS

I would like to start by recognizing the extraordinary support and guidance I have received during these years from my advisor, Gail-Joon Ahn. Indeed, my doctoral studies began in a difficult time for me, both at the professional and the personal level, and thanks to his kind support, patience and advice, I was finally able to enjoy the extraordinary experience of being involved in world-class graduate studies.

I would also like to thank my supervisors and friends Adam Doupé and Ziming Zhao, who provided a lot of high-level and technical advice on the different projects we had a chance to work on. In addition, I would also like to acknowledge my current and past labmates at the Laboratory of Secure Engineering for Future Computing (SEFCOM) and the Center for Cybersecurity and Digital Forensics (CDF) at Arizona State University: from the many things I learned throughout these years, the lessons about friendship and kindness I got from all of you are the most important ones I will never forget.

Last but not least, I would like to thank the extraordinary support I received from my family and friends in Chihuahua and El Paso, who shared with me the suffering and joy of this long trip that lasted for several years. Indeed, I have no words to express the gratitude and admiration I feel for all of you, as no matter how far away I was from home, your support and patience never let me feel alone. From now on, I promise to dedicate myself to be a much better love-giving person as all of you certainly are.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

In the last decades, modern societies have experienced a dramatic change due to the widespread introduction of computing infrastructures, which now influence almost every aspect of daily life. As computer systems become more sophisticated and efficient, they are becoming the core of many activities in both the public and private sectors. As an example, end-users now leverage mobile and web applications to perform tasks such as schooling, banking, shopping or traveling. In addition, websites and system administrators are currently gathering data about users, including usage patterns, either in a *direct* way, e.g., by asking a student to register to an online course, or in a *concealed* way: tracking cookies and IP addresses when shopping for later analysis. Such data, commonly known as a *digital footprint* (Girardin *et al.* (2008)), should also be leveraged on favorable and convenient terms for the end-users. For such a purpose, organizations may partner with each other to safely exchange information about end-users and running environments request for security purposes to mediate access to protected resources. As an example, in recent years, worldwide research-oriented institutions have engaged in active, high-performance and highly-collaborative projects that demand the shared consumption of resources, e.g, scientific data and distributed computation time. For instance, large amounts of data are transferred daily between independent high-performance computing facilities by using dedicated networking channels, allowing scientists to fully collaborate on the processing of data that may potentially lead to groundbreaking new discoveries.

As of today, security measurements in such collaborative projects are configured and maintained in a manual basis. For instance, authentication and authorization

are implemented by manually creating, configuring, and communicating credentials, e.g., by sending them over standard email attachments, and by deploying heterogeneous enforcement modules to restrict the nature, amount, and the physical location of resources to be shared in the context of a collaborative project. Such a task has been identified as tedious and error-prone as the aforementioned institutions currently depict complex in-house systems that handle a considerable number of end-users and must accommodate an equally increasing number of new collaborative projects on a daily basis, which greatly affects their reliability, scalability, and deployment time. As an example, the *energy sciences network* (ESnet) (US Department of Energy (2015)), currently supported by the United States Department of Energy (DoE), provides high-performance networking capabilities for approximately 40,000 users located in 128 DoE-sponsored research centers. ESnet also supports collaborative projects with international scientific-oriented organizations such as GÉANT (Europe's National Research and Education Networks (NRENs) (2015)) and NORDUnet (Nordic Council of Ministers (2015)) in Europe, as well as other organizations in Asia and the Americas, thus making the complete replacement of existing security infrastructure not viable.

With this in mind, providing a robust, flexible, scalable, and easy-to-understand approach that leverages independently-run systems and data for the automated specification, administration, and runtime enforcement of authentication and authorization policies is extremely desired.

In the context of collaborative projects, *attribute-based access control* (ABAC) (Hu *et al.* (2014)) as been recognized as a convenient way to mediate access to shared resources. Using ABAC, rich yet flexible policies can be created by leveraging *attributes*: well-defined security-relevant properties that are exhibited by the entities involved in the process of authorization and authentication, namely, end-users, re-

sources, and environmental infrastructures such as operating and file systems, credential management centers, network deployments, etc. As an example, policies restricting access to inter-organizational network connections may be defined using attributes obtained directly from the network infrastructure itself, (e.g., bandwidth capabilities), the data to be transferred (e.g., origin and size), and from the end-users attempting the connection, such as locally-defined credentials. However, no standard model describing ABAC exists in literature that formally defines its main components, the interactions between them, and the way attributes originated from independent *heterogeneous* organizations are to be leveraged for specifying and enforcing policies tailored for collaborative projects. In addition, there is a need for dedicated frameworks that can be used to verify that an ABAC solution is implemented correctly at the source-code level by participant organizations, thus ensuring the security guarantees as depicted by a theoretical model are indeed preserved in real-life implementations.

1.1 Thesis Statement

We now introduce the main hypothesis that is developed throughout this dissertation:

Attributes can be leveraged for providing a federated access management solution to support collaborations between independently-running organizations, which use the same or different access control models, enforcement mechanisms, as well as assurance and conformance frameworks.

This dissertation provides the following contributions towards supporting such an hypothesis:

1. Lately, approaches based on the concept of *federations* such as Shibboleth (Morgan *et al.* (2004)) and OpenID (Recordon and Reed (2006)) have been shown to be extremely useful for independently-run organizations to collaborate with each other for the purposes of authentication, at the same time they maintain their *independence* with respect to the administration of local security infrastructures, e.g., credential-based systems. Following this intuition, this dissertation starts by describing the challenges involved in providing a well-defined theoretical model for ABAC in the context of collaborative access management. Later, the duties and responsibilities involved in a solution that allows for participant organizations to engage in federations for the specification, discovery, and communication of attributes, policies, and access mediation decisions are described. This way, rich yet flexible policies can be specified and enforced, allowing for attributes defined in an intra-organizational (*local*) scope to be used in wider inter-organizational (*federated*) contexts, thus allowing for participant organizations to engage in safe collaborations while keeping full control of the decision making process, e.g., *what* to share, *when* to share, and *who* to share to.

2. Next, the challenges as well as a solution for a reference implementation of the theoretical model described above is presented, in such a way that it can potentially serve as a *guidance* for participant organizations to independently implement our proposed federations in a customized way, e.g., by reusing existing security infrastructures, without having to provide a complete solution from scratch.

3. In addition, in order to support the detection and removal of security vulnerabilities in the implementation of ABAC approaches for collaborations, this disser-

tation introduces an assurance and conformance framework at the source-code level based on the concept of *software assertions* (Rosenblum (1995)), and the use of existing techniques based upon them, such as *design by contract* (Hoare (1969)) and *behavioral interface specification languages* (BISL) (Burdy *et al.* (2003)). We will provide a description of the research challenges involved in fulfilling such a goal, as well as the description of a systematic solution that involves the participation of different *actors* in the software development process, namely architects, designers, coders, and testers.

4. Finally, as our proposed solution is expected to accommodate for an increasing number of both participant organizations as well as collaborative projects, this dissertation also presents the design and implementation of a *trust management framework* (Ruohomaa and Kutvonen (2005)), in such a way that participants can exchange *recommendations* on the perceived *trust* of other participants they have interacted with in this past, thus removing the need for all participants to know each other before engaging in new collaborative projects that involve resource sharing.

1.2 Dissertation Outline

This dissertation is organized as follows: it starts by providing some background in Chapter 2. Next, the challenges involved in providing a solution for our proposed federated access management (FAM) are presented in Chapter 3. Later, a theoretical model solving such challenges is discussed in Chapter 4. Next, a description is introduced on a *proof-of-concept* enforcement mechanism for FAM in Chapter 5, followed by an approach tailored for providing assurance guarantees when constructing such mechanism in Chapter 6. Later, this dissertation proposes an approach for managing trust between different collaborating organizations in Chapter 7. Related work

is reviewed in Chapter 8 and future work, as well as some discussion on the topics addressed in this dissertation is discussed in Chapter 9. Finally, some concluding remarks are delivered in Chapter 10.

Chapter 2

BACKGROUND

2.1 Collaborative Environments

Recently, emerging technologies such as *cloud computing* (Armbrust *et al.* (2010)) have opened new possibilities for resource sharing in the context of collaborative projects within different organizations. As an example, partner organizations can now outsource data repositories and computation time to clouds and allow for other cloud tenants to access those resources based on a previously-defined agreement. In addition, collaborative research efforts may also benefit from similar settings by allowing data to be transferred and processed in operational centers or clouds that happen to be located remotely. Such a paradigm may not only include scheduling for effective resource sharing within a given cloud, but may also require data being efficiently transferred within different independently-managed networks. Recently, other emerging technologies such as *software-defined networks* (SDN) (Monsanto *et al.* (2013)) have addressed this problem in the context of locally-managed intra-organizational resources, e.g., a single network domain under control of a single organization. However, to fully obtain the benefits of inter-organizational projects, an automated solution that considers both distributed and independently-run networks and resources is needed. With this in mind, several organizations providing high-performance networking capabilities have recently engaged into a collaborative effort for providing advanced resource sharing, e.g., path and bandwidth scheduling over distributed and independently-operated networks, in such a way that collaborations involving big data and high-performance computational requirements can be better supported.

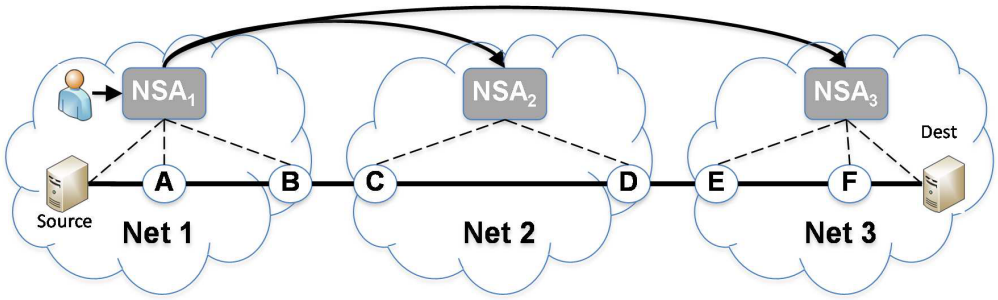

Figure 2.1: A Collaborative Setting Between Independent Organizations.

As an example, the Open Grid Forum (OGF) (Forum (2015)) introduced a multi-organizational effort called the *network services interface* (NSI) (Open Grid Forum (2015)) that is composed of a set of well-defined protocols that allow participant organizations to collaborate on research endeavors by implementing inter-organizational *services* in an automated way. The protocol devised for a given NSI service is implemented by so-called *network service agents* (NSA) which are expected to support all service-related tasks within the context of a given administrative domain. Fig. 2.1 shows an example depicting a data transfer between two hosts that are located within the administrative boundaries of two different organizations and whose networking path involves the participation of a third network serving as a bridge. In this example, each participating network implements the protocol devised for the NSI *connection* service by means of a dedicated NSA: a connection request R is first serviced by the *local* NSA where R originates (NSA_1 in Fig. 2.1). On each network, the local NSA is in charge of reserving local ports and bandwidth to create a connection within its network boundaries. NSA_1 is also in charge of contacting the other NSAs involved in serving R (NSA_2 and NSA_3) so that they can make reservations within their inner networks. In addition, all involved NSAs must handle network connections between independent networks by physically interconnecting any relevant *service termination points* (STPs), which are abstract (high-level) representations of actual network ports

and are labeled from A to F in Fig. 2.1. Once the connection path between the source and destination hosts is completed, the requested data transfer takes place.

2.2 Access Control Models

Access control, also known as *authorization*, has been historically recognized as an effective paradigm for providing security guarantees in computer-based systems. For instance, industry and academic researchers, as well as security and military agencies, have recognized since the early 1970's the need to provide well-defined rules for mediating access to sensitive resources, e.g., confidential data and information (Ferrari (2010)). In addition, identifying and verifying that software systems correctly follow such mediation rules was also identified as an emerging trend. An access control model is intended to unambiguously describe the interactions between the different *entities* that participate in an access mediation process. Such entities may include *actors*, e.g., end-users (human beings) and subjects (computer programs acting on behalf of end-users), a set of *targets*, e.g., protected resources, and a set of access rights, also commonly referred as *permissions*. As an example, an authorization model is expected to describe under which circumstances a given permission is to be granted or denied to a given end-user upon an access request. Such interactions are commonly described in the literature using formal mathematical approaches, e.g., set theory. Using an authorization model, *policies* can be specified and later enforced within a computer system. In addition, models may serve as a guide for implementing such enforcement *mechanisms* in a software system, and may also serve as a means for the psychological acceptance and understanding of the process of access mediation for both security officers and end-users. Over the years, several authorization models were introduced to better capture the specific needs of particular organizations and computer settings. Notable examples include *discretionary access control* (DAC)

(Ahn (2009)), *mandatory access control* (MAC) (Upadhyaya (2011)), and *role-based access control* (RBAC) (Sandhu *et al.* (1996)).

2.3 Attribute-based Access Control

Recently, *attribute-based access control* (ABAC) (Hu *et al.* (2014)) has emerged as a novel and interesting paradigm for mediating access to sensitive resources within software systems. In ABAC, a given access control request, e.g., reading a confidential data file, is granted upon the satisfaction of constraints involving security-relevant properties, also known as *attributes*, that are exhibited by the access control *entities* involved in the request. Commonly, such entities include the aforementioned *actors* and *targets*, as well as any applicable *context*, i.e., the running environment where a given software system is executed, such as an operative system or a cloud setting (Rubio-Medrano *et al.* (2013b)). Constraints may be defined by using standard comparison operators (e.g., $<$, $>$, $=$, etc.) as well as domain-dependent boolean functions over either predefined values or attributes. Constraints may also be related to other constraints using boolean operators (e.g. and, or, not). *Attribute provisioning* is the process of generating a given attribute and assigning it to a given entity, such that the attribute can be effectively leveraged for making access decisions. Such a process includes the discovery, creation, assignment, and communication of a given attribute. The entity, e.g., a domain controller, that generates a given attribute is called the attribute *source*. The entity assigning the attribute is known as the *provider*. Because many different constraints can be potentially created based on an equally large variety of attributes, ABAC has been found to be convenient for the specification of rich yet flexible authorization policies. In addition, it is suggested in literature that ABAC may serve as a theoretical *foundation* for representing other well-known access control models that have been largely discussed in literature and also widely

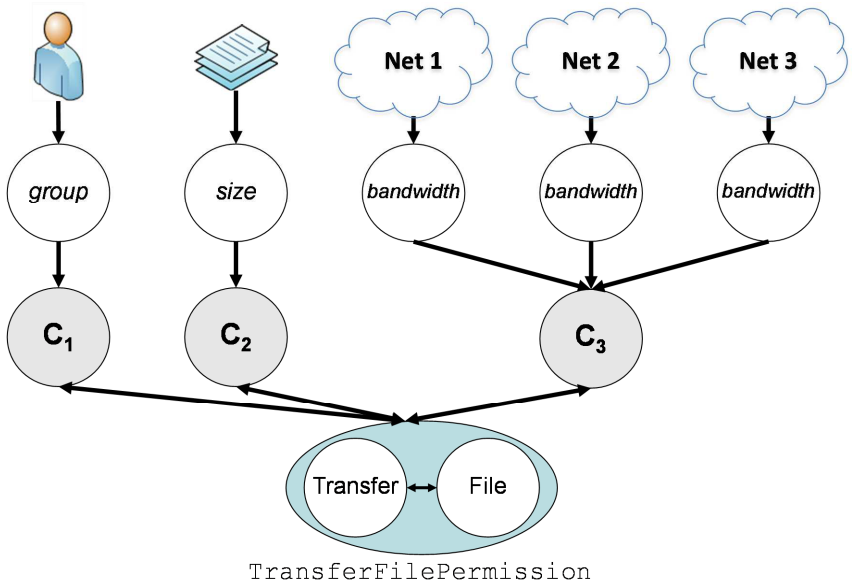

Figure 2.2: A Combined Use Case Depicting ABAC in a Collaboration Setting.

implemented in practice such as RBAC (Jin *et al.* (2012)).

Fig. 2.2 provides a graphical depiction of a sample ABAC policy that is to be enforced when *actors* request a data file (*resource*) to be transferred from a server located in their local network to a server located in another network (*environment*), depicting a use case like the one presented in Fig. 2.1. Such an access right is represented by a permission named `TransferFilePermission`, which is only granted if the following constraints are met: first, the constraint C_1 requires the *actor* issuing the request to be a member of a certain collaborative group, namely group G. Second, the size of the data file to be transferred must be less or equal to 10 TB (constraint C_2). Finally, constraint C_3 requires the available transfer bandwidth within each of the participating networks should be greater or equal to 1 Gbit/s.

2.4 Assertion-based Verification and Validation

The process of verification and validation has been a critical step to ensure software systems observe their expected runtime *behavior*, e.g., they do what they are supposed to when executed. Among the many existing techniques for such a purpose (D'silva *et al.* (2008)), *software assertions* (Rosenblum (1995)) are defined as formal constraints intended to precisely describe such runtime behavior and are commonly written as annotations at the source code level. Using assertions, developers can specify what conditions are expected to be valid before and after a certain portion of code is executed, e.g., the range of values that the parameter of a given function is allowed to take. *Design by contract* (DBC) (Hoare (1969)) is a software development methodology based on assertions and the assumption that the *creators* as well as the prospective *consumer* developers of a given software module establish a *contract* that must be followed for the module to be used correctly. Commonly, such a contract is defined in terms of assertions in the form of *pre* and *post* conditions, among other related constructs. Before using a DBC-based software module M, consumers must make sure that M's preconditions hold. In a similar fashion, creators must guarantee that M's postconditions hold once it has finished execution, assuming its corresponding preconditions were satisfied beforehand.

The *Java modeling language* (JML) (Burdy *et al.* (2003)), is a *behavioral interface specification language* (BISL) for Java, with a rich support for DBC contracts. Using JML, the behavior of Java modules can be specified using pre- and postconditions, as well as class *invariants*, which are commonly expressed in the form of assertions, and are added to Java source code as comments such as `//@` or `/*@...@*/`. For illustrative purposes, consider the DBC/JML contract shown in Fig. 2.3, which describes the expected runtime *behavior* of a Java method called `transfer`, which in turn implements

the collaborative case described in Fig. 2.1. Informally, the contract can be described as follows: a successful data transfer, represented by a return value `true`, will take place only if the both the parameters representing the destination IP address and the destination port are well-formatted and the file to be transferred exists in the local file system. If any of these conditions is not met, the transfer will not take place and `false` will be returned as a result. For simplicity, other potential issues present in real-life deployments, e.g., networking problems are elided.

The contract makes use of the *model* fields `sourceIP` and `sourcePort` (lines 3–4). In JML, it is possible to define model fields, methods and classes (Cheon *et al.* (2005)), which differ from their regular (*concrete*) counterparts in the sense they are used for specification purposes only, in an effort to better describe a given JML contract in a higher level of abstraction, without worrying about how it is implemented at the source code level. As an example, the model field `sourceIP`, shown in line 3, is used to provide an abstract representation of the IP address held by the physical machine where a Java class implementing the `FileTransfer` interface resides. Following JML rules, such a class will be required to provide a suitable implementation for the `sourceIP` model field. In addition, the contract can be described by using two specification *cases*. In JML, specification cases that are expected to terminate *normally*, that is, without *diverging* nor throwing exceptions at runtime, are defined by means of the `normal_behavior`. Conversely, cases that allow a method to throw a runtime exception are denoted by means of the `exceptional_behavior` keyword. The first case, shown in lines 10–17, contains preconditions (defined by means of the `requires` keyword) requiring the method parameter `file` (of type `java.io.File`) not to be `null`. In addition, the data file represented by such a parameter is also required to exist in the local file system, as depicted by the invocation of the `exists()` method. Finally, both the `destIP` and `destPort` must also be within some predefined

```
1  public interface FileTransfer{
2
3   //@ public model String sourceIP;
4   //@ public model int sourcePort;
5
6   //@ public invariant sourceIP != null &&
7   //@                  sourceIP.length() > 0;
8   //@ public invariant sourcePort > 0;
9
10  /*@ public normal_behavior
11    @  requires file != null          &&
12    @           file.exists()         &&
13    @           destIP != null        &&
14    @           destIP.length() > 0   &&
15    @           destPort > 0          &&
16    @           destPort <= 8080;
17    @  ensures \result == true;
18    @  also
19    @  public normal_behavior
20    @   requires file == null         ||
21    @            !file.exists()       ||
22    @            destIP == null       ||
23    @            destIP.length() <= 0 ||
24    @            destPort <= 0        ||
25    @            destPort > 8080;
26    @   ensures \result == false;
27    @*/
28  public /*@ pure @*/ boolean transfer(File file,
29                                       String destIP,
30                                       int destPort);
```

Figure 2.3: An Excerpt of a JML-annotated Java Interface.

parameters (lines 13–16). Postconditions for the first case, which are defined by the `ensures` keyword, guarantee that the result of the method, assuming the aforementioned preconditions have been met, will be the value `true`. In a similar fashion, the second specification case, shown in lines 19–26, ensures the method will return a value of `false` if any of the preconditions listed above is not met before the method gets executed. Because the `transfer` method is not allowed to modify the set of memory locations, e.g. instance variables or model fields, that are accessible to the `FileTransfer` interface, the method is regarded as `pure` in line 28. Finally, lines 6–8 depict assertions representing *invariants*: the model field `sourceIP` is never allowed to be `null` or to have a length less than zero. In addition, the model field `sourcePort` must be always greater than zero.

Over the years, a suite of verification and validation tools supporting DBC/JML contracts has been developed (Burdy *et al.* (2005)). As an example, `JET` (Cheon (2007)) is a dedicated tool tailored for providing automated *unit* testing of JML-specified Java modules. Using `JET`, testers can verify the correctness of a Java module by checking the implementation of each of its methods (either public, protected or private) against their corresponding JML specifications. The contract of a given method M is used as a test *oracle*, by first translating it into *runtime assertion checking* (RAC) code. Then, proper values (either primitive or reference ones) are *randomly* created for each of M's formal parameters, and compared to check compliance status against the RAC code created for M's precondition. If such a precondition is satisfied, a *valid* test case is said to be created. Otherwise, the test case is said to be *useless*, so it is discarded. If the test case is found to be *valid*, M's body is executed. If any exception not devised for M is thrown, the test case is regarded as *failed*. Otherwise, the RAC code for M's postcondition is executed. If such a postcondition is satisfied, the test is regarded as a *success*, otherwise, it is regarded as *failed*. Many different test cases

can be created for M after different combinations of values for M's parameters are created by `JET`. Each of these test cases is evaluated in a different execution *thread*. Ideally, if M meets its corresponding JML specifications, all *valid* test cases should ultimately be evaluated to a *success*.

In another example, `ESCJava/2` (Burdy *et al.* (2003)) is a tool supporting JML specifications that is based on a theorem prover and internally builds *verification conditions* (VCs) from both the analyzed source code and its corresponding JML specifications, which the theorem prover then attempts to prove, thus allowing for the automated analysis of whole code modules without actually running the applications, e.g., no test cases need to be generated. In particular, `ESCJava/2` leverages *modular reasoning* (Flanagan *et al.* (2002)), which is regarded as an effective technique when used in combination with static checking since code sections can be analyzed and their JML-based specifications can be proved by inspecting the specification contracts of the methods they call within their method *bodies*.

2.5 Trust Management Systems

In the scenario depicted in Chapter 2.1, collaborative tasks are commonly carried on by peers that implement resource sharing, e.g., data and network infrastructure, on the basis of previously-arranged collaboration agreements (either formal or not) that clearly specify the purposes of the collaboration, the amount of resources to be shared, as well as the individuals who will be granted access to such resources. Such a solution, while convenient in the context of a small number of both collaborative projects and participant organizations, may not fully escalate to incorporate an increasing number of participants as well as a dynamically-changing number of resources to be shared over constant and variable periods of time. As an example, research collaborations following our running example depicted in Fig. 2.1 are ex-

pected to consume dynamically allocated amounts of network bandwidth as well as a varying number of data files belonging to an equally varying number of end-users. As collaborations grow in both size and nature, new organizations may be willing to engage in such partnerships and may be also willing to consume or provide new shared resources with other participants, thus complicating the establishment of dedicated agreements for resource sharing between all participants and also resulting in a possible administrative burden that may require multiple system-level configurations in order to all of these new collaborations to take place, e.g., issuing local credentials, updating access mediation policies, configuring firewall systems, etc.

In the past, numerous approaches have been developed in the context of *trust management systems* (Blaze *et al.* (1996)) to provide support for operations, e.g., collaborative resource sharing, between different organizations and individuals who may have not previously known each other and may have not arranged in a well-defined interaction agreement beforehand. In such a context, *trust* can be defined as *the extent in which one party is willing to participate in a given action with a given partner, considering the risks and incentives involved* (Ruohomaa and Kutvonen (2005)). Moreover, in the context of collaborative resource sharing, the peer providing a shared resource can be known as the *trustor*, whereas the peer getting access to it can be known as the *trustee*. In addition, *risk* may involve exposing valuable assets, e.g., the aforementioned data and network facilities, to possible acts of *misbehavior* such as attacks or abuse in the consumption of resources, e.g., monopolizing bandwidth capabilities in detriment of other potential users or peers. Therefore, before engaging into a possibly-risky operation with an *untrusted* peer P, resource owners may want to assess a level of trustiness on P so they can decide whether or not to proceed.

In other to alleviate this problem, literature has presented approaches based on taking into account the perception about a certain participant peer as created by a

history of past interactions with other peers (Hendrikx *et al.* (2015)). Such a concept, commonly known as *reputation* (Ruohomaa and Kutvonen (2005)), may take into account previously-perceived behavior as observed by a certain number of peers with respect to the actions, e.g., resource consumption, as conducted by a given peer. When information about the reputation of a previously-unknown peer P is required, participant peers may exchange a *recommendation* message clearly stating the reputation of P in a way an interested requester can process it, e.g., by using a common numerical scale. As an example, in Fig. 2.1, Net 3 may be willing to share a recommendation on the perceived reputation of Net 2 with Net 1, assuming Net 2 is unknown to Net 1, but still required for the whole data file transfer process as it serves as a *bridge* for connecting the networks of Net 1 and Net 3. Also, such an example assumes Net 1 and Net 3 have a previously trust relationship themselves, in such a way that Net 1 is willing to request and accept recommendations from Net 3. Following such an example, participants should be allowed to define their own reputation scores based on a commonly-understood framework, e.g., by using a numerical scale that can also serve for the purposes of combining (aggregating) the reputation scores provided by a set of independent peers. Such a aggregation process may then be based on mathematically-based techniques such as summation, average, weighting and normalization (Hendrikx *et al.* (2015)). Finally, reputation scores should be updated over time, in an effort to better capture the observed behavior of peers whose reputation is being assessed. For such a purpose, a recommender peer may implement its own in-house history tracking scheme which may take into account the history of previous interactions or may also rely on a well-defined reputation evolution framework that is established in the context of a given collaborative project. A possible scenario for such a reputation evolution framework may include awarding positive scores for transactions that were conducted within expected bounds and awarding

negative scores for transactions that ultimately resulted in an abuse in the utilization of a shared resource, e.g., consuming more bandwidth than expected, or resulted in unwanted activities that may have compromised the overall security of the shared resources or its supporting operating framework, e.g., a hacking incident.

Chapter 3

PROBLEM STATEMENT

This chapter is intended to identify the research challenges involved in developing an approach for federated access management that combines ABAC and the concept of inter-organizational collaborative efforts. It starts by describing the challenges identified for ABAC itself, and then moves on to address the challenges involved in mediating access for resources shared between independently-run organizations. Later, this chapter discusses the challenges in implementing an assurance and conformance framework that includes support for the assertion-based construction of software modules implementing the approach presented in this dissertation as well as a framework for the purposes of trust management between participant organizations

.

3.1 A Theoretical Model Leveraging ABAC

As introduced in Chapter 1, an approach for FAM should strive to provide a formal description on the way different organizations are to exchange security-related information, a.k.a., *attributes*, for the purposes of access management. Therefore, it becomes necessary to provide a well-defined description, a.k.a., a *model*, of ABAC in such a way that security guarantees with respect to FAM can be made. With this in mind, the set of features that are desirable in such a model can be articulated as follows:

Attribute Definition. A proposed model should strive to provide an unambiguous definition of attributes, the main component underlying ABAC. That includes a

proper way to specify the format of attributes, e.g., a naming convention, data types and value sets.

Constraints Definition. A model for ABAC should also provide a well-defined description of constraints based on attributes, which are also core to the model itself, as described in Chapter 2. As an example, a formal description of constraints should define under which circumstances they are to be evaluated and their expected result, if any. Referring to Fig. 2.2 and the policy described in Chapter 2.3, a reference model for ABAC should support the C_1, C_2, and C_3 defined upon attributes obtained from different organizations.

Model Formalization. In addition, a formal description of a model for FAM must be provided. As an example, the relationships between its different composing elements, such as end-users, attributes and access rights (permissions), should be described in a well-defined, mathematically-based way, e.g., using set theory, semantic descriptions, etc. This way, further source-code level implementations can be unambiguously developed by precisely defining how components interact with each other. In addition, further validation and verification techniques can be potentially developed based on such a formal description, as it will be detailed in Chapter 6.

Security State. A model for FAM should also include a well-defined description of how attributes may influence the *security state* of a given software system, e.g., by satisfying attribute-based constraints related to access rights, so proper reasoning on security properties can be done as a result. As an example, the access decision resulting of evaluating the policy described in Chapter 2.3 could be potentially modeled as an access token to be issued to the requesting end-user for a certain period of time, in a similar fashion to the approach implemented by well-known methodologies such

as Kerberos (Neuman and Ts'o (1994)) or OAuth (Jones and Hardt (2012)).

Methodology Independence. A model supporting ABAC should be *independent* of any technology or methodology, e.g., it should be *agnostic* to any underlying framework in such a way that it can be implemented without requiring the mandatory *a priori* installation of supporting technologies, thus allowing for independent organizations to customize their implementations to better fit their own needs without having to accommodate additional requirements.

3.2 Collaborative Environments

As shown in the illustrative case depicted in Fig. 2.1, collaborations may involve the sharing of proprietary resources between independent organizations that are remotely located from each other, e.g., scientists in organization Net 1 may want to leverage the computational facilities offered by organization Net 3 by transferring a considerable amount of data to the well-equipped processing servers located within the boundaries of Net 3. As of today, such a setting would include leveraging the existing high-performance capabilities of each of the three networks to manually schedule for the data to be transferred, e.g., manually configuring supporting networking equipment such as routers and switches. In addition, security settings must be manually configured as well: creation and communication of new credentials, update of existing policies, configuration of firewalls and networks, etc. In such a context, incorrect configurations can also potentially open the door for non-trivial security vulnerabilities, e.g., granting more privileges than required to perform a certain task. With this in mind, a solution for providing access management guarantees for collaborative settings should accommodate for the following:

Intra and Inter-organizational Policies. Participants should be allowed to define both intra-organizational and inter-organizational policies governing the way a given resource is to be shared in the context of a collaboration. As an example, each organization depicted in Fig. 2.1 may want to enforce its own set of access policies for resources being shared within a local context. In addition, organizations may favor the creation of inter-organizational policies to mediate access when a request is being issued from an external collaborative organization. Following our running example, the policy depicted in Fig. 2.2 could be implemented as a part of an agreement between the involved organizations to better mediate the way data connections are established.

Reusing Existing Infrastructure. Participants should be allowed to leverage their own *in-house* authorization systems, which may in turn handle their own set of local credentials and possibly their own set of locally-relevant attributes. This may potentially result in problems such as *attribute incompatibility* (Paci *et al.* (2009)) or different attributes being assigned to the same entity by different domains, e.g., users getting credentials issued in the context of different collaborative projects, possibly resulting in a large set of credentials to be handled. In such a context, organizations may not favor a complete replacement of their current authentication and access control modules, as that may involve an unfeasible organizational effort and monetary cost.

Attribute Derivation. In practice, every access control entity, e.g., an end-user or a protected resource, that is involved in servicing a given access request, is expected to provide a set of attributes, e.g., user credentials or access certificates, which may have been assigned either by its local security domain or by an external one. When evaluat-

ing a given policy P, entities such as end-users must show the attributes required from them in the context of P. If such attributes are not shown, e.g., they are not available even when they may have been legitimately assigned beforehand, the evaluation of P may fail as a result, thus possibly causing legitimate access to be denied. Also in practice, attributes are commonly assumed to exist at policy evaluation time, either locally or remotely, e.g., stored in a dedicated centralized database, or may be in turn *derived* by processing other related attributes. However, existing infrastructures are not capable of seamlessly locating and transforming attributes in a distributed setting such as the one depicted by independently-run organizations. As an example, each of the organizations depicted in Fig. 2.2 is expected to provide a *bandwidth* attribute obtained from real-time network configuration settings. As each network may in turn manage its own dedicated network infrastructure systems, a proper setting should be in place to allow for such values to be extracted and transformed into a well-defined format that can be properly understood by other organizations for access mediation purposes.

Support for Authorization. Finally, existing approaches for inter-organizational security, e.g., OpenID (Recordon and Reed (2006)) and Shibboleth (Morgan *et al.* (2004)), are focused on authentication: support for authorization is limited and is mostly left for third parties to implement from scratch, e.g., attribute and policy definition, discovery, and evaluation. An approach for access management between independently-run organizations should leverage the experience obtained from such popular approaches to introduce well-defined authorization mechanisms that can be understood and enforced by participants in an efficient and secure way.

3.3 Enforcement Mechanisms

Based on the discussion introduced in Chapter 1, this section elaborates on the challenges devised for an enforcement mechanism that is intended to implement the approach depicted in Chapter 3.1 and Chapter 3.2, which is to be introduced in Chapter 4.

Policy Discovery and Evaluation. As mentioned in Chapter 3.2, collaborative organizations should be allowed to define both intra-organizational (*local*) as well as inter-organizational (*federated*) policies for mediating resource sharing. As an initial step, the set of policies (both local and federated) relevant to a given access request, should be properly located. In the context of independently-run organizations, policies could be potentially stored in repositories located either locally, e.g., a local database, or in a remote peer. Later, a policy that happens to be physically stored and evaluated within a given domain may end up being enforced in different domains, following a predefined collaboration agreement.

Incorporation with Domain-specific Functionality. In addition, there is a need to resolve inter-domain policies *on-the-fly*, e.g, incorporating policy evaluation with domain-specific functionality such as the data transfer connection depicted in our running example in Fig. 2.1. For such a purpose, the policy discovery, retrieval and evaluation process should be carried on as efficiently as possible, in such a way that the performance delay introduced by such a process can be potentially diminished. Moreover, a proper interface between the policy evaluation modules and any other relevant software components should be provided, e.g., implementing the authorization mechanism as a client service that can be invoked by external domain-specific components.

Attribute Provisioning. As described in Chapter 3.2, the policy evaluation process may rely on the runtime collection of attributes originated in different participant peers. For such a purpose, there is a need to standardize the creation and maintenance of automated attribute provisioning schemes, as different organizations may leverage attributes assigned to many different access entities. Moreover, each organization may in turn define their own internal processing for handling attributes. Therefore, a protocol for the proper discovery, retrieval, and communication of attributes is highly desired.

Reuse of Existing Infrastructure. As stated in Chapter 1, participant organizations may not favor the complete replacement of existing security-related infrastructure within their local domains, e.g., credential-based systems tailored for end-users. Therefore, an enforcement mechanism depicting our approach should encourage the reuse of existing infrastructure as much as possible.

Scalability and Platform Independence. Since an approach for access management in the context of collaborations is extended to serve a considerable number of participant organizations and collaborative projects, scalability and platform-independence must be also considered as a required feature. In addition, the process of joining the approach to be proposed in this dissertation should be carried on in an efficient and prompt way, e.g., without requiring complicated setups or introductory procedures that may complicate the adoption of the approach for interested organizations.

Decentralized Implementation. Due to the independently-run nature of the participants involved in collaborative settings, a decentralized architecture becomes desirable. This way, no central node, e.g., a shared policy repository, may be needed for

an access mediation approach to become operational, thus possibly alleviating any politic-centric trouble, e.g., participants not willing to fully delegate policy evaluation tasks to other participants. At the same time, potential *denial of service* (DoS) attacks (Needham (1993)), which typically flourish in centralized approaches, may be also significantly reduced.

3.4 Assurance and Conformance

As introduced in Chapter 1, this dissertation aims to provide a framework that can ensure the overall deployment of an inter-organizational solution for access management is done in a correct way, that is, it provides correct access mediation for shared resources at the same time it encourages participants to actively engage in different collaborative projects. As a first step, a correct implementation of an access management mechanism at the *wide* inter-organizational level should focus on the way each participating organization implements it independently at the source-code level. As a second step, as the number of participant organizations, as well as the number of collaborations is expected to increase over time, an initial assumption requiring all participants to know and trust each other beforehand may no longer hold. Therefore, participants should be able to assess a degree of trust on participants with they have had no interaction with in the past. With this in mind, this chapter discusses the challenges as well as a set of desired features involved in providing an assurance and conformance framework that combines the two steps just mentioned before.

3.4.1 Implementing Enforcement Mechanisms

Verifying and Validating Independent Implementations. As it has been anticipated in Chapter 3.3, the approach proposed in this dissertation relies on participant organizations implementing a set of protocols for mediating access to shared

resources. In such a context, the overall security of a given collaboration project may heavily rely on participants implementing our approach in a correct way at the source code level, as any deviation from the expected behavior may ultimately result in non-trivial security vulnerabilities, e.g., allowing unintended access to sensitive resources. With this in mind, there is a need for a framework that allows participants to construct, verify and validate their own independent implementations of our approach for access mediation, in such a way that the shortcomings just mentioned can be better avoided, thus encouraging the development of new collaborative projects that involve resource sharing as a result.

Supporting Extensibility and Customization. In addition, there is a need to allow for participant organization to further extend and customize our proposed approach to better meet their specific organizational and community-wide needs. As an example, participants within a certain federation may want to introduce their own attribute derivations as discussed before in this chapter. Therefore, support must be provided for the verification and validation of software modules implementing customized behavior, such as the attribute derivations just mentioned.

Supporting Integration of *In-House* Systems. As mentioned in Chapter 1 and Chapter 3.3, participant organizations are expected to leverage already-existing security-enforcement mechanisms, e.g., credential based systems, as their complete replacement may not be feasible due to organizational costs and deployment time. As such systems are expected to be integrated with our proposed approach, there is a need to ensure such integration is done in a correct way by providing support for the construction, verification and validation of source-code level implementations.

3.4.2 Managing Trust Between Participants

Trusting Unknown Participants. As mentioned before in Chapter 1, the number of collaborations between independently-run organizations is expected to grow over time due to the many benefits that engaging in such projects may offer: access to resources such as data, computation time, etc., as well as by the ultimate benefits intended for such endeavors, e.g., providing support for groundbreaking research discoveries. As the number of participant organizations grow, the establishment of trust relationships between all of them may get complicated. As an example, within a reduced number of collaborative partners, trust relationships may be developed between each other in such a way that each participant establishes resource sharing agreements with each potential partner. Such agreements may ultimately result in access mediation policies (either local or federated, as introduced in Chapter 3.2) that may be enforced at runtime. However, as the number of participant organizations grow, the establishment of such agreements may not be completely feasible, forcing participants to interact with previously-unknown peers with whom there is no history of previous interactions for the purposes of the enforcement of access mediation to shared resources.

Incorporating Security-based Functionality. Moreover, the perception of trust assigned to other participants should be incorporated into security-related functionality, in such a way that the level of trust assigned to a given interacting peer has a direct impact on security decisions, e.g., access mediation. As an example, leveraging the approach proposed in this dissertation, mutually-trusted partners may also agree on the way attributes are to be provisioned, as introduced in Chapter 3.1, thus allowing all parties to recognize and leverage each other's attribute in a safely way.

Fine-grained Trust Relationships. In addition, as collaborative projects may include different organizational groups, entities or even end-users, trust relationships may evolve differently over time, depending on the success or failure of each individual project. Therefore, there is a need for an approach that allows for participants to establish trust relationships at a fine-grained level, starting from the organizational one, all the way down to research groups and even individual end-users.

Providing a Mathematical Foundation. An approach tailored to provide trust management in the context of collaborative organizations may also require a mathematical-based foundation, in such a way that participating peers can model their perceptions on the trustiness of another peers based on a numerical scale, which can be later used to combine the scores as provided by different peers in a straightforward and easy-to-understand way, e.g., by performing a mathematical summation or an average as suggested by (Hendrikx *et al.* (2015)). This way, participants can calculate a perception of trust by collecting reputation scores from different sources. Such a process should be carried on periodically, possibly at runtime, before engaging into resource sharing operations with peers whose trust cannot be determined from direct *first-hand* experience. Conversely, peers should be allowed to update their reputation scores on other peers over time, as a response to changes in the history of shared transactions.

Leveraging Previous Community-wide Experience. In addition, peers should also be allowed to take into account the trust scores provided by other *known* peers with whom a highly-scored trust relationship has been developed in the past. This way, the scores provided by highly-trusted peers should be prioritized over the ones provided by other peers not in the same category, leveraging previous experiences

obtained from a community-wide context. For instance, following the example introduced in Chapter 2.5, Net 3 may be able to share its perceived trust score on Net 2, which can be later used by Net 1 to determine if Net 2 should be trusted for the file transfer process depicted in Fig. 2.1. Assuming Net 1 trusts Net 3 beforehand, the shared trust score will be taken into account when deciding if Net 2 should be trusted or not.

Providing a Decentralized Approach. Finally, a trust management framework supporting our approach should also follow the distributed architecture foreseen in Chapter 3.3. As an example, peers should be allowed to maintain their own local scores about their perceived reputation of other peers. In addition, peers should be allowed to share those with other peers (as suggested by the example introduced before) without the need of relying on a centralize node or set of nodes, thus staying true to the architectural design objectives introduced in previous chapters at the same time removing the chances of having nodes that might become *attractive* for hacking attacks, e.g., the aforementioned *denial of service* (DoS) ones (Needham (1993)).

Chapter 4

A THEORETICAL MODEL

In order to solve the challenges for inter-organizational access management described in Chapter 3.1, we propose that participant organizations engage in highly-collaborative and well-structured associations, also called as *federations*, for the provisioning (specification, generation, and communication) and the use of both attributes and policies to mediate access to shared resources. As depicted in a recent report by the *National Institute for Standards and Technology* (NIST) (Hu *et al.* (2014)), proper provisioning mechanisms may become a crucial component for the successful development of new technologies and infrastructures based on attributes. With this in mind, this chapter presents the core component of the overall *federated access management* that is discussed as a part of this dissertation. We start by first providing a conceptual definition of our proposed approach in Chapter 4.1. Later, we provide a formalization of our federated access management model in Chapter 4.2. Later, Chapter 9.1.1 provides some discussion on the way the approach presented in this chapter solves the challenges for inter-organizational access management described in Chapter 3.1.

4.1 Conceptual Model

As described in Chapter 1 and in Chapter 3, our approach is to be based on the utilization of attributes for access mediation purposes. With this in mind, we envision participant organizations leveraging attributes originated within their own local security domains in order to make access decisions. Such attributes are to be referred as

local ones in the rest of this dissertation, and are defined in Chapter 4.2.1. In addition, as many different *heterogeneous* attributes may exist between the security domains implemented by participants, there is also a need to agree on a set of attributes whose format, e.g., naming, data type and range of values, is well-defined and understood by everyone involved in a given federation. Such attributes, to be known as *federated*, should be in turn leveraged for defining and enforcing attribute-based *federated* policies for access mediation between participants. A definition for federated attributes is shown in Chapter 4.2.2. Before engaging in a transaction that involves sharing proprietary resources, all relevant federated policies should be located, parsed and evaluated before deciding if the transaction is to be authorized or not.

A graphical depiction of our approach is shown in Fig. 4.1: a locally-defined attribute a_1 belonging to a given user is transformed into a series of federation-recognized attributes (a_2, a_3, a_4) that are in turn provided by other organizations engaged in a federation and may be used for access control decisions. In a similar fashion, a local attribute a_3 is transformed into a globally-recognized attribute a_4 for access management purposes.

Fig. 4.2 presents a basic description of the methodology expected for our FAM approach: initially, attributes are to be assigned to entities, e.g., an end-user, in the context of the security domain established by each participant organization (0). Next, such an entity may initiate an access request to a given shared resource (1). Next, the set of attributes required for accessing such resource is communicated to the FAM framework (2), which then receives the attributes as provided by the requesting entity (3) and performs any necessary attribute provisioning as shown in Fig. 4.1 as well as the evaluation of all relevant policies (4). Finally, a final decision is communicated back and the requested resource may be accessed in case a positive decision was made (5)(6).

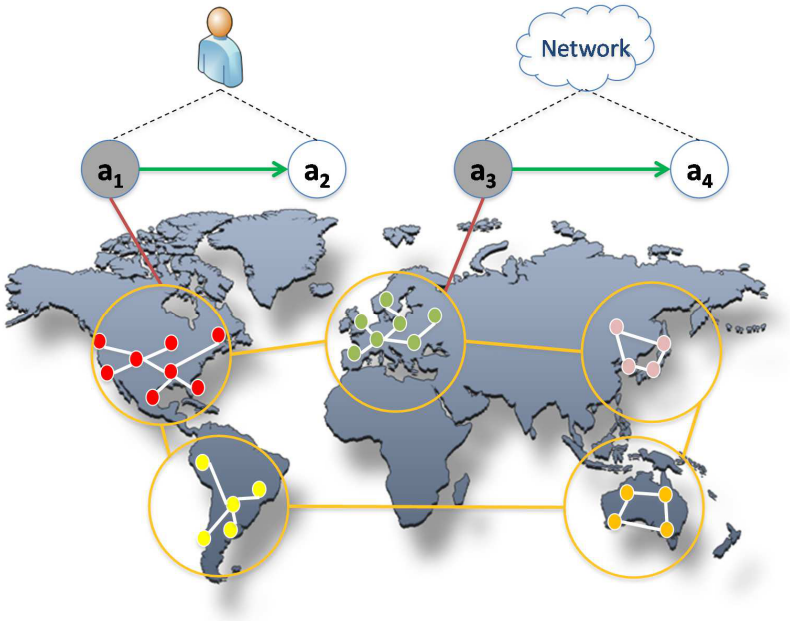

Figure 4.1: A Framework for Federated Access Management.

Our approach can be also described by leveraging the example shown in Fig. 4.3, which in turn depicts our running example shown in Fig. 2.1: the data file transfer operation is to be mediated by an inter-organizational *federated* policy as the one featured in Fig. 2.2. Such a policy is to be implemented by Net 2 (dashed line) and requires the *Ta* attribute, which serves as an access token related to the `TransferFilePermission` granting access to the aforementioned data transferring. The *Ta* attribute may be in turn obtained by transforming *local* attributes that are in turn provided by remote peers. As an example, Net 1 provides the local attributes *Ce* (local credential), *Ne* (network information), and *D* (file data size), which are then transformed into the federated attributes *G* (group membership), *S* (data size) and *Be* (network bandwidth) for policy evaluation purposes. Functionality is to be implemented by dedicated modules known as *access management agents* (AMA), which are later discussed in Chapter 5.

In order for our approach to be successfully implemented in practice, participant

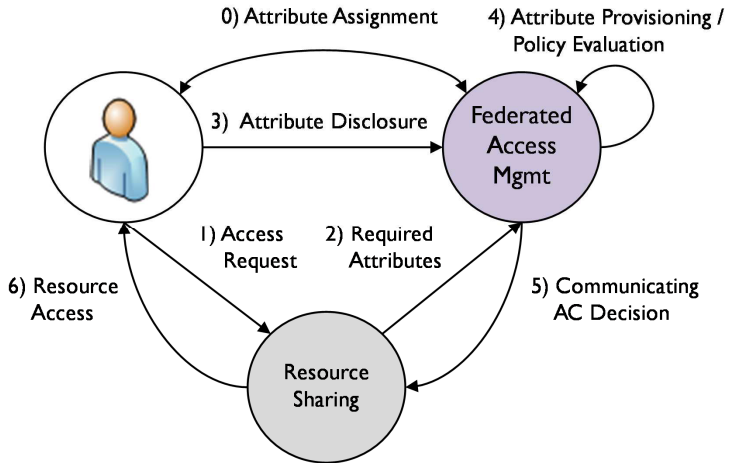

Figure 4.2: A Basic Methodology for Federated Access Management.

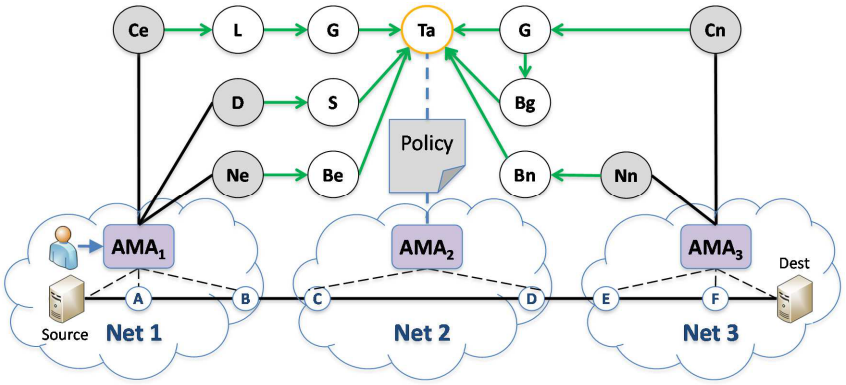

Figure 4.3: Our Proposed Solution Depicting our Running Example.

organizations engaging in our federations must fulfill the following requirements:

Resource Identification. As an initial step, participants should identify the set of organizational resources they are willing to share in the context of collaborations. As such resources may significantly vary in nature and quantity, e.g., data and computation time, a considerable effort needs to be invested in an organizational-wide analysis that may allow participants to rightfully determine what resources can be shared and under which conditions sharing can take place. Referring to our running

example in Fig. 2.1 and Fig. 4.3, participants must determine the network connection links and ports that will be use for inter-organizational data transfers, in such a way that the *quality of service* (QoS) (Sun *et al.* (2010)) devised for such connections can be properly guaranteed.

Attribute Identification. Participating organizations are to identify security-relevant properties within their local domains that may serve as *local* attributes for access mediation purposes. As an example, in Fig. 2.1, Net 1 should identify relevant metadata of the data to be transferred that contain the properties that are relevant under the policy depicted in Fig. 2.2, e.g., its size in bytes.

Attribute Mapping. Participants must map *local* attributes onto a set of well-known *federated* attributes to be used in the context of an inter-domain collaboration. Following our running example, the federated attribute labeled as S in Fig. 4.3 should provide an standard, widely-recognized definition of the size of a given chunk of data, e.g., a convention name, size unit, etc.

Attribute Discovery. Participants should allow organizational peers to discover the federated attributes they provide for policy specification purposes. Following our running example, Net 2 should be able to locate the attributes provided by Net 1 and Net 3 when constructing inter-domain policy for shared connections depicted in Fig. 4.3.

Policy and Attribute Administration. Organizations should implement a proper administrative model for creating, updating, and removing both local as well as federated attributes and federated policies that restrict access to protected resources within collaborative projects. Moreover, such an administrative model may also be

based on attributes as well. As an example, users depicting an attribute showing membership to a group of network administrators should be allowed to create and modify policies at the local and federated level.

Policy Retrieval. Upon receiving a resource access request, participants should retrieve the set containing local policies relevant to such request. Following our running example, Net 1 should retrieve any policies regarding data transfers originating in its local domain.

Attribute Provisioning. Participants should provision any local and federated attributes as specified in the policies relevant to a given access request. To enable this provisioning, participants are to make their federated attributes available for other peers to provision upon request. In addition, participants should provide an automated way to perform attribute mappings from local attributes into federated ones, and from federated to federated attributes as well, in such a way that end-users located within the same organization or in other peers can transform attributes when required. Following Fig. 4.3, Net 1 should provide a way to transform the credentials presented by an end-user into an attribute depicting membership to the collaborative group G that is required by the policy in Fig. 2.2.

Policy Dispatch. Participants should dispatch policy evaluation requests for federated policies that are relevant to a given access request. Conversely, participants should evaluate and provide results for any policy evaluation requests they receive as part of an evaluation process initiated by a federated peer. Back to our running example, participant networks should retrieve all attributes relative to a connection request that happen to be under the scope of their local security domain and should dispatch both attribute and policy evaluation requests to the other networks involved

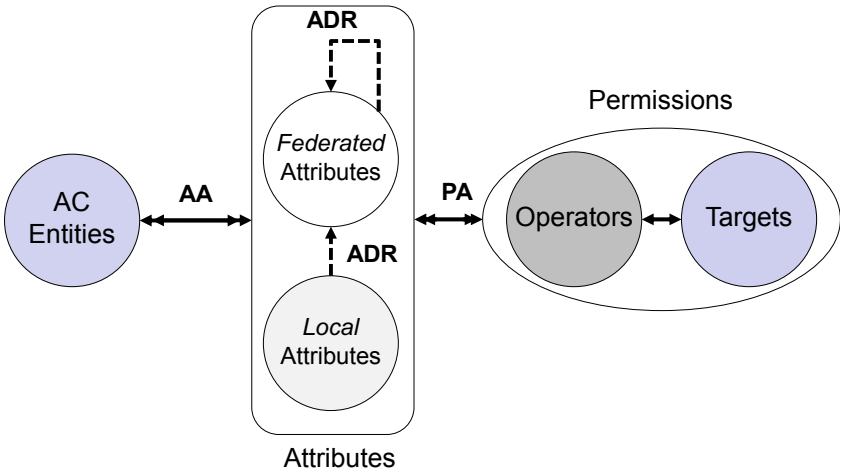

Figure 4.4: A Model for Federated Access Management.

in the construction of the network connection.

Results Aggregation. Finally, the evaluation decisions for the relevant policies, either local or federated, must be derived and combined to produce a final decision, which is to be communicated to the requesting entity, e.g., the end-user under the Net 1 domain in Fig. 2.1.

4.2 Model Formalization

Fig. 4.4 shows a visual representation of our proposed model: attributes are related to access control entities by means of the *attribute assignment* (AA) relation, allowing each entity to exhibit many different attributes and a single attribute to be potentially exhibited by more than one entity. As mentioned before, *Federated* attributes are publicly-known attributes that may be relevant in the context of a given collaboration project. *Local* attributes are related to federated attributes through *attribute derivation rules* (AD-Rules), which are shown as directed arrows in a dotted line in Fig. 4.4. The precise definition of such AD-Rules, e.g., how local attributes are ultimately related to federated ones, is defined by peers within the context of a

given collaboration. As we will discuss in Chapter 4.2.3, AD-Rules can be organized into a graph-like structure known as an *attribute derivation graph* (AD-Graph), which provides a representation of how attributes are related to permissions, which are in turn related to federated attributes by means of the *permission assignment* (PA) relation. Permissions are depicted as a combination of a protected source (target) and an operation that can be performed on it. A given attribute may be related to one or more permissions, and a given permission may be related to one or more attributes.

A formal description of our approach is shown in Fig. 4.5 and Fig. 4.6. The basic components are actors (ACT), targets (TAR), and context (CON), which together construct the set E of access control entities. Moreover, we also consider the sets of operations (OPER) and permissions (P). We also define the sets of names (N) and values (V), which are used for defining the sets of attributes (A) and federated attributes (F). The relationships between the elements of our model are described by defining the *attribute assignment* (AA) and *permission assignment* (PA) relations, as well as our proposed AD-Rules.

The local and federated policies proposed in our approach are modeled by the POL set, which is a subset containing entries belonging to the PA relation. The definition of AD-Graphs is based on the concepts of graph theory and the definition of AD-Rules. The access control decision process is modeled by functions *provisionAttrs*, *expectedAttributes*, *relevantPolicies*, *evaluate*, *combine*, and *checkAccess*. Function *provisionAttrs* calculates the set of attributes that can be provisioned from a given AD-Graph based on the local and federated attributes initially exhibited by a set of access control entities. Function *expectedAttributes* returns the set of attributes that are related to a given policy, by inspecting the POL set. Function *relevantPolicies* returns the set of policies that are relevant to a given access request, e.g., the policies that may grant the permission contained in such request. Function *evalu-*

ate returns *true* if a set of attributes provided as an input contains the ones defined for a given policy as returned by the *expectedAttributes* function. Otherwise, the function defaults to *false*. Function *combine* aggregates the results of evaluating the policies relevant to a given access request as returned by the *relevantPolicies* function. The actual implementation of this function may take into account the specific requirements depicted by each collaboration. As an example, some projects may find convenient to implement the *deny-overrides* (OASIS Standard (2013)) combination strategy, which returns *false* if one or more elements in the input set are *false*. Finally, function *checkAccess* implements the authorization checking functionality by invoking the aforementioned *combine* function on the results of invoking the *evaluate* function on each of the policies returned by *relevantPolicies*. The set of input attributes for the evaluation of each function is obtained by means of *provisionAttrs* by leveraging the set of attributes initially provided as an input to the *checkAccess* function.

4.2.1 Attributes

We define attributes as an abstraction of *security-relevant* properties that are exhibited by access control entities, namely, actors, targets, policies, and any applicable context. Their physical nature, e.g., if the attribute represents a file's metadata or an end-user credential, and the way those attributes are collected from the access control entities, remain dependent on each organizational domain.

As shown in Fig. 4.5, we define attributes to have the following three components: (1) a data *type*, which restricts the nature and the possible range of values defined for the attribute, (2) a *name*, which is later used for defining AD-Rules on them and is defined in the context of a given inter-organizational setting, and (3) a *value*, which is used when evaluating such AD-Rules. Examples of attributes include: <Double, *data.size*, 100.0>, <String, *data.source*, "*server.Net 1*">, and <Date, *system.date*,

- ACT, the set of actors.
- TAR, the set of targets.
- CON, the set of context instances.
- OPER, the set of operations.
- P $\subseteq$ TAR $\times$ OPER, the set of permissions.
- E = ACT $\cup$ TAR $\cup$ CON, the set of access control entities.
- N, the set of names.
- V, the set of values.
- T, the set of data types.
- A $\subseteq$ T $\times$ N $\times$ V, the set of attributes.
- F $\subseteq$ A, the set of federated attributes.
- AA $\subseteq$ A $\times$ E, the *attribute assignment* relation mapping attributes with a given access control entity.
- PA $\subseteq$ P $\times$ A, the *permission assignment* relation mapping permissions and attributes.
- POL $\subseteq 2^{PA}$, the set of local and federated policies for access management purposes.
- ADR = { r | r: $2^A \rightarrow 2^F$}, the set of attribute derivation rules mapping sets of attributes to sets of federated attributes.
- ADG, the set of directed, weakly connected, and possibly cyclic attribute derivation graphs. A graph g = <NODES, ARCS> $\in$ ADG if NODES $\subseteq 2^A$ and ARCS $\subseteq$ ADR. We say (n_1, *arc*, n_2) $\in g$ if n_1, $n_2 \in$ NODES and *arc* $\in$ ARCS and $n_1 \subseteq$ *domain*(*arc*) and $n_2 \subseteq$ *codomain*(*arc*).

Figure 4.5: A Model Description of Our Approach (I).

"*11-01-2016*">.

4.2.2 Federated Attributes

We envision our proposed *federated* attributes as a special case of static (non-modifiable), widely-recognized, fully-trusted and possibly custom-defined attributes

- *provisionAttrs*: $2^A \times$ ADG $\rightarrow 2^A$, a function mapping a set of input attributes I with the set of attributes that can be provisioned from a given AD-Graph *adg*. An attribute f is said to be provisioned if there exists a set of attributes $I' \subseteq I$ and a set of paths $P = \{p,\ p = x_0,\ x_1,\ ...x_n,\ \text{n} \geq 0\} \in adg$ such that $\forall$ p $\in$ P, $x_0 \in I'$ and $x_n = f$, and $\forall\ x_i,\ x_j$ in p, $1 \leq i < n$, $j = i + 1$, $\exists\ r \in$ ADR such that $x_j \in r(x_i)$.
- *expectedAttributes*: POL $\rightarrow 2^A$, a function returning the set of attributes that are related to a given policy *pol*. Formally, returns all $a \in$ A such that $(p,\ a) \in pol$.
- REQ = $\{req = <act,\ p = <tar,\ oper>>|\ act \in$ ACT, $p \in$ P$\}$, the set of access control requests, allowing an actor *act* to request for a permission p.
- *relevantPolicies*: REQ $\times\ 2^{POL} \rightarrow 2^{POL}$, a function returning the set of policies that are relevant to a given request $req = (act,\ p)$ given an input set of policies *polSet*. Formally, returns all $pol \in polSet$ such that $(p,\ a) \in pol$.
- *evaluate*: POL $\times\ 2^A \rightarrow \{true,\ false\}$, a boolean function that checks if the set of attributes *attrs* satisfy a given policy *pol*. Formally, returns *true* if there exists a set $S = \{e = (p,\ a),\ e \in pol,\ a \in attrs\}$ and $|S| > 0$, or *false* otherwise.
- *combine*: $2^{\{true,false\}} \rightarrow \{true,\ false\}$, a boolean function that combines the results of evaluating a set of policies. May be implemented based on the specific objectives of each collaborative project.
- *checkAccess*: REQ $\times$ ADG $\times\ 2^{POL} \times\ 2^A \rightarrow \{true,\ false\}$, a boolean function that checks if a given request *req* should be granted or denied based on an AD-Graph *adg*, a set of policies *polSet* and a set of input attributes *I*. Formally, the *combine* function is invoked taking as an input the results of evaluating the set of policies *relSet* obtained from *relevantPolicies(req, polSet)*. A policy $pol \in relSet$ is evaluated using both the set of input attributes I and the attributes obtained from *provisionAttrs*(I, *adg*).

Figure 4.6: A Model Description of Our Approach (II).

that become relevant within a certain security context. They may be generated by processing either local or other federated attributes, e.g., applying an access control constraint on attribute values. Moreover, they may be used to provide a representation of more abstract concepts such as security states, capabilities, clearance labels, organizational jobs (e.g., *roles*), etc. Federated attributes are obtained by processing local attributes from access control entities under a given organizational domain. Such processing is to be modeled through the AD-Rules, thus allowing federated attributes to be related to access rights (permissions).

As an example, AD-Rules may provide functionality intended to validate a given local attribute by inspecting its value component and producing a proper federated attribute as a result. Thus, a validated federated attribute ensures that a given collaboration state remains secure.

As described before, permissions can be assigned to federated attributes, which then serve as a layer of association between local attributes and permissions defined in another organizational domain for collaborative purposes. Such a layer helps identify the local attributes that may be involved in granting a given inter-domain permission, as well as the set of constraints represented by AD-Rules that may be involved in such a process. Moreover, our approach allows for AD-Rules to take federated attributes as an input or may also take both local as well as federated ones as an input to produce federated attributes as a result, as depicted in Fig. 4.4, thus allowing for expressing richer inter-domain policies based on processing already existing federated attributes.

4.2.3 Attribute Derivation Rules and Graphs

As introduced before, *attribute derivation rules* (AD-Rules) are expected to provide a mapping between local attributes and federated attributes. For this purpose,

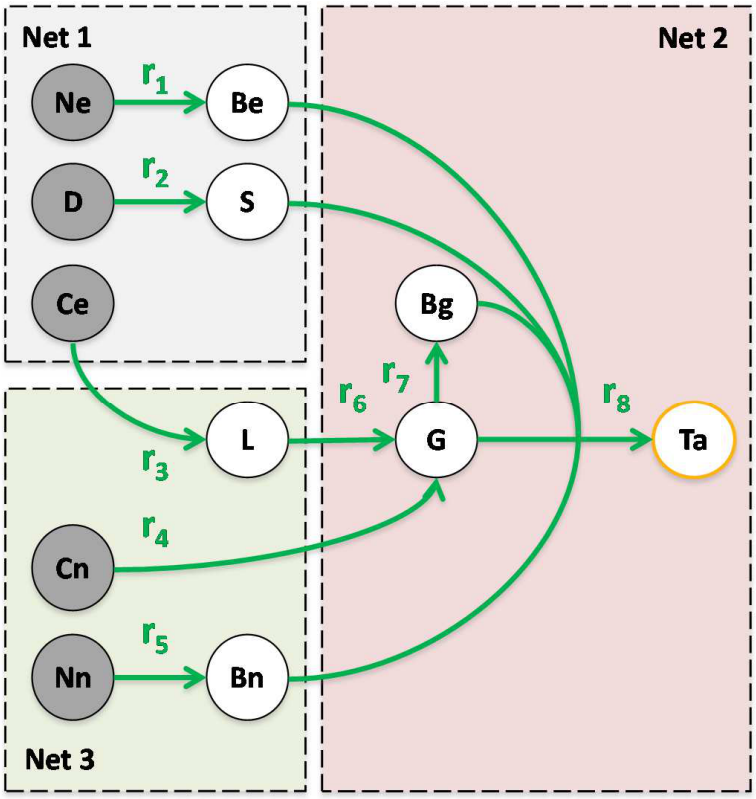

Figure 4.7: A Distributed AD-Graph Depicting the Sample Policy of Fig. 2.2.

AD-Rules are said to be *non-injective* * , as two or more elements from an input set of attributes (domain) may be mapped to the same element in the output set (co-domain). In addition, AD-Rules can be *chained* together to produce a graph-like structure showing how attributes can be provisioned. Such *attribute derivation graphs* (AD-Graphs) are *directed*, because AD-Rules represent unidirectional edges (due to their nature as functions). Moreover, AD-Graphs are also *weakly* connected, as there is no requirement for all nodes (attributes) to be connected to each other. Finally, AD-Graphs are also possibly *cyclic*, as a customized chaining of AD-Rules may end up introducing a cycle in the produced AD-Graph.

As an example, Fig. 4.7, presents a graphical depiction of a set of AD-Rules

*A function f: A $\rightarrow$ B is said to be *injective* or *one-to-one*, $\forall$ a, $a'\in$ A, a $\neq$ a'$\Rightarrow$ $f(a) \neq f(a')$.

combined together to create an AD-Graph following our running example, allowing for different attributes to be obtained by transforming other attributes taken as an input. For instance, in such a Figure, attribute G can be generated by means of the AD-Rules labeled as r_4 and r_6.

AD-Graphs may also support collaborative processing by allowing a division into proper subgraphs, each subgraph implemented in a different security domain: as mentioned before, each participating domain is in charge of defining its own permissions, local and federated attributes, as well as the AD-Rules and AD-Graphs to generate those. AD-Graphs can be modeled as a distributed graph: a given AD-Graph G defined for a federation F may be divided into a set of subgraphs G'_1, G'_2, ... G'_n, such that each G'_i is to be processed by a different domain in F.

4.2.4 Attribute Provisioning

As described before, our approach is intended to provide efficient provisioning (discovery, derivation, and communication) of attributes within a federated setting. These attributes may be in turn obtained from different entities such as end-users, shared resources, execution environments, etc. In addition, attributes may also provide a representation of security-related properties obtained from different layers of *abstraction* within a given organizational domain, e.g., network configurations and services, software applications, or personal data, while others may be *synthetically* created based on federated policies, e.g., security clearance levels.

With this in mind, attribute provisioning is crucial to handle access requests in the context of inter-organizational resource sharing. Such a process includes allowing for participating organizations to know about the AD-Rules that are implemented by other organizations and are involved in a given AD-Graph G. Concretely, participants need up-to-date information so that they can extract correct paths within G that can

produce the desired federated attributes. With this in mind, attribute provisioning can therefore be divided into two process: path discovery and path traversal.

The *path discovery* process allows for each organization to distribute information about its locally-implemented AD-Rules to the federation, so that they can potentially maintain a representation of G for path calculation. However, there are several practical challenges: first, each organization needs to be notified when changes to G occur, e.g. adding or removing a given AD-Rule, which may create a large set of communication messages between participants. Second, there is an added maintenance cost, e.g. processing time, that participating organizations must incur for handling and maintaining an up-to-date G. Finally, storage efficiency may become an issue when a large G must be locally maintained. An alternative approach would be creating a central database storing G, along with a set of replicas for enhanced availability. However, such a scheme may suffer from service bottlenecks and consistency issues when communicating updates to the replicas. In addition, a centralized server may become the subject of a DoS attack, which could certainly limit the availability of the overall attribute provisioning scheme, thus potentially preventing participating organizations from serving inter-organizational access requests. With this in mind, there is a need for a distributed approach that allows for participating organizations to release information about the AD-Rules they implement in such a way that the administration burden, e.g., number of communication messages, is significantly reduced. In addition, such an approach should also prevent organizations from having to store a complete AD-Graph locally for path discovery purposes and should provide support against attacks targeting a single point of failure. We present an implementation tailored for meeting such goals in Chapter 5.

Following the model described in Fig. 4.5 and Fig. 4.6, the *path traversal* process allows participating organizations to invoke the AD-Rules included in a given path

p in G that may ultimately produce a given federated attribute. Invocation of such AD-Rules should be done by following a sequence starting from the first AD-Rule in p up to the last one. Each time an AD-Rule is executed, the produced set of attributes is added to a set of input attributes for the next AD-Rule in the sequence. In addition, the invocation of an AD-Rule r enables to locate the federated domain implementing r, the set of input attributes, as well as the set of produced attributes. A request for the invocation of r should include the set of attributes that serve as its input. Finally, the attributes produced by r, if any, should be then communicated back to the requesting organization.

Leveraging the definitions shown in Fig. 4.5 and Fig. 4.6, the problem of resolving an access request to a shared resource within a federation can be modeled as a path traversal problem within a given AD-Graph: determining if there exists a path between a set of starting nodes (local attributes) and an ending node (federated attribute), and using the obtained attributes to evaluate the policies that grant the requested permission over the desired resource. A procedure for resolving an access request *Request*, derived from the model shown in Fig. 4.4, is shown in Fig. 4.8: given an AD-Graph *Graph*, a set of policies *Policies*, and a set of input attributes *InputAttrs*, the procedure starts by obtaining the set of policies that are relevant to *Request*, e.g., the ones granting the permissions for the resources listed in *Request* (Line 2). Then, each relevant policy is evaluated against the set of input attributes (Line 4). If such evaluation fails, the procedure invokes the auxiliary function *getPaths*, which inspects *Graph* to obtain a set of paths starting with an attribute in *InputAttrs* and ending with an attribute in *expectedAttrs* (Lines 6–8). Subsequently, each path is traversed by executing each of its included AD-Rules in sequential order. If new federated attributes are generated, they are aggregated into the *provisionedAttrs* set, which is then provided as an input for the evaluation of currently inspected *policy*

```
1: procedure CheckAccess(Request, Graph, Policies, InputAttrs)
2:     relevantPolicies ← relevantPolicies(Request, Policies)
3:     for each policy in relevantPolicies do
4:         result ← evaluate(policy, InputAttrs)
5:         if result = false then
6:             provisionedAttrs ← InputAttrs
7:             expectedAttributes ← expectedAttributes(policy)
8:             paths ← getPaths(InputAttrs, Graph, expectedAttrs)
9:             while hasNext(paths) do
10:                path ← next(paths)
11:                provisionedAttrs ← provisionAttrs(path, provisionedAttrs) ∪ provisionedAttrs
12:                result ← evaluate(policy, provisionedAttrs)
13:                if result = true then
14:                    break while
15:                end if
16:            end while
17:        end if
18:        results ← results ∪ result
19:    end for
20:    return combine(results)
21: end procedure
```

Figure 4.8: Algorithm for Checking a Resource Access Request.

(Lines 9–16). Finally, policy evaluation results are combined for a final decision (Lines 18–20).

As an example, the AD-Graph in Fig. 4.7 implements the inter-organizational policy described in Fig. 2.2 as follows: each participant is responsible for distributing information to partnering organizations about the AD-Rules they implement, including the input and output attributes (either local or federated), as well as the way to invoke them, e.g., by providing a web service interface. As an example, Net 1 will

publish information on the AD-Rules labeled as r_1 and r_2. As described before, participants use such information during the path discovery phase to effectively locate the AD-Rules that can produce a given federated attribute. For instance, when attribute G is requested, the sequence of AD-Rules composing the different paths leading to it will be retrieved from the information published by all participants. Once such paths are located, the path traversal process can be carried on. Continuing with our example, attribute *Ce*, which depicts a locally-issued credential in the context of Net 1, is transformed by the AD-Rule labeled as r_3 into the federated attribute L that features membership to a research group L known to Net 1 and Net 3. Following our definition depicted in Fig. 4.5, *Ce* may be modeled as an attribute of a custom-made type *Net 1.Credential*, whose domain may be defined as a finite set of strings that follow a certain previously-defined formatting. In a similar fashion, L can also be defined as an attribute of a type named *Net 1.LocalGroup*, which may in turn depict a domain composed of a single element denoting group membership. Subsequently, r_3 will then perform a mapping between a subset of the elements (strings) in the input domain to a single element in the output domain. In practice, r_3 may be implemented by retrieving the list of group members from a local database file. Attribute L is subsequently processed by the AD-Rule r_6 in the Net 2 domain, producing the federated attribute G, which in turn depicts membership to an inter-organizational collaborative group. Following our previous intuition, G can be modeled as an attribute of type *FederatedGroup*, whose domain is composed of a single element. Recall that following the discussion introduced in Chapter 4.1, G is expected to be known to all participants of a given collaboration, e.g., the ones depicted in our running example. With this in mind, r_6 may provide a simple mapping between an input and an output domain both consisting of a single element. Later, attribute G, along with attributes S, *Be*, *Bg*, and *Bn*, is forwarded to the AD-Rule labeled as r_8, producing

the *Ta* attribute as a result.

Chapter 5

A DISTRIBUTED ENFORCEMENT MECHANISM

Inspired by recent approaches such as OpenID (Recordon and Reed (2006)), Shibboleth (Morgan *et al.* (2004)), OAuth (Jones and Hardt (2012)) and Facebook Login (Facebook Inc. (2015)), which are mostly intended for federated security in the context of authentication, we have devised an enforcement mechanism for the model presented in Chapter 4 that primarily relies on a distributed architecture deployed over the participants of collaborative projects. We start in Chapter 5.1 by providing a description of our proposed *access management agent* (AMAs), which are expected to implement most of the functionality devised for our approach within the context of a single participant organization. We then move on to provide in Chapter 5.2 details on a proposed implementation for the *path discovery* and *path traversal* procedures detailed in Chapter 4.2.4, which include a description on how the AMAs belonging to different organizations are to interact with each other. In addition, in Chapter 5.2.3, we detail our experimental results on a set of real-life collaborative policies derived from cases similar to the one featured in our running example. Finally, Chapter **??** describes how the enforcement mechanism introduced in this chapter meets the challenges described in Chapter 3.3. Similarities and differences of our approach with the aforementioned existing technologies are to be discussed as a part of our description of related work in Chapter 8.

5.1 Access Management Agents

Following our running example depicted in Fig. 2.1, we envision each participating organization implementing a dedicated software module to be referred as *access management agent* (AMA) in the rest of this dissertation. Such an agent module will be in charge of handling the functionality devised for our approach by implementing five different layers of abstraction, as shown in Fig. 5.1: the *policy administration* layer is expected to provide functionalities for the retrieval, creation, update and removal of both local and federated policies. For such a purpose, it relies on both the *policy discovery* and *attribute discovery* layers in charge of locating policies and attributes, respectively, which may be physically located within the local domain or in a remote one. In addition, the *policy evaluation* layer will be in charge of resolving resource access requests by first locating the set of policies that are deemed as *relevant*, e.g., the ones that mediate access to the requested resource, and by provisioning all attributes enlisted in such policies by means of the *attribute derivation* layer. This layer will be in charge of locating and provisioning attributes both in the local domain as well as in external ones by issuing proper requests to other AMAs implemented by remote peers.

5.1.1 Responsibilities for Participants

By implementing our proposed AMAs, each participant organization will be entitled the following:

Attribute Implementation. Participants are to define the federated attributes required for each policy guarding access to each shared resource, either by leveraging federated attributes that are in turn provided by another participant or by implementing federated attributes originating in the local context, or by any combination

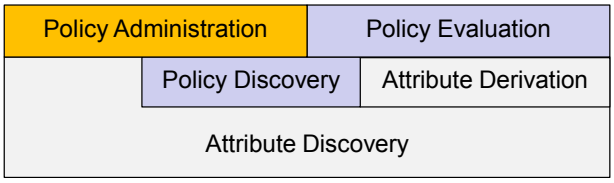

Figure 5.1: A Layered-based Depiction of an Enforcement Mechanism for FAM.

of the options just described.

Trusting Attributes of Others. Each federated participant is to trust the attributes originated/managed under its local domain. In addition, participants in a federation are to fully trust the attributes *provided* by other peers, assuming they trust the participant providing such attributes. Chapter 7 will present a framework for allowing participant to determine the peers they trust even when they may have had no previous interaction with them.

Attribute Derivations. Finally, each participant is to define the attribute derivation transformations for the federated attributes under its control and provide an implementation for them following the architectural layer defined as *attribute derivation*. Such a process also includes allowing other participants to discover such attributes by *publishing* information about them in the *attribute discovery* layer.

Policy Administration and Retrieval. Finally, following the *policy administration*, *policy discovery* and *policy evaluation* layers described before, participants should define and provide means for the administration of both local and federated policies that mediate access to resource sharing devised in collaborative projects. In addition, participants should retrieve policies relevant to a given local or federated request issued either within the local security domain or by another participant peer. Such policy retrieval should be implemented as efficiently as possible, e.g., physical

storage and response time.

5.1.2 Source Code Implementation

Our *proof-of-concept* implementation consists of 146 Java classes and 52 interfaces that are grouped into 29 packages and combine for a total of 9,238 lines of code, as reported by the `cloc` tool (Danial, Al (2015)). Following the approach introduced before, functionality comprising our approach is carried out by software agents known as AMAs running on each participant network. With this in mind, we have developed a dedicated AMA that can handle the management and the evaluation of federated and local policies, as well as the provisioning of federated attributes.

Policy Administration Layer. The creation, update, and removal of policies is implemented by means of a *policy administration point* (PAP) that leverages a customized version of the well-known *extensible access control markup language* (XACML) language (OASIS Standard (2013)). Such customization depicts a set of policy rules and attribute selection functions that better suit policies such as the one listed in our running example. For instance, we have introduced new attribute selection functions to handle access tokens such as the *Ta* attribute shown in Fig. 4.7. Policies are parsed and stored in a repository, which is based on a relational database. When a new policy is added to the repository, policy targets are extracted and used for indexing purposes. In addition, policy rules are transformed into *serialized* Java objects and are stored in the database for further retrieval.

Policy Discovery and Evaluation Layers. Evaluation of local and federated policies is implemented by a *policy decision point* (PDP) and a *policy information point* (PIP) modules. Upon an access request, the PIP retrieves all relevant policies

from the repository by using the targets listed in the request as search conditions. For all relevant policies, the set of rules is retrieved and *deserialized* into Java objects depicting our customized version of the XACML language. This way, a given policy is not continuously parsed by XACML tools every time it is found relevant for evaluation. For such a purpose, we have extended the Sun XACML 1.0 implementation (Sun Microsystems, Inc. (2015)). Our policy repository is implemented by means of a MySQL CE Server (Oracle Corporation (2015)). In addition, the interaction between the AMA and outside requesters is provided by means of a dedicated web service that leverages the Apache CXF framework (The Apache Software Foundation (2015)).

Attribute Discovery and Derivation Layers. We implemented the DHT functionality discussed in Sec. 5.2.1 and Sec. 5.2.2 by implementing a DHT module that is included as a part of the modules performing policy evaluation tasks. Such a module leverages the Open Chord 1.0 API (Kaffille and Loesing (2015)): an open source implementation of the Chord DHT (Stoica *et al.* (2001)) that allows for remote peers to implement a DHT ring by communicating over TCP/IP sockets. In addition, our proposed AD-Rules, as discussed in Chapter 4.2.3, were implemented by leveraging a client-server architecture over TCP/IP sockets with the standard `java.net` package.

An architectural depiction, focused on our running example, is shown in Fig. 5.2: an connection request between Net 1 and Net 3 is handled by the *policy decision point* (PDP) module implemented by AMA_2 on top of Net 2 (1). The set of policies relevant to a request is obtained by the *policy information point* (PIP) module by querying a local database repository (2). Following our running example, this set contains the policy described in Chapter 2.2. Next, such a policy is *parsed* and the list of referenced attribute names is forwarded to the local DHT module for provisioning purposes (3). Following Fig. 5.3, DHT modules within each AMA implement a *ring*

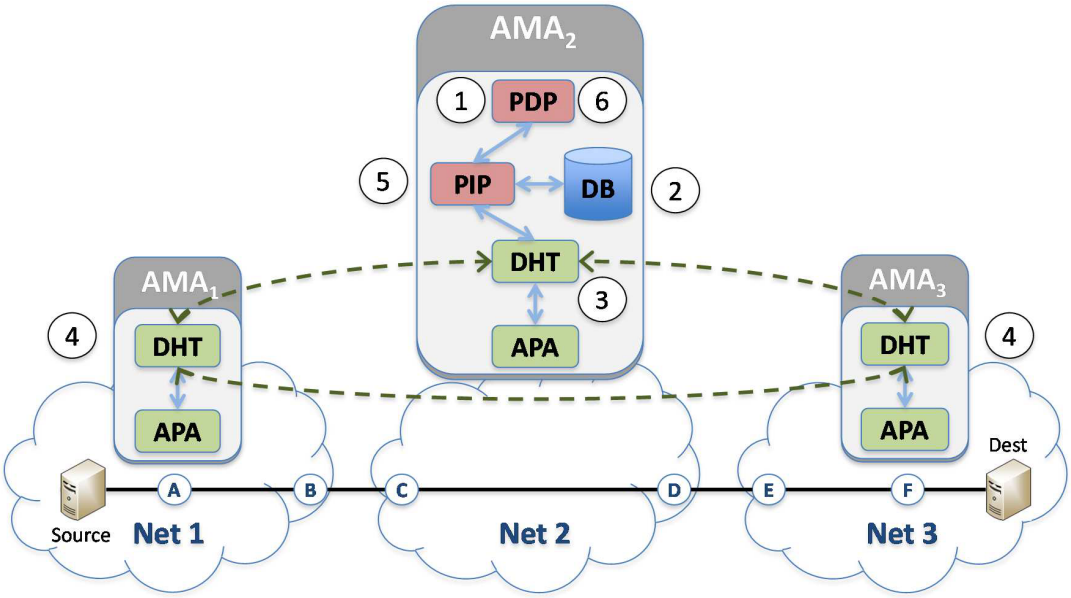

Figure 5.2: Evaluating an Access Management Policy.

for locating and provisioning federated attributes. When requested, attributes are provisioned by the DHT modules implemented by the AMAs on top of Net 1 and Net 3 respectively by means of *attribute provisioning agents* (APA), which are dedicated modules with trusted access to local infrastructure, e.g., credential-based systems, within each participating network (4). Provisioned attributes (if any) are sent back to the PDP within Net 2 for evaluation (5) and (6).

5.2 Distributed Enforcement

In this section, we present an implementation of the *path discovery* process presented in Chapter 5.2.1 that is based the concept of *distributed hash tables* (DHT) (Stoica *et al.* (2001)). In addition, we discuss our implementation on the *path traversal* process shown in Chapter 5.2.2 that is based on a client-server architecture for the remote invocation of our proposed AD-Rules.

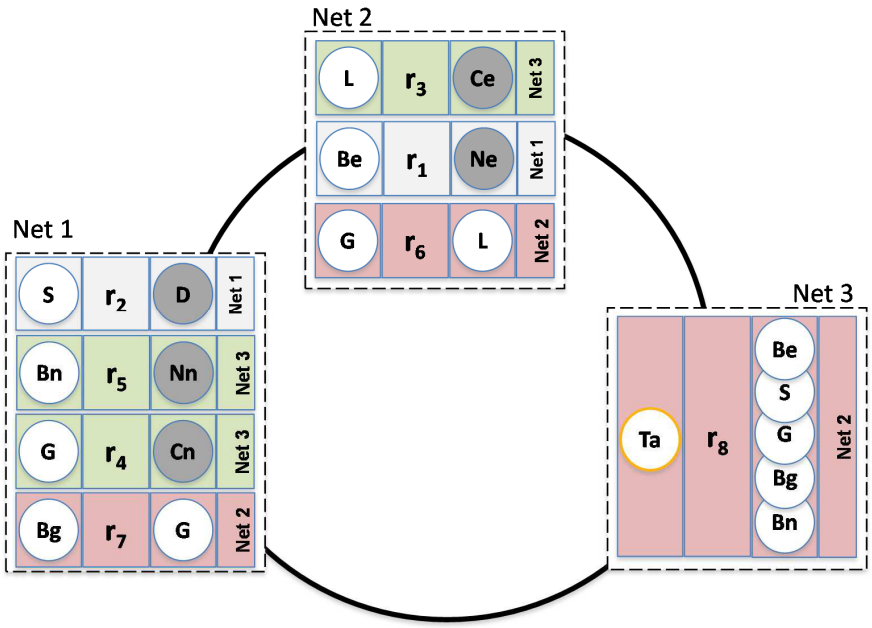

Figure 5.3: An Illustrative DHT *Ring* Depicting the AD-Graph of Fig. 4.7

5.2.1 Path Discovery

Fig. 5.3 illustrates the path discovery process based on our running example. We allow for participants in a federation F to join a DHT *ring* to publish and retrieve information about the AD-Rules that may produce federated attributes. This process may be in turn decomposed into two inner components, namely, AD-Rule publishing and AD-Rule retrieval.

The procedure for publishing an AD-Rule is conducted as follows: each domain is in charge of inserting an *entry* into the DHT for each AD-Rule they implement for a given AD-Graph under the context of F. Such an entry should include information about the input attributes (either local or federated ones), the name of the AD-Rule, and the set of federated attributes to be produced as a result. Moreover, some information on how to *execute* such AD-Rule should be also provided, e.g., a *universal resource locator* (URL). As an example, the Net 1 domain will publish

an entry into the DHT containing information about the AD-Rule r_1, including the local input parameter Ne, which conceptually depicts information about the current state of the local network, and the federated attribute Be, which provides a standard representation of the current bandwidth capacity. In addition, such an entry should contain a valid URL for other federated peers invoking the AD-Rule r_1 remotely. Following the insertion procedure for DHTs (Stoica *et al.* (2001)), such an entry may end up being stored for future location at a different federated peer, following a hashing scheme based on the standardized naming convention for federated attributes introduced in Chapter 4.2.1. In Fig. 5.3, the entry for the AD-Rule labeled as r_1 (published by Net 1) ends up being stored by the DHT node under the scope of the Net 2 domain. Conversely, AD-Rules may be retired from a given AD-Graph by removing their corresponding entries from a given DHT ring. Recall such procedure may not necessarily remove the production of federated attributes in the context of an AD-Graph, as such attributes may be produced by another AD-Rule in the DHT ring, e.g., in Fig. 5.3, removing the entry for the AD-Rule r_6 does not prevent an attribute G from being produced by the AD-Rule labeled as r_4.

The retrieval procedure for entries containing information about AD-Rules is to be conducted as follows: a participating organizational domain O interested in producing a given attribute A may retrieve the set E of entries corresponding to A in the DHT ring, e.g., by hashing the A's identifier. Then, by inspecting the information about AD-Rules contained in E, O must determine if there exists a local or federated attribute under its local domain that can be used as an input parameter to an AD-Rule to produce A. If so, information from the corresponding entry in the set E is retrieved and the AD-Rule is invoked, following the procedure to be detailed in Chapter 5.2.2. However, if no suitable entry is found, e.g., all input attributes to the entries in E are out of scope or cannot be locally produced, O may attempt to explore

the DHT ring once again for entries producing the attributes taken as an input to the entries in E, thus potentially producing a set P of graph paths in an AD-Graph stored in the DHT. Such a process may be repeated up to the point when no more entries can be obtained from the DHT or a cycle in the AD-Graph stored in the DHT is detected, e.g., when an iteration retrieves entries that were previously retrieved in the past, or a path can be traversed. A path in P is traversed, e.g., by calling the sequence of AD-Rules contained in it, only if it starts with an attribute under the scope of O and ends with the desired attribute A.

Considering our running example, an entity under the Net 1 domain may provision an attribute *Ta* depicted in Fig. 4.7 as follows: the DHT featured in Fig. 5.3 retrieves the entry for the AD-Rule labeled as r_8 from the ring node implemented by Net 3. As the input parameters of r_8 are all federated attributes, Net 1 inspects the DHT ring once again for determining proper AD-Rules provisioning those attributes. Then, entries generating *Be* (r_1), *Bg* (r_7), *Bn* (r_5 and r_9), *S* (r_2) and $G(r_4, r_6)$, are returned. For the federated attribute *Be*, Net 1 can provide the local attribute *Ne* required for r_1, thus creating a *traversable* path within the distributed AD-Graph. In addition, for the federated attribute S, Net 1 can also provide the required local attribute D required for r_2, thus creating a path as well. In the case of G, the entry belonging to r_4 may be discarded as its input attribute (*Cn*) is local only to Net 3. However, in the case of the entry for r_6, Net 1 may inspect the DHT ring once again for an entry producing the input attribute L. Next, the entry for r_3 is returned taking *Ce* as an input. Since *Ce* is local to Net 1, another traversable path is constructed. With respect to an attribute *Bn*, the AD-Rules labeled as r_5 can be also discarded as its input attribute (*Nn*) is local to Net 3. However, r_9 can be used as it takes the federated attribute G as an input, and a path producing G has been already obtained. Similarly, an attribute *Bg* can be obtained from r_7 as such

an AD-Rule takes G as an input. The setting depicted in Fig. 4.7 and Fig. 5.3 allows for the AD-Rules labeled as r_7 and r_9 to disclose network-related information, e.g., bandwidth, only when membership to an inter-organizational project (as depicted by the G attribute) can be shown.

5.2.2 Path Traversal

Our implementation supports the process of path traversal by allowing for each organizational domain O to implement a software agent that is capable of handling requests for the invocation of the AD-Rules that are under the scope of O. Information on locating such agent and invoking the implemented AD-Rules should be consistent with the entries published in the DHT ring described in Sec. 5.2.1, e.g., Net 1 may provide a TCP/IP agent that implements the AD-Rule labeled as r_1 in Fig. 4.7 and Fig. 5.3. For a given path P composed of n entries obtained from a DHT ring, the traversal procedure would include requesting for the execution of each entry starting from the entry at the first position and collecting the attributes produced by the AD-Rule being invoked (if any). The process continues as soon as new attributes are produced on every AD-Rule invocation and finishes either when a given AD-Rule depicted by an entry in the path is not able to produce any attributes or the final entry (at position n - 1) has been executed and the final attributes have been produced as a result.

Fig. 5.4 shows a simplified version of a class diagram depicting our attribute provisioning scheme, which allows participating domains to produce their own AD-Rules by developing Java classes that *subclass* our `ADRule` interface. Moreover, attributes, as described in Sec. 4.2.1 and Sec. 4.2.2, can be modeled by classes implementing the `Attribute` interface or subclassing the `AbstractAttribute` class. Entries depicting information related to AD-Rules are modeled by means of the `ADRuleEntry` inter-

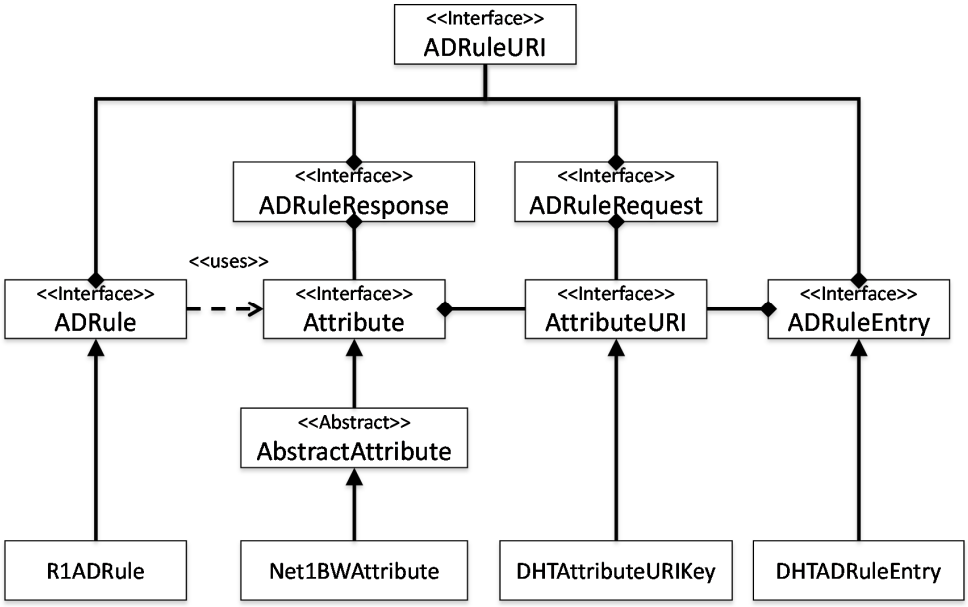

Figure 5.4: A Simplified Class Diagram Depicting our Enforcement Mechanism.

face. Such entries are stored into the DHT ring by using the identifier of the output attribute(s) as a key and a serialized `ADRuleEntry` as the value. As an example, the Java object belonging to the entry depicting information about the AD-Rule labeled as r_1 in Fig. 5.3 (depicted as `R1ADRule` in Fig. 5.4) is serialized and stored into the ring by using *Be* as the hash key. In a similar fashion, the location of such an entry is discovered using the name *Be* as the key and retrieving the serialized object from the ring node in Net 1 (Fig. 5.3). In addition, a path P in a distributed AD-Graph is represented by a sequence of Java objects, `ADRuleEntry`, that have been obtained from a DHT ring. The path discovery process described in Sec. 5.2.1 was implemented by customizing the well-known *depth-first search* (DFS) algorithm (Tarjan (1972)) With this in mind, traversing P involves retrieving the input attributes as well as the information of the domain implementing each AD-Rule from each of the entries in P and requesting for their execution by following P's sequence. Remote execution of an AD-Rule R involves the *client* domain sending a TCP/IP socket request in the form

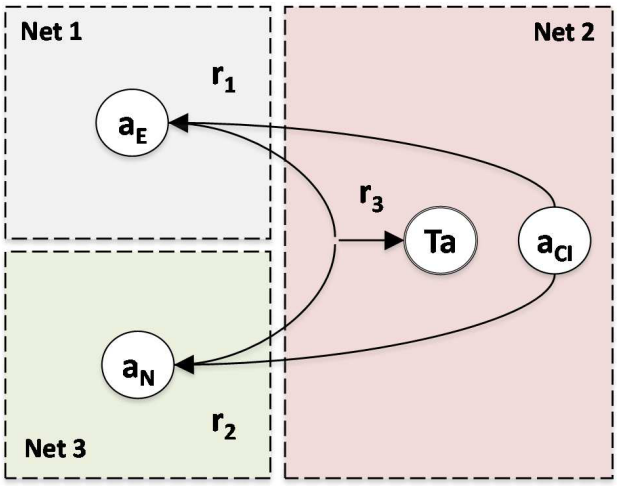

Figure 5.5: The Link Ownership (LO) Policy.

of a serialized class object which is an interface, `ADRuleRequest`. This interface takes the input attributes (objects implementing the `Attribute` interface) devised by R's entry. The *server* domain then attempts to execute R upon the verification of the input attributes contained in the request object, and sends back a response in the form of a serialized object via another interface, `ADRuleResponse`. Such a response object may contain the federated attributes if the input validation and execution of R succeed, or an error message otherwise.

5.2.3 Experimental Evaluation

For our experimental evaluation, we simulated a set of collaborative networks such as the one depicted in Fig. 2.2 by instantiating four virtual machines on a private cloud running the OpenStack controller software (Rackspace (2015)). Each instance was equipped with 2 VCPU, 2 GB RAM and 20 GB of Storage, running Ubuntu Desktop 12.04 (Precise Pangolin).

In our first experiment, we examined a set of sample policies obtained from actual discussions hosted in the context of the OGF/NSI group introduced in Chap-

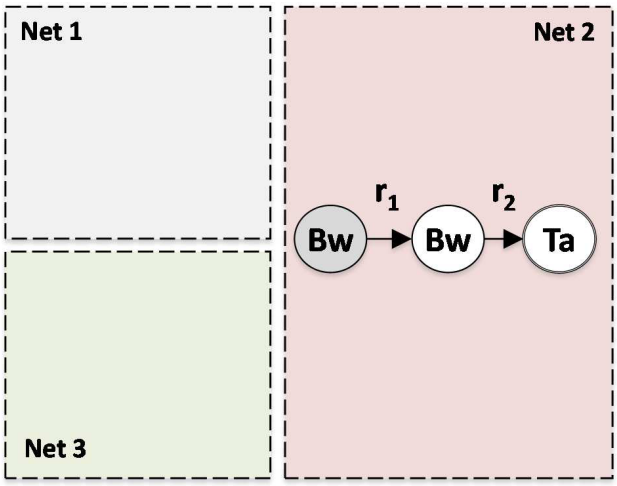

Figure 5.6: The Bandwidth Restriction (BR) Policy.

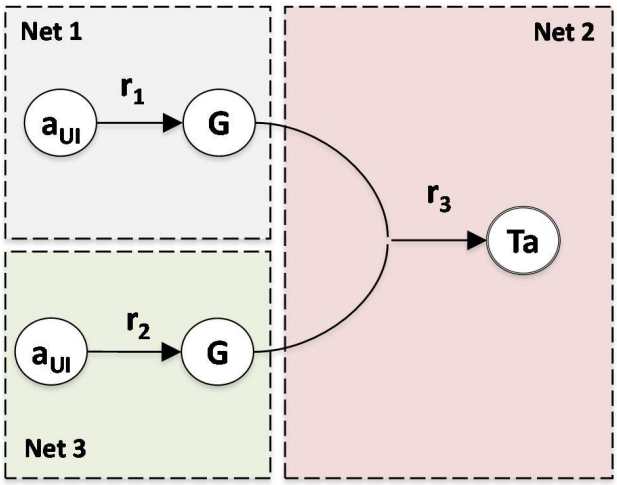

Figure 5.7: The Group Membership (GM) Policy.

ter 2 (Roberts *et al.* (2015)) (Trompert and MacAuley (2015)), all of them depicting a collaborative setting as described in Chapter 3.2, which involves two non-adjacent networks (Net 1 and Net 3) trying to establish a connection by using a third network as a *bridge* (Net 2). First, we have included the policy depicted in our running example, whose implementing AD-Graph is shown in Fig. 4.7. Such a policy leverages attributes obtained from the set of end-users, e.g., attribute G, attributes obtained

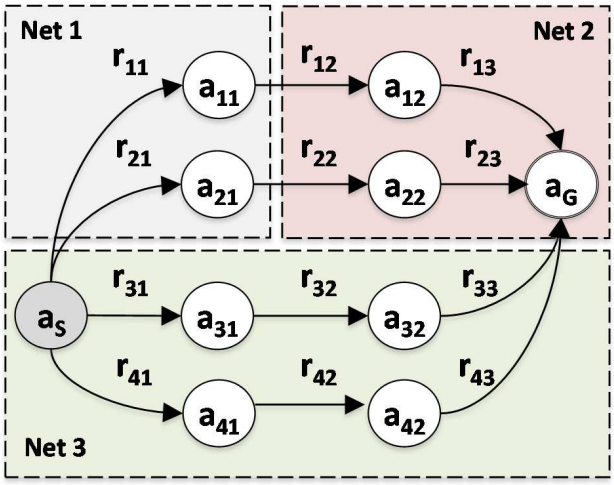

Figure 5.8: A Sample Simulated Policy.

from the set of protected resources, e.g, attribute D and attributes from the running environment, such as the *Be* attribute.

Conversely, Fig. 5.5 depicts policy *Link Ownership* (LO), which allows for the aforementioned connection to take place only when the two interested networks (Net 1 and Net 3) have explicitly expressed their authorization by issuing federated attributes representing access tokens, labeled as a_E and a_N respectively, which are then used by Net 2 to produce the *Ta* required for authorizing the requested connection. As with our running example policy, authorization is modeled through the *Ta* attribute, which, as mentioned before, represents an access token.

Policy *Bandwidth Restriction* (BR, Fig. 5.6) allows for the connection to take place only when a minimum amount of bandwidth is available in the connection links, in an effort to offer a certain QoS. With this in mind, the federated attribute *Bw* can be obtained from local information stored within Net 2. This case depicts an interesting situation where attributes are to be obtained from a single collaborative organization, however, the effects of the federated policy ultimately reside in an inter-organizational scope.

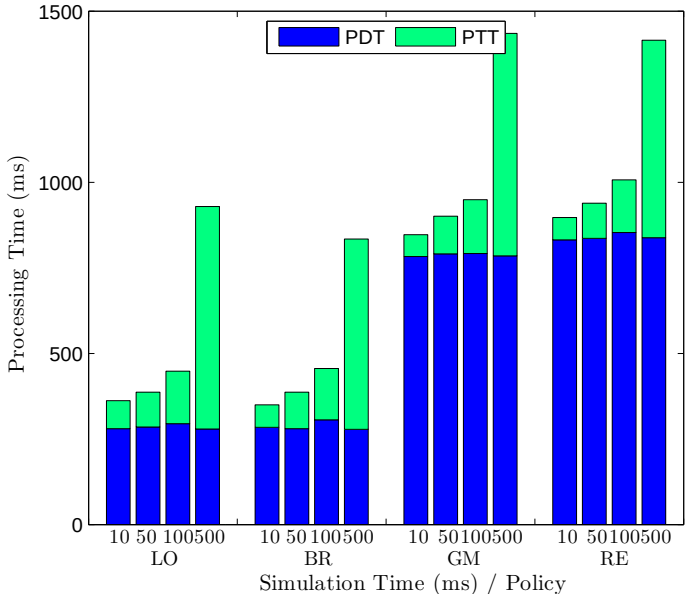

Figure 5.9: Experimental Results for OGF/NSI Policies.

Finally, policy *Group Membership* (GM, Fig. 5.7) allows for establishing the connection only when both end-users at the source and destination end-points are members of a certain collaborative group G. In such a case, a_{UI}, depicting information on the users attempting the connection, serves as an input to the AD-Rules producing G which are in turn implemented by Net 1 and Net 3.

In our experiments, we implemented the policies just described by leveraging our customized version of the XACML language described previously. In addition, entries for the AD-Rules defined by such policies were stored in a DHT ring composed of our simulated AMAs, as shown in Fig. 5.3. Such AD-Rules were in turn implemented by each participating AMA by means of APA software modules such as the ones described in Fig. 5.2. As an example, the APA module of the AMA on top of the Net 1 domain implemented the r_1, r_2, and r_3 AD-Rules of our running example policy as shown in Fig. 4.7, and its DHT module contained the entries belonging to r_2, r_3, r_4, and r_5 as shown in Fig. 5.3.

On each experiment, we manually crafted a request targeting the goal attributes for each policy, e.g., the *Ta* attribute, and issued an access to the AMA_2 implemented by the VM representing the Net 2 network. We measured the *path discovery time* (PDT) for constructing paths within a given AD-Graph, e.g., locating a path from a source attribute to a desired attribute, and the *path traversing time* (PTT) taken for our approach to provision the set of attributes depicted in *P*. We also introduced code to add a variable delay to the execution of each AD-Rule, in an effort to simulate both its *processing* time, e.g., accessing local infrastructure for attribute provisioning, as well as the network latency when working on a distributed setting. We call such code as *simulation time* (ST). In our experiments, the time invested in the retrieval of our sample policies remained under a range of 0 to 3 ms, mostly due to the fact that only a few policies were stored within our database repository. In addition, as our sampled policies required only the provisioning of a single attribute serving as an access token, e.g., *Ta*, time invested in the policy evaluation process was also in a range of 1 to 4 ms.

Results for our first experiment are shown in Fig. 5.9: variation in the overall processing time was mostly due to variation introduced in the PTT. This is due to the ST code we introduced as a part of our experiments, as well as to the fact that the number of DHT entries belonging to each of the policies under test remains the same, thus provoking the PDT time to construct paths within the AD-Graphs depicted for our experimental policies to remain within a manageable range.

We designed a second experiment to measure the performance of our implementation against policies depicting AD-Graphs with a varying number of both paths (*branches*) and composing nodes (*links*). As an example, Fig. 5.8 shows a policy made of 4 branches, each of them depicting 4 links. On each experiment instance, we crafted a policy intended to provision the attribute produced by the DHT entry

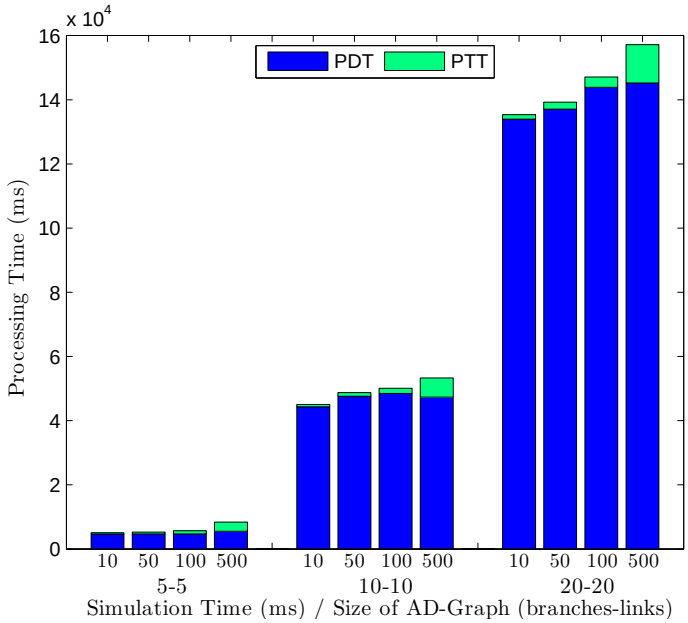

Figure 5.10: Experimental Results for Simulated *Lazy* AD-Graphs.

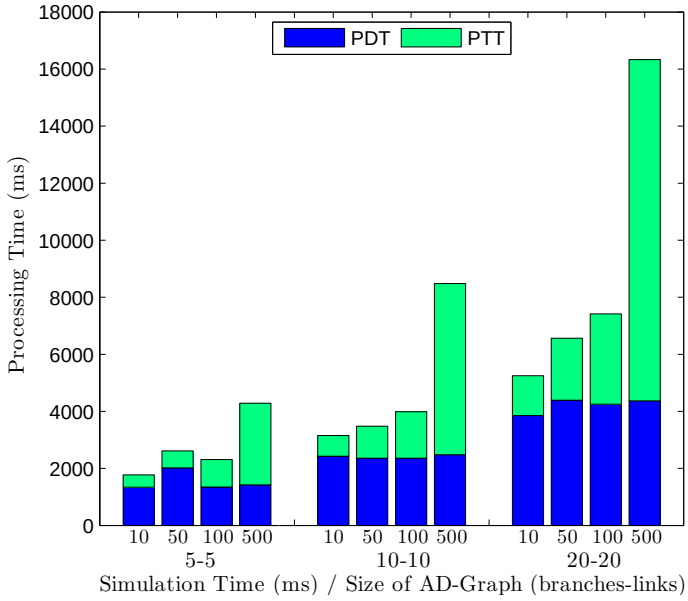

Figure 5.11: Experimental Results for Simulated *Eager* AD-Graphs.

located at the last node of each path in the simulated AD-Graph. We then issued a request for a *required* attribute (labeled as a_G in Fig. 5.8), including as an input the attribute depicted in the entry located in the position 0 of the path (labeled as a_S). In addition, the AD-Rules composing each AD-Graph were randomly distributed within the simulated AMAs that were used in our testbed. As with the previous experiment, we introduced ST code to simulate the execution time of each AD-Rule involved in the AD-Graph as well as the network latency by using a configurable parameter and measured the PDT and PTT as described previously. As with our first experiment, our crafted policies were simplified to include only our goal attribute a_G, and were stored within only a few other policies in the repository, thus making the policy location and evaluation time negligible with respect to the PDT and the PTT.

Fig. 5.10 shows our results when constructing AD-Graphs of size (b-l) where b stands for the number of branches and l stands for the number of links on each AD-Graph, e.g., the first four-column set shows the processing time observed when constructing AD-Graphs of size (5-5), and varying the ST to have values of 10, 50, 100, and 500 ms. In this experiment, the processing time consists mainly of the PCT, as the process of constructing paths is significantly affected by the different number of possible paths within the simulated AD-Graph that need to be explored when attempting to provision the a_G attribute having the a_S attribute as a starting point. Since our *lazy* approach first calculates the set of paths within the AD-Graph before starting the path traversal process, path discovery may represent an important bottleneck for highly-dense AD-Graphs. An alternative approach would consider attempting the path traversal process as soon as a new path P in a given AD-Graph G is constructed, by letting a dedicated subprocess handle the traversal of P whereas the main process keeps exploring paths in G. As soon as a new path in G is identified, a new subprocess is created. If one of these subprocesses is able to provision the goal

attributes, e.g., a_G, the main process, as well as any created subprocesses, are halted and policy evaluation may be carried on next. We implemented this *eager* approach and the experimental results are shown in Fig. 5.11. As expected, the PDT is greatly reduced, as it is halted as soon as a path in the sampled AD-Graph is constructed and traversed.

Chapter 6

AN ASSURANCE AND CONFORMANCE FRAMEWORK

As introduced in Chapter 3.4.1, the proper implementation of the enforcement mechanism described in Chapter 5 is crucial to guarantee the approach proposed in this dissertation indeed preserves important security properties related to authorization, e.g., preventing unintended access to sensitive resources and also preventing previously-authorized accesses to be incorrectly denied at runtime. We start in Chapter 6.1 by providing an introduction to the construction of enforcement mechanisms as the one depicted in Chapter 5 by means of leveraging well-known techniques in the field of software engineering. Then, we move on to present some additional background on specification techniques for software construction in Chapter 6.2, which we leverage in Chapter 6.3 for the purposes of our approach. Finally, we provide experimental evidence supporting our proposed solution in Chapter 6.4. Later on, in Chapter 9.2.1, we provide a discussion on how the approach presented in this chapter meets the requirements for assurance and conformance described in Chapter 3.4.

6.1 Assertion-based Software Construction

The problem for providing correct implementations of the approach presented in this dissertation gets complicated by the existence of customized source-code constructs, which may in turn leverage existing *in-house* systems belonging to participant organizations, as it has been also described before as a part of the initial requirements for our collaborative settings. In addition, incorrect implementations of our proposed AD-Rules and AD-Graphs, as depicted in Chapter 4.2.3, may also introduce non-

trivial security vulnerabilities, as they may allow for unintended attribute derivations to take place, thus potentially producing attributes in a wrongful way, which can ultimate affect the evaluation result of a given authorization policy (either local or federated). With this in mind, we now present an approach tailored to assist developers and testers when constructing enforcement mechanisms based on the ideas discussed in this dissertation. We start in Chapter 6.2 by extending our background description on assertion-based verification and validation as it was first introduced in Chapter 2.4 in order to provide the necessary supporting methodologies for our proposed approach, which is then presented in Chapter 6.3 and Chapter 6.3.1. We finalize this chapter section by presenting in Chapter 6.4 the results of an experimental procedure we conducted in order to show the effectiveness of our approach for the purposes we have just described.

6.2 Specification Modeling with DBC and JML

As it was first described in Chapter 2.4, DBC has been shown to be an effective methodology for precisely describing the runtime *behavior* of a given software module, e.g., what it is supposed to do when executed. In such a context, JML implements the DBC paradigm by recurring to software assertions which better describe such a runtime behavior and also serve as a supporting methodology for verification and validation (Burdy *et al.* (2003)). In order to better support the development of modern software modules, specification-only types, e.g., classes or interfaces, were added to JML to provide behavioral specifications in a *higher* level of abstraction, e.g., without recurring to specific source-code level constructs. Such types, known as *model specifications* (Cheon *et al.* (2005)), allow for highly-customized, independent and self-contained modules to be constructed by precisely defining the way they must behave when executed without including any specific implementation details. In ad-

dition, these specification-only models can be *mapped* to actual low-level source code implementations in order to support the verification and validation of JML-specified modules.

As an example, Fig. 6.1 shows an excerpt of a JML-annotated Java interface called `FAMAttribute`, which is intended to provide a source-code representation of attributes in the context of our approach, as it was first introduced in Chapter 4.2.1. Lines 5-7 show three *model variables* intended to describe the main basic components required for an attribute: a *name*, a *type*, and a *value* in the range determined by *type*. Such model variables are later used in the rest of the JML constructs to specify the way an attribute should be constructed in a correct way. For instance, the invariants shown in Lines 10-12 specify that the aforementioned model variables must be instantiated to proper objects during the whole *lifetime* of an object implementing the `FAMAttribute` interface. This way, a correctly-constructed object may prevent unwanted situations in which an attribute is missing a basic required component, e.g., the name component being set to `null`, which could in turn deviate in unwanted security-risky situations at runtime, e.g., forcing the code implementing a policy evaluation routine to crash due to a `NullPointerException`, thus possibly affecting the results of an access mediation request. Moreover, as shown in Line 5, the *name* model variable is of type `FAMAttributeName`, which in turn can be specified using JML constructs to further defined how attribute names in the context of our approach should be handled, e.g., by requiring all names to be of a certain minimum or maximum length. In addition, JML allows for defining *model methods* that can be used to further specify both structural constraints as well as runtime behavior. As an example, Lines 15-21 introduce the `isFromProperType` model method that is later used in the DBC contract shown in Lines 23-28 to further restrict the nature of the value that will be included as a component of a given attribute. This way, such a value component must always

```
1 package edu.asu.sefcom.fam.model.attributes;
2
3 public interface FAMAttribute{
4
5  //@ public instance model FAMAttributeName name;
6  //@ public instance model Object value;
7  //@ public instance model Object type;
8
9
10  //@ public invariant this.name != null;
11  //@ public invariant this.value != null;
12  //@ public invariant this.type != null;
13
14
15  /*@ public model pure boolean
16    @   isFromProperType(Object newValue, Object type){
17    @
18    @   Class typeClass = type.getClass();
19    @   return typeClass.isInstance(newValue);
20    @ }
21    @*/
22
23  /*@ public normal_behavior
24    @  requires newValue != null &&
25    @           isFromProperType(newValue, this.type);
26    @  assignable this.value;
27    @  ensures this.value.equals(newValue);
28    @*/
29  public void setValue(Object newValue);
30
31  ...
32 }
```

Figure 6.1: An Excerpt of a Java Interface with Model Specifications.

be of the data type specified by the model variable *type* referred in Line 7, thus also preventing a situation when those two components differ, which can cause some unwanted runtime behavior as a consequence.

As mentioned before, model specifications can be mapped to actual implementation-specific constructs for the purposed of verification and validation. As an example, Fig. 6.2 shows class `FAMBasicAttribute`, which implements the `FAMAttribute` interface. Such a class provides so-called *concrete* implementations for the model constructs discussed before as follows: for the *value* and *type* model variables, a straightforward implementation is provided by creating instance variables of the same Java type, e.g., `java.lang.Object` as shown in Lines 10-11. Such a strategy is convenient when the Java type defined for the model variables matches the one desired for the concrete implementation code. The actual *mapping* between concrete and model variables is done by means of the JML `represents` construct, as shown in Lines 13-15. This way, every time a model variable is reference in a JML specification (as in Lines 23-28 of Fig. 6.1) JML-based tools such as the ones introduced in Chapter 2.4 may replace it with the corresponding concrete variable for verification purposes.

A more interesting case is presented for the *name* model variable. This time, the concrete implementation depicts a Java type (`java.lang.String`) that is different from the one intended for the model variable (`FAMAttributeName`). In such a scenario, a model method can be introduced in such a way the concrete Java type can be used to produce a proper representation of the type depicted by the model variable. As shown in Fig. 6.2 (Lines 17-21), the model method `getAttributeName` takes the concrete variable *_name* as an input in order to produce an object of class `FAMAttributeName` which is then mapped to the *name* model variable by means of the aforementioned `represents` construct in Line 10. This way, the `getAttributeName` allows for a concrete variable (*_name*) to be mapped to a model variable (*name*) that is said to

be defined in a higher level of abstraction. Therefore, in the rest of this dissertation, we refer to such model methods as *abstraction functions* (Cheon *et al.* (2005)).

6.3 Assertion-based Security Models

As it was hinted in Fig. 6.1 and Fig. 6.2, we propose an approach that combines the concepts of specification modeling and software assertions for describing security features at the source-code level. These so-called *assertion-based security models* are intended to provide compact, well-defined and consistent descriptions that may serve as a common reference for implementing security-related functionality. Our approach strives to fill in the gap between high-level descriptions of security features, which are mostly abstract and implementation-agnostic, and supporting descriptions focused at the source-code level, which are intended to cope with both security-related and behavioral-based specifications, such as the ones described in Section 2. As it will be described in Chapter 8, previous work has also explored the use of software assertions and DBC-like contracts for specifying security features such as access control policies. However, our approach is intended to leverage the *modeling* capabilities offered by software specification languages, such as the ones described in the previous Chapter 6.2, using a well-defined reference description of a security model as a source, in such a way that it not only allows for the correct communication, enforcement and verification of security-related functionality, but it also becomes independent of any supporting constructs, e.g., *application programming interfaces* (APIs) and *software development kits* (SDKs), thus potentially allowing for its deployment over applications composed of high-customized *heterogeneous* modules.

Fig. 6.3 presents a graphical depiction of a framework for software construction: initially, we aim to develop our a security model tailored for our FAM approach, which includes a description of its main components, their syntactic interface, their runtime

```
1 package edu.asu.sefcom.fam.model.mapping;
2
3 import edu.asu.sefcom.fam.model.attributes.FAMAttribute;
4
5 //@ model import edu.asu.sefcom.fam.model.attributes.FAMAttributeName;
6
7 public class FAMBasicAttribute implements FAMAttribute{
8
9  protected /*@ spec_public @*/ String _name;  /*@ in name;  @*/
10  protected /*@ spec_public @*/ Object _value; /*@ in value; @*/
11  protected /*@ spec_public @*/ Object _type;  /*@ in type;  @*/
12
13  //@ public represents this.name  <- getAttributeName(_name);
14  //@ public represents this.value <-_value;
15  //@ public represents this.type  <-_type;
16
17  /*@ public pure model FAMAttributeName
18    @                   getAttributeName( non_null String strName){
19    @  return new FAMBasicAttributeName(strName);
20    @ }
21    @*/
22
23  public FAMBasicAttribute( /*@ non_null @*/ String name,
24                            /*@ non_null @*/ Object value,
25                            /*@ non_null @*/ Object type){
26   this._name = name;
27   this._value = value;
28   this._type = type;
29 }
30
31  public void setValue(Object newValue){
32   this._value = newValue;
33  }
34 ...
35 }
```

Figure 6.2: An Excerpt of a Java Class Implementing Model Specifications.

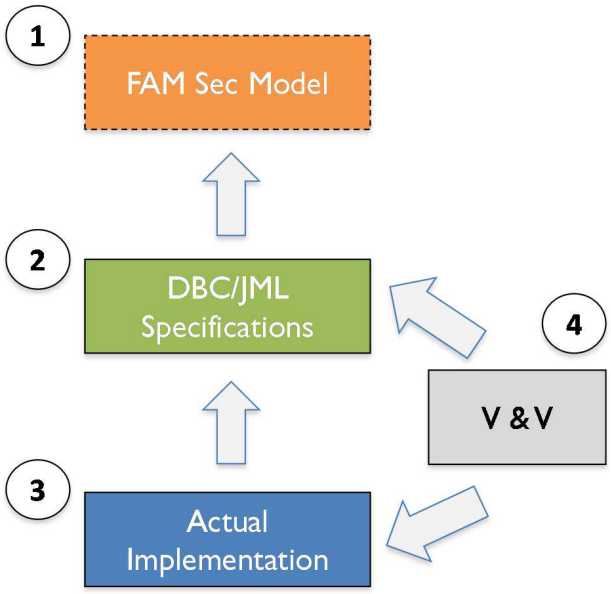

Figure 6.3: A Methodology for Assertion-based Construction.

behavior, as well as the relationships between each other, in a similar approach to the description of the `FAMAttribute` interface depicted in Fig. 6.1, which is intended to describe the way attributes following our approach should be implemented at the source-code level (Fig. 6.3 (1)). A detailed description of our proposed security model, to be referred as *FAM security model*, is shown in Chapter 6.3.1. Next, our FAM security model is to be later leveraged for participant organizations in order to develop their own DBC/JML contracts that reference the specification-only classes such a model includes as well (Fig. 6.3 (2)). In a subsequent step (Fig. 6.3 (3)), participant organizations leverage the DBC/JML specifications described before when constructing their own customized implementations of our approach, e.g., by providing concrete implementations of the specification-only interfaces included in it, as it is shown in Fig. 6.2. Finally, both the DBC/JML specifications as well as the implementation source code are to be fed to dedicated tools such as the ones described in Chapter 2.4 for verification and validation purposes (Fig. 6.3 (4)).

In the rest of this chapter, we will provide evidence on the effectiveness of our proposed framework for software construction. First, we will provide details on our proposed FAM security model in Chapter 6.3.1, including the way security functionality relevant to our overall approach is modeled using DCB/JML features. Then, we will describe how to leverage such a model in order to construct a custom-made implementation by using DBC/JML specifications on top of it and by providing source-code-level constructs that follow such specifications. Later, in Chapter 6.4, we will presents our results on an experimental procedure tailored to measure the effectiveness of our approach for detecting non-trivial security vulnerabilities.

6.3.1 An Assertion-based Security Model for FAM

In order to better describe the set of DBC/JML specifications as well as the Java modules that have been developed as a part of our proposed FAM security model, we now present a set of specification categories that help us provide a systematic approach for describing security-related functionality. For each category we describe its main purpose, its relevance with respect to detailing security issues, as well as the set of DBC/JML constructs that are supported. In addition, we exemplify the use of each category by presenting specification excerpts from our actual FAM security model code. Finally, Table 6.1 presents a summary of some of the components depicting our approach we are discussed in this chapter and the specification categories we used for describing their intended functionality at runtime.

Structural Specifications. We start our discussion by presenting a very basic category that is composed of specifications intended to provide descriptions on the structural composition of modules implementing security-related functionality. Such an *internal* structure may be composed of instance variables, data structures, as well

as the instance methods that may ultimately manipulate those (commonly known as *setters/getters*). This way, this category strives to provide specifications that can constraint either the set of values, the dimensions, or the features such internal structures must observe in order for the module being specified to remain in a *consistent* state throughout its lifetime.

Structural specifications can be expressed in DBC/JML by means of class invariants, model methods, as well as method contracts that make use of model variables. For security-related purposes, modules that stay in an inconsistent state may introduce unintended security vulnerabilities as a result, either in an internal way, or by forcing other modules to enter into vulnerable states because of them. As an example, Fig. 6.1 presents some structural specifications for our proposed `FAMAttribute` module, which, has mentioned before, is intended to model the functionality desired for attributes within our approach. Lines 5-8 present a set of model variables that are constrained by means of the invariants shown in Lines 10-12, e.g., all of them are forbidden from being `null` at anytime during the existence of an object implementing interface `FAMAttribute`. In addition, Lines 23-28 present a method contract that besides reinstating the aforementioned constraint on the *value* model variable, also introduces an additional one as detailed by the `isFromProperType` model method. This way, the three inner components of an attribute, as described in Chapter 4.2.1, should be implemented in such a way that they always stay within the desired bounds as expressed by the aforementioned specifications, thus possibly preventing unwanted situations in which they may pose a security risk. As an example, setting the model variable *value* to `null` may force another component, e.g., `FAMPolicyEvaluator` to crash due to an unexpected `NullPointerException`, which could potentially affect the way a given policy for inter-organizational resource mediation is ultimately evaluated and enforced.

Compositional Specifications. Our next specification category is intended to describe the way different components are to interact with each other, either by means of nesting components inside one another, or by requiring mutual pointer references between them at runtime. Such interactions can be described in DBC/JML by means of class invariants as well as contracts for methods that ultimately access those components. For security purposes, components may be composed into subsystems whose functionality may be affected if they are not able to interact with each other in a proper way.

As an example, Fig. 6.4 shows an excerpt of a Java interface `FAMPolicyRepository`, which is intended to provide an abstract implementation independent component that can handle the storage of both local and federated policies as it is depicted by the *policy administration layer* featured in Chapter 5.1.2. Policies are modeled by means of the `FAMPolicy` interface, which is composed by a set of inner components of type `FAMResource`: an abstract representation of a shared resource within the context of our approach. In such a setting, a given policy is stored within the repository by using each of its composing resources as an index key. This way, such a policy will be retrieved when a future access request on any of its composing resources must be mediated. Fig. 6.4 (Lines 7-9) provides an abstract representation of the policy repository by means of a model variable called *theRepository*, which leverages the `JMLEqualsToEqualsRelation` specification-only type provided by the JML framework. Following the discussion on model constructs presented in Chapter 6.2, JML provides a rich set of model types that are intended to be used in specifications only, in order to provide behavioral descriptions with higher levels of abstraction. Concretely, the `JMLEqualsToEqualsRelation` provides a mathematical relation between two Java types. This way, the *theRepository* model variable is to provide an entry relating a given resource of type `FAMResource` with a corresponding mediating policy

```
1 package edu.asu.sefcom.fam.model.policies;
2
3 //@ model import org.jmlspecs.models.JMLEqualsToEqualsRelation;
4
5 public interface FAMPolicyRepository{
6
7  //@ public instance model JMLEqualsToEqualsRelation theRepository;
8
9  //@ public invariant theRepository != null;
10
11  /*@ public normal_behavior
12    @  assignable theRepository;
13    @  ensures (\forall FAMResource resource;
14    @                   policy.getResources().contains(resource);
15    @                   theRepository.has(resource, policy));
16    @*/
17  public void addPolicy(/*@ non_null @*/ FAMPolicy policy);
18
19 ...
20 }
```

Figure 6.4: An Excerpt of a Java Interface Describing a Policy Repository.

of type `FAMPolicy`, as it is shown in the contract depicted in Lines 11-16. Using this abstract specification, implementers of a policy repository may choose different ways to provide actual storage capabilities for policies, assuming the relation between resources and policies is correctly implemented as it is shown in Fig. 6.4.

Model Data Structures and Methods. As it was first introduced in the previous section, we leveraged several specification-only types provided by the DCB/JML framework in order to provide an abstract representation of important data structures within the different components included in our FAM security model. As it was also mentioned before, such model data structures allow to specify important security-relevant constraints on top of them, which should be then preserved by any further

implementation that provides its own *concrete* data structures based on particular needs. In addition, we introduced a series of model methods intended to provide advanced descriptions of intended security-related functionality. We have developed such an approach based on the assumption that advanced descriptions, while still representable by means of other JML constructs, e.g., invariants and contracts, may have a simplified representation using model methods, as they are mostly based in Java source code, which can potentially be easier for future implementers to understand and follow when crafting their own source code.

With respect to security purposes, both model variables and methods provide a convenient way to specify non-trivial constraints and functionality that must be preserved in further implementations. Fig. 6.5 presents an specification for the `FAMPolicy` component representing both local and federated policies in our security model, which was also introduced before in this dissertation. Based on the background topics discussed in Chapter 2.3 as well as on the discussion on policies presented in Chapter 4.2, Chapter 5.1, and Chapter 5.2, we have chosen to model policies as a combination of sets containing protected resources, policy rules as well as the attribute names that are listed in those rules. This way, the attributes names required for evaluating a given policy can be extracted by the attribute provisioning scheme detailed in Chapter 5.2 before the actual evaluation of a policy can take place. Fig. 6.5 shows an specification of such sets in Lines 5-7, which leverage the abstract JML type `JMLEqualsSet`. As mentioned before, implementers can provide their own custom-made way to provide a concrete data structure representing those sets. In addition, Lines 11-28 provide a specification on the way attribute names (represented by type `FAMAttributeName`) are to be obtained from the rules contained in the policy (`FAMPolicyRule`) and concentrated into a dedicated set. As shown in the sample specification code, implementers should inspect each policy rule, extract

its inner attribute names, and concentrate all into the *attributeNames* set, which is then used by the method contract shown in Lines 30-36 to describe the contents being returned to a runtime called that invokes the `getAttributeNames` method.

Client-based Invariants/Contracts. As an special case of our proposed DBC/JML specifications, we have also identified the need to provide advanced invariants and method contracts that can precisely describe the way a given component within our FAM security model is to be leveraged by other *external* components, a.k.a., *clients*, by means of invoking security-sensitive public methods. Commonly, such clients will delegate security functionality to the modules implementing our FAM security model. Therefore, a well-defined and precise description detailing any initial assumptions, parameter passing, as well as a description of any return data is extremely desired.

With respect to security purposes, recent pitfalls in the proper specification of security-implementing APIs have been identified as the main source of serious vulnerabilities in major security software (Georgiev *et al.* (2012)). With this in mind, we aim to provide a dedicated categorization that highlights the functionality our FAM security model provides to external *client* components, in such a way that all interactions between the involved parties can be carried on safely as intended.

Fig. 6.6 presents a set of DBC/JML specifications tailored to describe the functionality of the `FAMProvisioningRuleDiscoveryEntryPath` interface, which is intended to represent the attribute discovery paths as described in Chapter 5.1 and Chapter 5.2. In such a context, entries depicting each of the AD-Rules to be included in the path are to be included in a model data structure known as *thePath* (Line 5). In addition, Lines 18-22 depict a very important invariant with respect to attribute discovery paths that must be preserved during the whole lifetime of an object implementing interface `FAMProvisioningRuleDiscoveryEntryPath`: the entries located in

```
1 package edu.asu.sefcom.fam.model.policies;
2
3 public interface FAMPolicy{
4
5  //@ public model instance JMLEqualsSet rules;
6  //@ public model instance JMLEqualsSet resources;
7  //@ public model instance JMLEqualsSet attributeNames;
8
9  //@ public represents this.attributeNames <- getFromRules(this.rules);
10
11  /*@ public pure model JMLEqualsSet getFromRules(JMLEqualsSet rules){
12    @   JMLEqualsSet result = new JMLEqualsSet();
13    @   JMLIterator iter = rules.iterator();
14    @
15    @   while(iter.hasNext()){
16    @      FAMPolicyRule rule = (FAMPolicyRule) iter.next();
17    @      List attributeNames = rule.getAttributeNames();
18    @      for(int j = 0; j < attributeNames.size(); j++){
19    @         FAMAttributeName attributeName =
20    @                  (FAMAttributeName) attributeNames.get(j);
21    @         if(!result.has(attributeName)){
22    @                    result.insert(attributeName);
23    @         }
24    @      }
25    @   }
26    @   return result;
27    @ }
28    @*/
29
30  /*@ public normal_behavior
31    @  ensures \result != null && \result.size() > 0 &&
32    @           (\forall FAMAttributeName attributeName;
33    @                    this.attributeNames.has(attributeName);
34    @                    \result.contains(attributeName));
35    @*/
36  public /*@ pure @*/ List getAttributeNames();
37 }
```

Figure 6.5: An Excerpt of a Java Interface Describing a FAM Policy.

thePath must be *chained* one after the other so that the attribute path can be further *traversed* for attribute provisioning purposes, as depicted in Chapter 5.2.2. Such a property is indeed crucial to ensure paths remain in a consistent state and can be effectively used for their intended purpose by other client modules. As shown in Lines 18-22, such a property is then preserved by required all entries being consecutive to each other, e.g., AD-Rule entries A and B such that A appears immediately before B in *thePath*, to satisfy a constraint requiring their the intersection of the sets depicted for both A output attributes as well as B input attributes to be non-empty, e.g., the size of the resulting set should be greater than zero. A model method specifying how such intersection can be calculated is shown in Lines 7-16. Finally, the aforementioned constraint is also enforced in the method contract shown in Lines 24-32 which requires clients invoking the `addRuleDiscoveryEntry` method to satisfy such a property before a new AD-Rule entry can be added to the attribute path.

Specifications for Method Customization. Finally, we present the last category of DBC/JML specification included in our proposed FAM security model. As it is intended in major languages following the *object-oriented programming* (OOP) paradigm (Rentsch (1982)), the functionality intended for software modules such as classes and interfaces is expected to be updated by means of providing customized version of existing methods, either by implementing a given interface or by using the concepts of *subtyping* and *method overriding*. This way, a newly introduced module may update the functionality of an already-existing one by implementing a dedicated interface or by first *subclassing* it and then overriding any method of interest.

While highly-convenient, such an approach may introduce non-trivial security vulnerabilities, as existing functionality devised in the original component may be broken in the newly-developed one, which may complicate the way such new component is to

```
1 package edu.asu.sefcom.fam.model.provisioning;
2
3 public interface FAMProvisioningRuleDiscoveryEntryPath{
4
5  //@ public instance model List<FAMProvisioningRuleDiscoveryEntry> thePath;
6
7  /*@ public pure model List intersection(List input, List output){
8    @  List result = new ArrayList();
9    @  for(int i = 0; i < input.size(); i++){
10   @   if(output.contains(input.get(i))){
11   @    result.add(input.get(i));
12   @   }
13   @  }
14   @  return result;
15   @ }
16   @*/
17
18 /*@ public invariant
19   @  (\forall int i; 0 <= i && i < thePath.size() - 1;
20   @    intersection(thePath.get(i+1).getInputAttributes(),
21   @                 thePath.get(i).getOutputAttributes()).size() > 0);
22   @ */
23
24 /*@ public normal_behavior
25   @   requires this.intersection(
26   @            entry.getInputAttributes(),
27   @            this.getRuleDiscoveryEntry(this.size())
28   @            .getOutputAttributes())
29   @            .size() > 0;
30   @   assignable thePath;
31   @   ensures thePath.get(thePath.size()-1).equals(entry);
32   @*/
33 public void addRuleDiscoveryEntry(/*@ non_null @*/
34             FAMProvisioningRuleDiscoveryEntry entry);
35 ...
36 }
```

Figure 6.6: An Excerpt of a Java Interface for an Attribute Provisioning Path.

interact with others in the context of implementing security-related functionality for a given sub-system. In order to prevent this situation, we have resorted to leveraging DBC/JML contracts for methods that are intended for further customization. This way, the basic required functionality for a given module is stated using an original DBC/JML contract. Later, a customized new version of such a module may also introduce new specification *cases* by means of the `also` construct. Following the semantics defined for DBC/JML contracts, any implemented/overridden method will be expected to fulfill not only its new set of DBC/JML specifications, but also the ones defined for the original one as well, thus preserving existing related functionality contained in the original software as a result.

As an example, Fig. 6.7 presents a Java interface called `FAMProvisioningRule` that is intended to represent our proposed AD-Rules discussed in Chapter 4.2.3. The actual functionality for transforming an attribute into another is to be implemented by means of the `process` method, which can be then customized by modules implementing such an interface or by modules subclassing an existing implementation of it. The contract for the `process` method shown in Fig. 6.7 can be described as follows: preconditions (Lines 6-9) require that the attribute derivation request, represented by an object implementing the `FAMProvisioningRequest` interface, contains a set of attributes whose names contain the ones intended to serve as an input to the transformation being performed by the current AD-Rule. In case such a precondition is fulfilled, the method's postconditions (Lines 11-22) state that the responding message, represented by an object of interface `FAMProvisioningResponse`, will contain either the resulting attributes as intended for the AD-Rule, in case the transformation was carried on successfully, or a single default *empty* attribute (`FAMEmptyAttribute`) otherwise, for the case the transformation was not performed due to some unexpected circumstances, e.g., runtime errors.

Fig. 6.8 shows a Java class `FAMIdentityProvisioningRule` that implements the aforementioned `FAMProvisioningRule` interface to provide a customized transformation that simply returns the same set of attributes provided as an input, thus depicting an *identity* function. In such a case, the source code intended for the `process` method (Lines 9-14) must follow the specifications defined for it in the `FAMProvisioningRule` interface as described before, as well as the ones that are related to its expected customized behavior, which are shown in Lines 5-7: the resulting response message will include all the attributes including in the original request and nothing else. In Chapter 6.4, we present our experiments intended to verify that actual implementations as the one provided by Fig. 6.8 meet their corresponding DBC/JML specifications as defined by our proposed FAM security model.

6.4 Experimental Evaluation

In this chapter, we describe an experimental process tailored to provide evidence of the suitability of our proposed FAM security model to allow for the construction of custom-made enforcement mechanisms following the approach described in this dissertation. In addition, we aim to show how our proposal can serve as a media for conducting a conformance procedure for such customized implementations, in such a way that the desired security features, as expressed by means of the DBC/JML constructs described in the previous chapter, are indeed preserved at the source-code level. Finally, we present an experimental procedure tailored to show the suitability of our proposal for detecting non-trivial security vulnerabilities that may be introduced as a part of the development process and can be potentially located by using our FAM security model as a testing oracle. We provide a description of the methodology used as well as some experimental results that support our claims.

```
1 package edu.asu.sefcom.fam.model.provisioning;
2
3 public interface FAMProvisioningRule{
4
5  /*@ public normal_behavior
6    @  requires (\forall FAMAttributeName attributeName;
7    @                    this.getRuleDiscoveryEntry().
8    @                    getInputAttributes().contains(attributeName);
9    @                    request.getAttributesNames().contains(attributeName));
10   @
11   @  ensures \result != null &&
12   @          \result.getPeerInfo().equals(request.getPeerInfo()) &&
13   @          \result.getAttributes().size() > 0 &&
14   @          (\result.getAttributes().size() > 1 ==>
15   @          ((\forall FAMAttribute attribute;
16   @                   \result.getAttributes().contains(attribute);
17   @                         !(attribute instanceof FAMEmptyAttribute))) &&
18   @              (\forall FAMAttribute attribute;
19   @                   \result.getAttributes().contains(attribute);
20   @                   this.getRuleDiscoveryEntry().
21   @                   getOutputAttributes().contains(attribute.getName()))
22   @          );
23   @*/
24  public /*@ pure @*/ FAMProvisioningResponse
25                      process(/*@ non_null @*/ FAMProvisioningRequest request);
26 }
```

Figure 6.7: An Excerpt of a Java Interface for an Attribute Provisioning Rule.

```
1 package edu.asu.sefcom.fam.model.testing;
2
3 public class FAMIdentityProvisioningRule implements FAMProvisioningRule{
4
5  /*@ also
6    @  ensures \result.getAttributes().equals(request.getAttributes());
7    @*/
8  public FAMProvisioningResponse process(FAMProvisioningRequest request){
9    List<FAMAttribute> attributes = new ArrayList<FAMAttribute>();
10   attributes.addAll(request.getAttributes());
11
12   return new FAMBasicProvisioningResponse(this.getRuleDiscoveryEntry(),
13                                           request.getPeerInfo(),
14                                           attributes);
15 }
16 ...
17 }
```

Figure 6.8: An Excerpt of a Java Class for an Attribute Provisioning Rule.

Constructing an Implementation. We start our discussion by presenting our results when constructing a customized implementation of our FAM security model, following the assertion-based DBC/JML descriptions that were introduced in Chapter 6.3.1.

Initially, we provided Java classes implementing the interfaces contained in the FAM security model, which in turn provide a representation of each of the components of an enforcement mechanism as depicted in Chapter 5.

Next, we proceed to construct each component by following the structural specifications devised for each of them. As an example, Fig. 6.2 presents an implementation of the `FAMAttribute` interface presented in Fig. 6.1: for each model variable we provide a concrete implementation and also use DBC/JML constructs in such a way that a mapping between the two is established, e.g., using the `represents` and `in`

Table 6.1: A Set of Sample Components of our FAM Security Model.

	Structural	Compositional	Model Variables/Methods	Invariants/Contracts for Clients	Customization
`FAMAttribute`	√		√		√
`FAMPolicy`	√	√	√		
`FAMPolicyRepository`		√	√	√	√
`FAMPolicyEvaluator`	√		√		√
`FAMPolicyDecisionCombinator`					√
`FAMProvisioningDiscoveryEntryPath`	√	√		√	
`FAMProvisioningDiscoveryEntryRepository`		√	√	√	
`FAMProvisioningRule`	√	√		√	√
`FAMProvisioningAgent`	√	√	√	√	

constructs (Lines 9-15). Later in this chapter, we show how such mapping allows for a verification and validation process for conformance and vulnerability detection to be carried on. In addition, we provide proper class constructors and getter/setter methods that follow the class invariants defined over the aforementioned model variables, in such a way that the structural *consistency* of the implemented class can be preserved.

In a subsequent step, we combined different components with each other following

our proposed compositional specifications, in an effort to implement functionality that is intended for security-subsystems within our enforcement mechanism. As an example, Fig. 6.4 provides a representation of a software module that is intended to store policies (represented by the `FAMPolicy` shown in Fig. 6.5), so they can be later located for further evaluation, implementing the functionality devised in Fig. 6.4: the repository itself is modeled by means of a dedicated data structure implementing a *hashtable* (`java.util.Hashtable`, Lines 5-7). A policy is added to the repository by the implementation of the `addPolicy` method (Lines 27-36), which retrieves the list of protected resources (`FAMResource`) listed in the policy and inserts and entry into the hashtable linking both the policy and each resource for future retrieval.

Continuing with the example shown in Fig. 6.9, we also provided a concrete implementation for the model data structure *theRepository* listed in Fig. 6.4 (Line 7) by means of the aforementioned hashtable labeled as *repositoryTable*. Mapping between those two variables (model and concrete) is actually implementing by means of the `convert` model method, shown in Lines 11-25 of Fig. 6.9: each entry in the hashtable is extracted and used to populate entries into the specification-only type `JMLEqualsToEqualsRelation`. This way, any reference within our DBC/JML specifications to the *theRepository* model variable with be linked to an object constructed from parsing the concrete implementation provided by the *theRepository* hashtable by means of the aforementioned `convert` model method, thus implementing an abstraction function as described before in this chapter. Such a feature will be extremely useful for the conformance and vulnerability-detection testing to be discussed later in this chapter.

Our next step included following the specifications contained in class invariants and method contracts in order to provide implementations of security-related functionality for software clients. As an example, Fig. 6.10 depicts the implementation

```
1 package edu.asu.sefcom.fam.model.mapping;
2
3 public class FAMBasicPolicyRepository implements FAMPolicyRepository{
4
5  private /*@ spec_public @*/
6       Hashtable<FAMResource, List<FAMPolicy>> repositoryTable;
7                                                 /*@ in theRepository; @*/
8
9  //@ public represents this.theRepository <- convert(repositoryTable);
10
11 /*@ public model pure JMLEqualsToEqualsRelation convert(Hashtable table){
12   @  JMLEqualsToEqualsRelation result = new JMLEqualsToEqualsRelation();
13   @  Enumeration tableKeys = table.keys();
14   @  while(tableKeys.hasMoreElements()){
15   @    FAMResource resourceKey = (FAMResource) tableKeys.nextElement();
16   @    List policies = getPolicyList(resourceKey);
17   @
18   @    for(int i = 0; i < policies.size(); i++){
19   @     FAMPolicy policy = (FAMPolicy) policies.get(i);
20   @     result = result.add(resourceKey, policy);
21   @    }
22   @  }
23   @  return result;
24   @ }
25   @*/
26
27 public void addPolicy(/*@ non_null @*/ FAMPolicy policy){
28   List<FAMResource> policyResources = policy.getResources();
29
30   for(int i = 0; i < policyResources.size(); i++){
31    FAMResource resource = policyResources.get(i);
32    List<FAMPolicy> policyList = getPolicyList(resource);
33    policyList.add(policy);
34    this.repositoryTable.put(resource, policyList);
35  }
36 }
37 ...
38 }
```

Figure 6.9: An Excerpt of a Java Class for a Policy Repository.

of a Java class `FAMBasicProvisioningAgent`, with provides a representation for the APA modules discussed in Chapter 5.1.2. In such a class, the functionality devised for the *path discovery* process, as described in Chapter 5.2.1, is implemented by method `getPaths` (Lines 5-36) by leveraging the well-known *breadth-first search* (BFS) algorithm (Lee (1961)). Our implementation initially places the names of the *goal* attributes contained in the original request into the *queue* data structure and then tries to retrieve entries *producing* such attributes from the local entry repository (Line 7-10). The process of locating entries producing an attribute is called attribute *expansion*. Every single time a new entry is located, they are added into a resulting path and its input attributes are added to the *queue*. Following the BFS algorithm, the process is repeated until no attributes are left in the *queue*. This way, paths are constructed in a *backwards* fashion until no more attributes are left to be expanded.

In such an implementation code, we leverage the invariant and method contract specifications describing how a given path should be constructed, e.g., by creating a consistent object of class `FAMProvisioningRuleDiscoveryEntryPath` as shown in Fig. 6.6. Recall from Chapter 5.2.1 and Chapter 6.3.1 that an attribute-provisioning path must contain entries representing AD-Rules in such a way that for each pair of consecutive entries A and B must contain a non-empty set depicting the intersection of A's input attributes and B's output ones. During the discovery of possible paths in method `getPaths`, we preserve the aforementioned constraint by means of the auxiliary method `getPreviousEntries` (Line 19), which takes as an input the name of the attribute being *expanded* and retrieves any locally-stored entries that ultimately may produce such an attribute. Later, Lines 23-31 iterate over such entries and, for each entry, a new shall copy of the current path being explored is created and the entry is added in the beginning position of it. This way, the aforementioned constraint is preserved and every resulting path is said to be in a consistent state.

```
1 package edu.asu.sefcom.fam.model.mapping;
2
3 public class FAMBasicProvisioningAgent implements FAMProvisioningAgent{
4
5 public List getPaths(FAMProvisioningRuleDiscoveryRequest request,
6                      List<FAMAttributeName> attributeNames){
7  List result = new ArrayList();
8  Queue queue = new LinkedList();
9  HashMap pathsMap = new HashMap();
10 queue.addAll(this.getPreviousEntries(request.getRequestedAttributeName()));
11
12 try{
13  while(!queue.isEmpty()){
14   FAMProvisioningRuleDiscoveryEntry entry = queue.remove();
15   FAMProvisioningRuleDiscoveryEntryPath path = this.getStoredPath(entry, pathsMap);
16
17   for(int i = 0; i < entry.getInputAttributes().size(); i++){
18    FAMAttributeName attributeName = entry.getInputAttributes().get(i);
19    List producingEntries = this.getPreviousEntries(attributeName);
20
21    for (int j = 0; j < producingEntries.size(); j++){
22     FAMProvisioningRuleDiscoveryEntry producingEntry = producingEntries.get(j);
23     FAMProvisioningRuleDiscoveryEntryPath newPath = path.clonePath();
24
25     if(!newPath.hasRuleDiscoveryEntry(producingEntry)){
26      newPath.prependRuleDiscoveryEntry(producingEntry);
27      pathsMap.put(producingEntry, newPath);
28      attributeNames.addAll(producingEntry.getOutputAttributes());
29
30      if(!attributeNames.containsAll(producingEntry.getInputAttributes())){
31       if(!queue.contains(producingEntry)){queue.add(producingEntry);}}
32      ...
33  }}}}
34 }catch(Exception e){...}
35 return result;
36 }
37 ...
38 }
```

Figure 6.10: An Excerpt of a Java Class for an Attribute Provisioning Agent.

```
1 package edu.asu.sefcom.fam.model.mapping;
2
3 public abstract class FAMBasicProvisioningRule implements FAMProvisioningRule{
4
5  public FAMProvisioningResponse process(FAMProvisioningRequest request){
6   List attributes = new ArrayList();
7   attributes.add(new FAMEmptyAttribute());
8
9   return new FAMBasicProvisioningResponse(this.getRuleDiscoveryEntry(),
10                                          FAMProvisioningAgent.localPeerInfo,
11                                          attributes);
12  }
13 }
```

Figure 6.11: An Excerpt of a Java Class for an Attribute Provisioning Rule.

Finally, we leveraged the specification tailored for further method customization in order to produce the basic functionality as intended by the *basic* component and then leveraging customized versions of it that introduce addition runtime behavior while preserving the original one. Such a step followed an approach similar to the one already shown in Fig. 6.7 and Fig. 6.8: basic functionality for AD-Rules is implemented by a basic component in the form of an abstract Java class and further refinements are then introduced to produce customized behavior, which should preserve the original one at any time for the sake of consistency. As an example, basic functionality for the Java interface depicted in Fig. 6.7 is implemented by the `FAMBasicProvisioningRule` abstract class shown in Fig. 6.11: the implementation code for the `process` method simply returns an attribute list with an object of the `FAMEmptyAttribute` as a result. Later, such a behavior is customized by the implementation shown in Fig. 6.8.

Conformance Testing. As mentioned before in this chapter, we aim to provide experimental evidence that can show the suitability of our proposed FAM security

model to serve as a guidance for the correct construction of source-code implementations of our enforcement mechanism, which is shown in Chapter 5. With this in mind, we now describe an experimental procedure tailored to assess the conformance of our custom-made implementation, as described in Chapter 6.3, with respect to the FAM security model we have also described in Chapter 6.3.1.

Methodology. For such a purpose, we resorted to a methodology based on manually producing different execution traces based on the DBC/JML specifications contained in our FAM security model, in such a way that we can reproduce security-related functionality as intended at runtime, following the descriptions for inter-organizational resource sharing mediation as they are described both in Chapter 4 and Chapter 5. We developed different traces for each of the specification categories introduced in Chapter 6.3.1. As an example, traces intended for structural specifications included the instantiation of different objects from the classes providing concrete implementations of DBC/JML interfaces representing the major components included in our approach: attributes (`FAMAttribute`), policies (`FAMPolicy`), AD-Rules (`FAMProvisioningRule`) and so on. In addition, we also introduced traces of method invocation calls for the methods commonly known as getters/setters. In a similar fashion, execution traces for compositional specifications included the creation of different objects belonging to dedicated sub-systems, e.g., the policy storage and retrieval functionality depicted in Fig. 6.4 and Fig. 6.5, as well as the introduction of method calls exercising the functionality intended for them, e.g., adding, retrieving and removing a variant number of policies stored in a given repository.

In addition, we tested our proposed specifications that include model data structures and methods by introducing sequence traces that exercise the concrete data structures implementing such model constructs and verifying that the invariants de-

vised for them are always preserved. As in the sequences for the previous compositional category, we also tested the concrete data structures by adding or removing a varying number of elements and by explicitly attempting to add them in the *boundaries* defined for each specific data structure, e.g., the initial and the final position for data structures of type `java.util.List`. We also provided traces for our client-based method specifications by selecting the ones that implement the most security-sensitive functionality and the ones that ultimately have an effect on preserving important class invariants. As an example, we provided traces adding and removing AD-Rule entries (`FAMProvisioningRuleDiscoveryEntry`) into the attribute provisioning path depicted in Fig. 6.6. Finally, our conformance testing methodology for methods intended for further customization included creating traces for the original method and then adding sequences of method calls tailored to meet its original specifications. Later, we also provided method calls taking as an input objects and/or values intended to test the specifications that describe customized behavior, and checking they behave as expected at the same time the behavior of the original module is preserved as well.

Supporting Tools. Our testing traces as just described were tested by leveraging *runtime assertion checking* (RAC) code. As introduced in Chapter 2.4, dedicated tools can transform DBC/JML specifications into RAC code that can be then executed along with the one devised for concrete implementations. This way, any discrepancy between those two will be detected and a signal in the form of the runtime error raised. In our experiments, we leveraged the `JET` tool described in Chapter 2.4 to compile both our DBC/JML specifications included in our FAM security model as well as our implementation classes in order to produce Java bytecode enhanced with RAC constructs. As an example, Fig. 6.12 shows a simplified version of Java source

code that emulates RAC constructs for illustrative purposes. Such a code depicts a compilation of the concrete implementation of class `FAMBasicAttribute` as shown in Fig. 6.2, which in turn follows the specifications defined for the `FAMAttribute` interface depicted in Fig. 6.1. Compilation of the `FAMBasicAttribute` with the `JET` tool will produce RAC code as follows: taking as an example the `setValue` method, code constructs included in its original *body*, e.g., the ones included in Fig. 6.2 Line 32), are moved into a new private method called `setValue$original`, as shown in Lines 19-21 of Fig. 6.12. In addition, the preconditions devised for such a method, which are listed in Fig. 6.1 Lines 24-25, are translated into the `setValue$PRE` method. Each precondition assertion is then translated into an evaluation condition of an `if` statement, and several assertions are glued together by means or an `or` (||) operator. This way, any violation of an assertion at runtime will result in an error of type `JMLPreconditionError`, along with a description of what went wrong. In a similar fashion, the postconditions devised for the `setValue` method, shown in Line 27 of Fig. 6.1, are translated into the code contained within the `setValue$POST` method, which implements an approach analogous to the one described before for assertion checking by raising an error of type `JMLPostconditionError` in case a violation is detected. Finally, the body of the `setValue` method, as shown in Fig. 6.12 in Lines 8-10, is replaced with method invocation calls to the aforementioned new methods as produced by the RAC-based compilation: as expected, an invocation of the method checking preconditions is followed by the invocation of the original method contents, finalizing with the assertion-checking code for the corresponding postconditions.

Results. During the course of our experiments, we were able to detect several discrepancies between our provided implementation and their corresponding DBC/JML specifications as defined in our FAM security model, which resulted in

the RAC code raising errors as the ones described above. Some of such discrepancies were due to an initial incomplete understanding of the actual specifications intended for our approach, or simply because of common pitfalls in coding, e.g., array indexes going out of bounds. In order to fix such a problem, we resorted to a careful analysis of the model specifications and modified the implementation code accordingly, thus resulting in an *implementation refinement* process. As an example, the implementation of method `addPolicy` shown in Fig. 6.9 (Lines 27-36) exhibited two initial pitfalls: first, the iteration range defined for the `for` loop defined in Line 30 was initially incorrect, as the stopping condition was set to `i < policyResources.size() - 1`. Second, the method call listed in Line 33, which effectively inserts the policy into the list of policies defined for a given resource, was initially missing. After receiving error messages when executing such a code enhanced with RAC constructs, proper corrections were made as a consequence.

An additional set of discrepancies involved a careful revision of our proposed DBC/JML specifications, which ultimately resulted in the actual specification being updated to better reflect the intended runtime behavior. An example of such situation, which we call *specification refining*, included updating the model invariant depicted by interface `FAMProvisioningRuleDiscoveryEntryPath` in Fig. 6.6, Lines 18-22. An initial version depicted a stricter constraint requiring that, for every pair of consecutive AD-Rule entries A and B, the set of output attributes produced by A is a subset of the input attributes devised for B. After careful examination, such a constraint was regarded as too strong, deriving in the one that is depicted in Fig. 6.6.

We conducted an additional set of experiments to measure the impact on runtime execution time of the implementation code enhanced with RAC constructs. For such a purpose, we leveraged the execution traces discussed above and measure the time taken for the RAC-enhanced code to complete a trace run. For comparison

```
1 package edu.asu.sefcom.fam.model.attributes;
2
3 public abstract class FAMBasicAttribute extends FAMAttribute {
4
5  protected  Object _value;
6
7  public void setValue(Object newValue){
8   setValue$PRE(newValue);
9   setValue$original(newValue);
10  setValue$POST(newValue);
11 }
12
13 protected void setValue$PRE(Object newValue){
14   if(newValue == null || !isFromProperType(newValue, this.getType())){
15    throw new JMLPreconditionError("Preconditions failed!");
16   }
17 }
18
19 private void setValue$original(Object newValue){
20  this._value = newValue;
21 }
22
23 protected void setValue$POST(Object newValue){
24  if(!this._value.equals(newValue)){
25   throw new JMLPostconditionError("Postconditions failed!");
26  }
27 }
28 ...
29 }
```

Figure 6.12: An Excerpt of a Java Class Depicting RAC Code.

purposes, we also measure the execution time taken by our implementation code to run the same traces without any RAC code included. In addition, in the case of the compositional specifications described before, we also provided a varying number of composing elements to the traces being executed, in such a way we can observe the runtime performance of the modules under test by taking into account different configurations. In our experiments, we observed that RAC has an impact in runtime performance that is linear with respect to the number of assertions being checked, as shown in Fig. 6.12, as well as with respect to the number of composing elements being considered. As an example, Fig. 6.13 shows our experimental results for the `FAMBasicPolicyRepository` module as shown in Fig. 6.9, whose FAM security model specifications are shown in Fig. 6.4. In such a experiment, we varied the number of protected resources (`FAMResource`) as well as the number of policies (`FAMPolicy`) guarding access to them. As observed in Fig. 6.13, the performance detriment introduced by the execution of RAC code remains within manageable terms as soon as the number of resources and policies stays low. As such a number increases, the execution time gets increased as well, due to the increasing number of RAC code assertions that need to be checked within those inner components as well on each execution trace. A similar pattern was observed in an additional experiment depicting the `FAMProvisioningRuleDiscoveryEntryPath` module shown in Fig. 6.6. As with the previous example, we varied the number of composing elements by introducing different entries depicting AD-Rules. Once again, we observed a linear increase with respect to the number of inner components and a performance detriment caused by an increasing number of RAC-based assertions to check at runtime. Finally, an experiment showing a different trend is shown in Fig. 6.15, which was intended to depict the `FAMProvisioningRule` module as presented in Fig. 6.7 and Fig. 6.8. As discussed before, the sample AD-Rule implemented by Fig. 6.8 simply returns as a

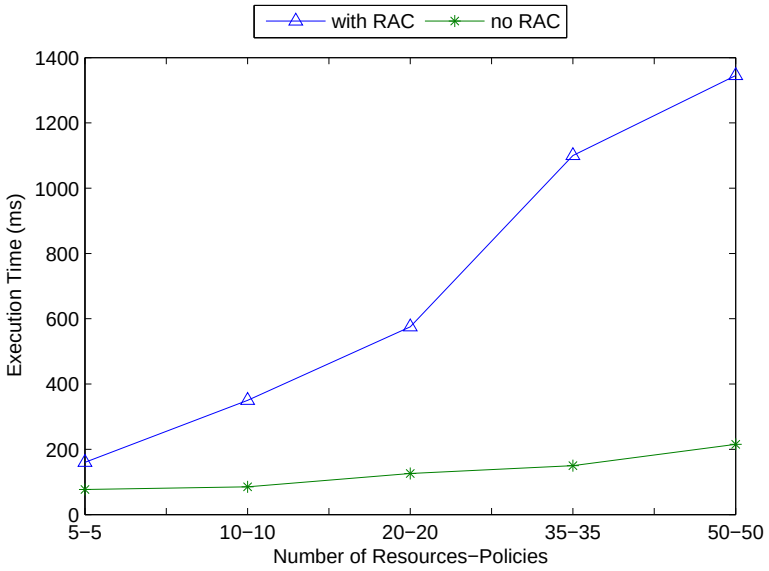

Figure 6.13: Experimental Results for a Policy Repository Module.

result all of the attributes it receives as an input. In our experiment, we kept varying the number of attributes provided to our implementation, and observed a linear performance impact with respect to the number of attributes as with our previously discussed experiments. However, the difference between the runtime execution times obtained for our non-RAC implementation as well as the RAC-enhanced one was not as wide as with our previous experiments (10-15 ms). This is due to the nature of the DBC/JML specifications listed in Fig. 6.7 and Fig. 6.8, which constraint the behavior of the AD-Rule being implemented but introduce no constraints on composing elements such as the varying number of attributes introduced on each experiment instance. As such attributes depict no RAC-based assertions to check, they introduce no noticeable performance detriment as a consequence.

Limitations. Even when our conformance testing approach was extremely successful to perform a *refinement* process for both the DBC/JML specifications as well

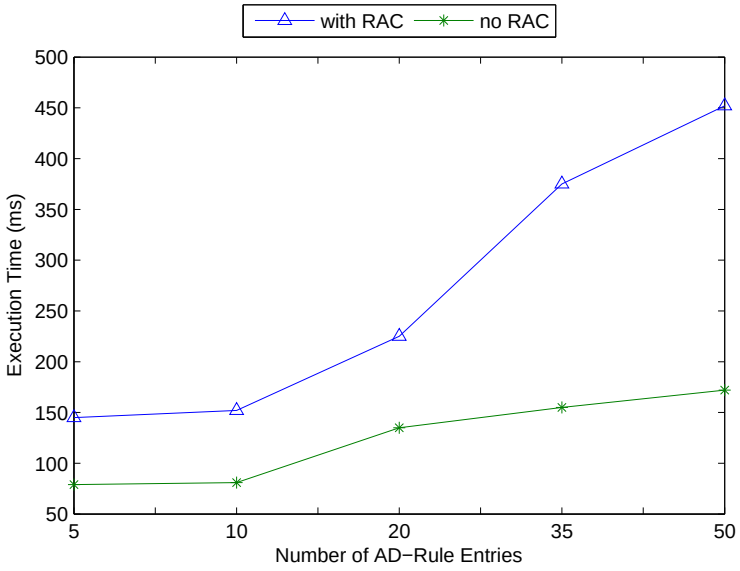

Figure 6.14: Experimental Results for an Attribute Provisioning Path.

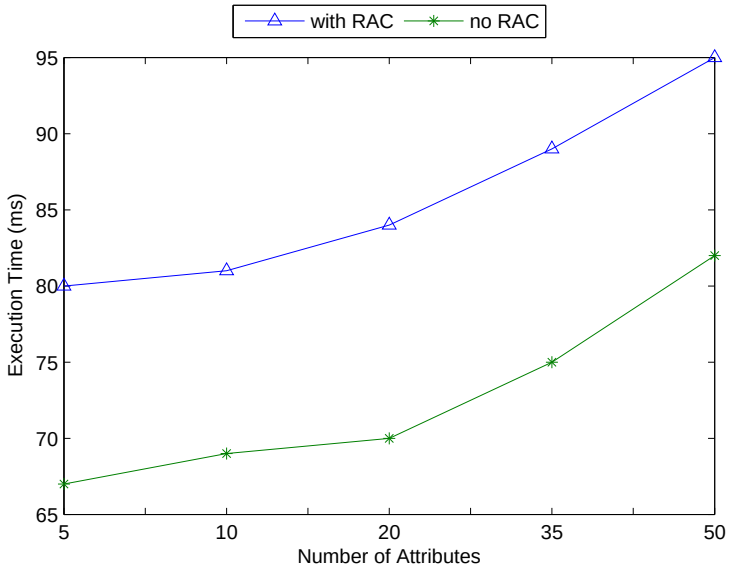

Figure 6.15: Experimental Results for an Attribute Provisioning Rule.

as our customized implementation of our FAM security model, still some limitations may be introduced by the fact our traces simulating runtime behavior may have not been able to provide a deep exploration of the multiple execution paths that may be in turn included in a given implementation, thus preventing our experimental process for discovering some discrepancies between specifications and implementing source code that may not be easily spot at plain sight. An approach to alleviate this problem may include considering static analysis techniques such as the ones presented by (Doupé *et al.* (2012)) to calculate different execution states in which advanced discrepancies specifications-implementation can be detected and corrected.

Vulnerability Testing. In addition to the conformance experiments just described, we also performed an experimental evaluation intended to measure the effectiveness of our approach for detecting security vulnerabilities within our customized implementation of our FAM security model as discussed in Chapter 6.3. This way, we aim to provide insight on the capabilities of our approach for finding mismatches between the DBC/JML specifications and source-code implementations based on them, in such a way that any potential discrepancies, which can be later categorized as vulnerabilities, can be property detected and corrected.

Methodology. In order to fulfill the purposes just described, we resorted to an approach primarily based on *mutation testing* (Jia and Harman (2011)): we manually inserted well-crafted modifications, a.k.a., *mutants*, into our implementation source code that may potentially represent security vulnerabilities. Later, we followed a similar approach as the one described before for our conformance testing process: we compiled the mutated source code with the `JET` tool to produce RAC-enhanced code, which was then executed by leveraging the aforementioned traces that simulate the

expected runtime behavior of the implemented enforcement mechanism. This way, if our RAC-enhanced code is able to detect a given mutant, e.g., by raising a runtime error as a consequence of an assertion-based checking code being violated, the mutant is then said to have been *killed*. With this in mind, we measured the effectiveness of our approach by the number of mutants that were actually killed by our RAC code with respect to the number of mutants that were crafted into the implementation in the first place.

In addition, we strove to introduce mutants in our implementation code as realistically as possible. Taking into account the classification of DBC/JML constructs introduced in Chapter 6.3.1, we produced mutated source code as follows: for the structural specifications category, we inserted mutations that were intended to deliberately violate the inner invariants and method contracts defining the *consistency* of a given module. As an example, in Fig. 6.2, we inserted a mutation in method `setValue` (Line 32) in such a way that the instance variable *_value* was set to `null`, thus deliberately violating the DBC/JML structural class invariant defined in Fig. 6.1 (Line 11). For our compositional specifications, we inserted mutants in the body of methods intended to preserve compositional properties. As an example, in Fig. 6.9, we commented Line 34, which belongs to the `addPolicy` method and is in charge of effectively adding a new entry into the *repositoryTable* structure, thus violating the compositional constraints stated in the DBC/JML contract depicted in Lines 11-16 of Fig. 6.4.

With respect to our model data structures and methods, mutants were designed in such a way that their corresponding concrete implementations were in conflict with them. As an example, in Fig. 6.9, we introduced mutants into the `convert` model method designed to provide a mapping between the model structure labeled as *theRepository* and its corresponding concrete implementation *repositoryTable*. For

```
1 package edu.asu.sefcom.fam.model.mapping;
2
3 public class FAMBasicProvisioningRuleDiscoveryEntryPath
4              implements FAMProvisioningRuleDiscoveryEntryPath, Cloneable{
5
6
7  private /*@ spec_public @*/ List _path; /*@ in thePath; @*/
8
9  //@ public represents thePath <- this._path;
10
11  public void addRuleDiscoveryEntry(FAMProvisioningRuleDiscoveryEntry entry){
12   this._path.add(entry);
13  }
14 ...
15 }
```

Figure 6.16: An Excerpt of a Java Class Depicting an Attribute Provisioning Path.

such a purpose, we modified Line 20 of Fig. 6.9 to prevent an entry to the `JMLEquals-ToEqualsRelation` model structure to be created, thus leaving it empty for specification purposes, not being able to meet the constraints defined in the DBC/JML contracts that refer to the *theRepository* model variable as a consequence. A similar approach was designed for the specifications tailored for describing runtime behavior intended for Clients: we introduced mutants into the bodies of method implementing functionality primarily intended for clients in such a way that there was a discrepancy between them and their corresponding DBC/JML contract, which was in turn violated by the implementation code. As an example, Fig. 6.16 shows an implementation of the `FAMProvisioningRuleDiscoveryEntryPath` shown in Fig. 6.6. In this case, we deliberately modified Line 12 to prevent the entry passes as a parameter from being inserted into the *_path* data structure, thus violating the DBC/JML contract devised for such a method.

Finally, mutants for source code intended for further customizations included mod-

ifying the implementations depicting such customized behavior in such a way that they potentially create a discrepancy with their corresponding DBC/JML specifications as devised in a base component, e.g., an interface or a *superclass*. As an example, we modified Line 10 of Fig. 6.8 so that no attributes are included in the response message of type `FAMProvisioningResponse`, thus violating the specifications for the `process` method as shown in Lines 11-23 of Fig. 6.7.

Supporting Tools. As mentioned before, we followed a similar procedure as with our previous conformance experiments: using our *mutated* implementation code as well as the DBC/JML specifications of our FAM security model as an input, we leveraged the `JET` tool to produce RAC-enhanced Java bytecode that is then used to execute simulated traces of expected behavior. On each experiment instance, we recorded the mutant being introduced, the components, e.g., classes being on each trace, as well as the result of executing each trace, either normal termination, e.g., the mutant went *undetected*, or abnormal termination because of an assertion-based DBC/JML error, which may indicate the mutant was effectively *killed*.

Results. As expected, our experimental process was able to successfully detect all the mutants introduced in our sample implementation which followed the methodology described in this chapter. As an example, Fig. 6.17 shows a runtime error being thrown as a result of the mutant being introduced in the source code shown in Fig. 6.16, which prevents the actual construction of the path by erasing the method call depicted in Line 12. While most of the mutants were detected right away as the corresponding RAC code implementing DBC/JML specifications found a discrepancy within the initial set of statements comprising our simulated traces, some other mutants take additional time to be effectively *killed*. As an example, we

designed a mutant for the implementation shown in Fig. 6.10, in such a way that the invocation of method `getPreviousEntries` in Line 19 was modified to retrieve the same entries all the time instead of the ones producing the attribute being analyzed at a time. While such modification was expecting to break the invariants for the `FAMProvisioningRuleDiscoveryEntryPath` defined in Fig. 6.6, Lines 18-22, the actual detection of the mutant was not self-evident in the first place, as our experimental running trace included constructing paths depicting the same entry of the AD-Rule shown in Fig. 6.8, which takes as an input and output sets the same set of attributes. As a consequence, our mutant went undetected as all paths constructed using the same AD-Rule will always preserve the invariant just mentioned.

Limitations. Even when our approach was successful in detecting the mutants we introduced to deliberately violate the specifications contained within our FAM security model, which may in turn represent potential vulnerabilities, still a more extensive mutant creation process may be needed to further evaluate the capabilities of our approach for detecting advanced situations what may represent hard-to-spot vulnerabilities. As an example, an advanced approach can be tailored to produce mutants that follow the expected control and data flow of a given customized implementation and try to break DBC/JML specification in a more interesting way. In addition, advanced mutation-based testing may also take into account the behavior of supporting *in-house* security systems, which, as described in Chapter 3 and Chapter 5, are intended to provide *backend* functionality for our proposed AD-Rules. This way, the actual functioning of a given implementation can be tested in a broader way by incorporating components not devised within the experimental process discussed in this chapter.

```
1 ------------------------------------------------------------------------
2 Testing... edu.asu.sefcom.fam.model.testing.FAMProvisioningEntryPathTester
3 ------------------------------------------------------------------------
4 Exception in thread "main"
5     org.jmlspecs.jmlrac.runtime.JMLInternalNormalPostconditionError:
6 by method FAMBasicProvisioningRuleDiscoveryEntryPath.addRuleDiscoveryEntry
7    regarding specifications at
8 File "FAMProvisioningRuleDiscoveryEntryPath.java", line 54, character 63 when
9 'entry' is (Rule:[A-0, A-1, A-2, A-3]->[A-0-0, A-0-1, A-0-2, A-0-3, A-0-4])
10 'this' is edu.asu.sefcom.fam.model.mapping
11    .FAMBasicProvisioningRuleDiscoveryEntryPath@...
12 at edu.asu.sefcom.fam.model.mapping.FAMBasicProvisioningRuleDiscoveryEntryPath
13 .addRuleDiscoveryEntry(FAMBasicProvisioningRuleDiscoveryEntryPath.java:590)
14 at edu.asu.sefcom.fam.model.testing.FAMTesterUtils.getEntryPath(...)
15 at edu.asu.sefcom.fam.model.testing.FAMProvisioningEntryPathTester.innerTest(...)
16 at edu.asu.sefcom.fam.model.testing.FAMMainTester.test(FAMMainTester.java:30)
17 at edu.asu.sefcom.fam.model.testing.FAMProvisioningEntryPathTester.main(...)
```

Figure 6.17: An Excerpt of a Runtime Error Depicting RAC Code.

Chapter 7

A TRUST MANAGEMENT SYSTEM

As mentioned in Chapter 1 and Chapter 3, our proposed FAM federations, as detailed in Chapter 4, are expected to incorporate new organizational members and new end-users over time. In addition, the number of collaborative projects involving shared resources is also expected to grow. In such a context, new and heterogeneous interactions between different organizations and users may be introduced over time, and some of them may involve the participation of entities not having a record of previous successful interactions with each other. In such a case, participants may find it difficult to trust these previously-unknown peers without any previous piece of evidence that can facilitate such a process. In addition, organizations may find it convenient to establish *fine-grained* trust relationships not only at the organizational level, but also at the level of research departments, groups and end-users. With this in mind, this chapter presents an approach tailored to provide a way for participants to define and exchange trust scores on the perceived trust of other participants, in such a way that interactions between previously-unknown peers can be better supported, thus solving the issues for trust management described in Chapter 3.4.2. We start in Chapter 7.1 by providing a definition of trust as well as by providing some use cases for collaborations within the context of our proposed FAM approach. Later, in Chapter 7.2, we present our proposed solution for a trust management system that can be implemented on top of the theoretical model described in Chapter 4 and the enforcement mechanism described in Chapter 5. In addition, we also describe our prototype implementation of such a system in Chapter 7.3 and provide experimental evidence of its suitability for runtime deployments in Chapter 7.4. Later on in Chap-

ter 9.3.1, we will provide a description of how our approach solves the challenges for trust management as described in Chapter 3.4.2.

7.1 Definition of Trust

We start our discussion on trust management by exploring the way trust can be understood in the context of our proposed federations for FAM. As it has been introduced in this dissertation, the main motivation of our approach is to provide access mediation for collaborative resource sharing. With this in mind, an initial definition of trust must include participant organizations trusting each other for the purposes of sharing proprietary resources, e.g., making a *fare* use of them in such a way abusing or unsafe situations are prevented. In addition, participants are also to trust each other for the proper implementation of our proposed FAM approach, including policy specification, retrieval, and evaluation, as well as for the purposes of attribute provisioning: publishing, discovering, derivation and communication. As an example, participants may need to trust the implementation of AD-Rules as provided by other peers, as an incorrectly-implemented AD-Rule may have unpleasant consequences for FAM purposes, e.g., allowing for unintended attribute derivations to take place.

Trust Relationships. With this in mind, our approach for FAM may support different use cases depicting trust relationships between participants. Fig. 7.1 provides a graphic depiction of a basic use case in which the participant serving as the origin of a trust judgment is is known as the *trustor* and the receiving of such trust is known as the *trustee*, thus depicting a *direct* relationship between the two participants. Within our FAM approach, such a relationship may be established when the trustor incorporates attributes that are originated within the domain implemented by the trustee, e.g., when crafting and evaluating a given inter-organizational policy.

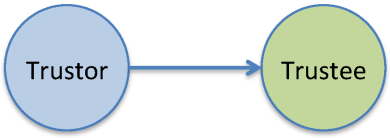

Figure 7.1: A Direct Trust Relationship Use Case.

Moreover, Fig. 7.2 presents a case for an *indirect* relationship when a delegation scheme allows for a trustor to trust a third-party known as the *delegatee*, which is in turn trusted by the aforementioned trustee entity. Within our FAM approach, such indirect relationships may be established by allowing the trustee to transform the attributes of the delegatee into the ones expected by the trustor, e.g., by means of *chained* AD-Rules.

An additional case is show in Fig. 7.3, in which the attributes originated from different trustees are transformed into federated attributes that are in turn leveraged by the trustor, thus depicting a *multicast* situation in which the relationship of trust between trustor and trustees is based on an additional component, e.g., the aforementioned federated attributes, that serves as a layer of indirection that many different trustees may leverage. As with the previous example, transformation of attributes local to trustees into federated ones can be performed by our proposed AD-Rules.

Finally, Fig. 7.4 presents a combined case when the different trust relationships as described above are combined together: the attributes of a delegatee are first transformed into the ones of the trustee, which then are mapped to the federated attributes trusted by the trustor. We believe other combinations like the one depicted in Fig. 7.4 may exist in practice.

Trust in Attribute Provisioning. As it was hinted before during our discussion on trust relationships, the attribute provisioning scheme we have discussed in Chapter 4 and Chapter 5 plays a major role in the establishment of trust relation-

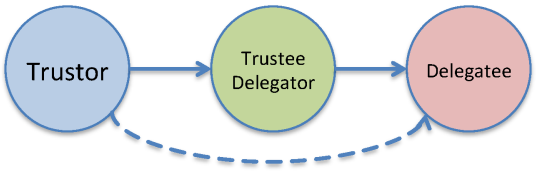

Figure 7.2: An Indirect Trust Relationship Use Case.

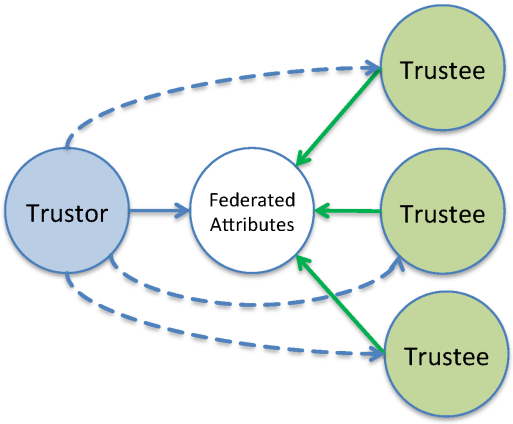

Figure 7.3: A Multicast Trust Relationship Use Case.

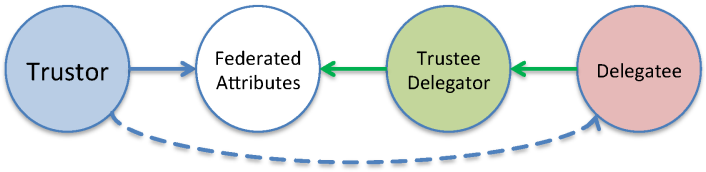

Figure 7.4: A Combined Trust Relationship Use Case.

ships between participants, as the process of path discovery and path walking may be greatly affected by the degree of trust between the different participants providing each of the AD-Rule depicted in those. Fig. 7.6 provides a graphical depiction of an AD-Graph composing an attribute path, which is in turn composed of a sequence of AD-Rules implemented by different participant organizations. In such a figure, the *input* attribute A_3 is to be converted into the *goal* attribute A_1. For each attribute, a corresponding *source entity* is shown, e.g., the participant organization implementing the AD-Rule creating such attribute. In such a setting, the overall result of the

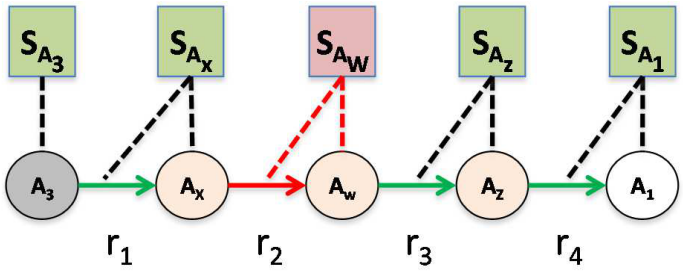

Figure 7.5: An AD-Graph with Source Entities.

path traversal procedure, as depicted in Chapter 5.2.2, may depend upon the trust assigned to each of the source entities included in the path. As an example, if one of the source entities fails to implement its corresponding AD-Rule correctly, e.g., the entity labeled as S_{A_W} producing the attribute labeled A_W in Fig. 7.6, the derivation process for goal attribute A_1 may be affected as a result, thus also possibly affecting the evaluation of policies leveraging such attribute for access mediation purposes.

7.2 Trust Management for FAM

Inspired by previous approaches we will discuss in Chapter 8 as a part of our review of works already in the literature, we now present our approach for establishing a trust management framework intended to work on top of our proposed FAM solution which has been already discussed previously in this dissertation.

Trust Scores. Initially, we aim to model trust as an increasing monotonic numeric value within a certain predefined scale depicting a well-defined range of values. Such a scale is to be determined by the participants of each federation and is expected to be maintained by all of them during the whole duration of collaborative projects. As an example, in a numeric scale depicted by the range [0, 1], the value of 0 may represent the complete absence of trust, e.g., the entity obtaining such a score can be regarded

as *completely untrusted*, whereas on the other hand the value of 1 may be assigned to a *fully trusted* entity. The actual meaning of all intermediate values may be determined by a consensus within the participants of a given federation. In our proposed approach, participants will be allowed to keep their own scores on the perceived trust of other peers within a given federation, and will be allowed to store and change those at will, e.g., by implementing a dedicated local repository. In addition, participants will be allowed to keep trust scores not only at the organizational-level, but also in fine-grained levels such as departmental, research groups, or even end-users, etc.

Trust Scores for AD-Graphs. Following the discussion on trust for AD-Graphs presented before in this chapter, we aim to allow for participants to calculate trust scores on the AD-Graphs depicting attribute paths that were previously obtained from a path discovery process such as the one described in Chapter 5.2.1. As an example, Fig. 7.6 presents an updated version of the attribute path depicted in Fig. 7.5 that contains trust scores for each of the source entities as depicted in the path. In such a figure, and in the rest of this chapter, we assume a trust scale in the range [0, 1] that leverages the value of 0 for untrusted peers and 1 for fully trusted ones, as mentioned before, and also the intermediate values of 3/4 for partially trusted (*well-behaved*) peers, 1/4 for possibly untrusted (*misbehaving*) ones, as well as 1/2 as the initial *neutral* value with all other options do not apply. As an example, the trust score for the source entity producing attribute A_3, denoted $T(S_{A_3})$, is 1, which denotes the entity is fully-trusted. Conversely, the score for the source entity S_{A_W} is 1/2, which represents the neutral initial value within our proposed scale.

With this in mind, the trust score of a given AD-Graph P, denoted $T(P)$, can obtained as follows:

Definition 1 $T(P) = min(\ T(A_1),\ T(A_2),\ \ldots\ T(A_{n-1}),\ T(A_n))$

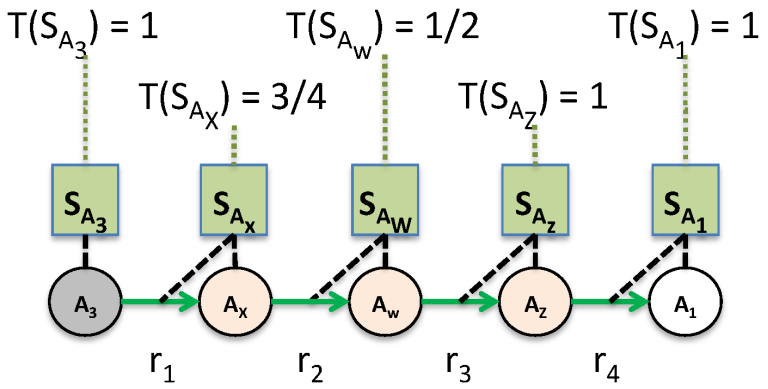

Figure 7.6: Assigning Trust Scores to Source Entities within an AD-Graph.

where T(A_i) is the trust value on each of the attributes involved in P, and can be obtained from the trust score of its source entity, e.g., $T(S_{A_3})$, or from the trust score assigned to the AD-Rule R_i producing it, if a fine-grained approach is in place. As an example, the trust score of the attribute path P as depicted in Fig. 7.6 can be calculated as $T(P) = \min(T(S_{A_3}), T(S_{A_X}), T(S_{A_W}), T(S_{A_Z}), T(S_{A_1})) = \min(1, 3/4, 1/2, 1, 1) = 1/2$.

In addition, we may combine the trust scores of two AD-Graphs P_1 and P_2 as follows:

Definition 2 *$T(P_1, P_2) = min(\ T(P_1),\ T(P_2))$*

As example, Fig. 7.7 presents an alternative AD-Graph that is in turn composed of three inner AD-Graphs, namely P_1, P_2, and P_3. We may calculate the trust scores for each of them individually and combined them together as follows: T(P_1, P_2) = min(T(P_1), T(P_2)) min (1/2, 1) = 1/2. Also, T(P_1, P_2, P_3) = min(1/2, 1) = 1/2.

Recommendations. In addition, we will allow for participants to exchange recommendations with each other on the trust scores assigned to peers within a given federation. This way, participants may ask for recommendations on the trust scores

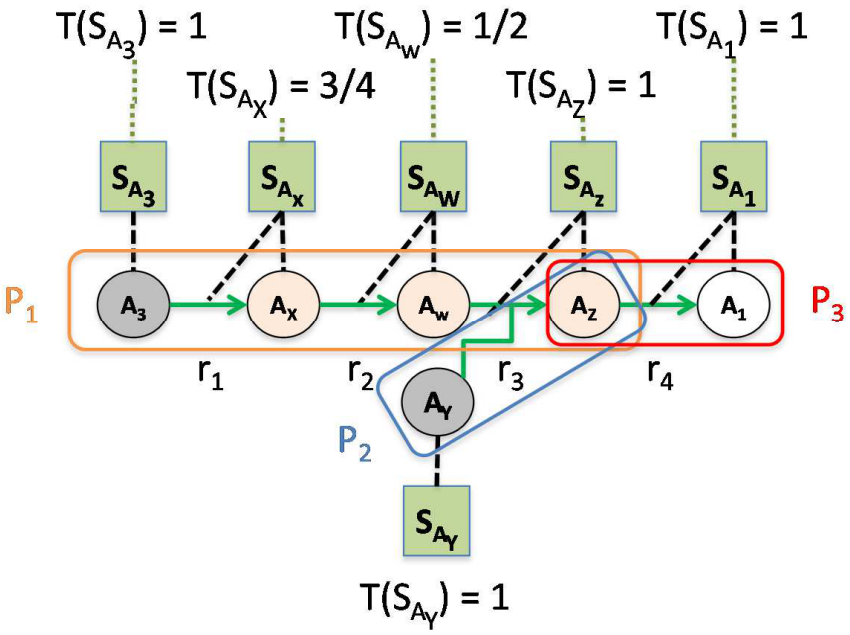

Figure 7.7: Combining Different Trust Scores for Paths within an AD-Graph.

of a participant they have had no previous interaction with. Such exchange may take place upon request, in an *offline* mode independent of any FAM-related operation, or as a part of the access mediation process we have presented in Chapter 4 and Chapter 5. As an example, when calculating the trust score of path P as shown in Fig. 7.6, the score of the source entity labeled as S_{A_W} may be unknown. Following our proposed recommendation system, other peers, e.g., S_{A_Z}, may be able to provide a recommendation on S_{A_W} which can be later used to calculate a trust score for P as shown in Definition 1. Moreover, the value of a given recommendation may also depend on the reputation of the issuing peer. This way, when receiving multiple recommendations for the same entity, highly-trusted peers may provide the most valuable ones. This notion is captured in the following definition:

Definition 3 *T(Recommendee) = T(Recommender) × R(Recommendee)*

where *R(Recommendee)* stands for the trust score received as a recommendation as issued by *Recommender*. In such a setting, each participant organization is required to maintain, at least, the reputation scores of the source entities providing the attributes that serve as an input to the AD-Rules it provides. As an example, in Fig. 7.6, S_{A_Z} will be required to maintain a trust score on S_{A_W} as the AD-Rule it provides (r_3) takes attribute A_W as an input. This way, each source entity in a given path P can provide a recommendation on the source entity located in the *previous* AD-Rule in P. However, a negative recommendation from a *low-trusted* peer may result in a given path being discarded, e.g., S_{A_Z} recommending S_{A_W} in a negative way cause P to evaluate to 0. The same situation may arise when leveraging the recommendation of a previously-unknown peer, e.g., S_{A_W} recommending S_{A_X}.

An algorithm for calculating trust scores for a given AD-Graph, which solves the problem just described, is shown in Fig. 7.8: taking as an input an AD-Graph *Path*, the algorithm starts by inspecting each AD-Rule contained within *Path*, labeled as r, and attempts to retrieve the local trust score for the source entity s implementing r (Lines 3-4). If no local score is found, the algorithm leverages the auxiliary function *getRecommendedScoreOfSource()* in order to retrieve a recommendation on s (Lines 5- 6). In case no recommendation exists, then the default score *DefaultScore*, which has been provided as a parameter, is used (Lines 7-8). In case a recommendation is found, the trust score of the recommender is examined to avoid a lowly-trusted recommender from having an impact on the overall trust score being computed for *Path*. In the case the trust score obtained for the recommender (Line 10), is found to be less than the *MinTrust* parameter, the default *NegativeScore* is used (Lines 11-12). Otherwise, we leverage Definition 3 to calculate the score of the recommendee taking the trust score of the recommender as well as the actual recommended score as an input (Line 14). Finally, the algorithm implements Definition 2 to calculate the

```
1: procedure CalculateTrustScore(Path, MinTrust, DefaultScore, NegativeScore)
2:     result ← 1
3:     for each AD-Rule r in Path do
4:         ruleScore ← getLocalScoreOfSource(r)
5:         if ruleScore = NOT_FOUND then
6:             ruleScore ← getRecommendedScoreOfSource(r)
7:             if ruleScore = NOT_FOUND then
8:                 ruleScore ← DefaultScore
9:             else
10:                recommenderTrust ← getScoreOfPreviousSource(r)
11:                if recommenderTrust ≥ MinTrust then
12:                    return NegativeScore
13:                else
14:                    ruleScore ← recommenderTrust × ruleScore
15:                end if
16:            end if
17:        end if
18:        result ← min(result, ruleScore)
19:    end for
20:    return result
21: end procedure
```

Figure 7.8: Algorithm for Calculating the Trust Score of an AD-Graph.

overall trust score for *Path* (Line 18).

7.3 Implementation

Representing Trust Recommendations. As mentioned previously in this chapter, participants may also want to maintain trust scores at different levels of granularity with respect to other peers in a given FAM federation. As an example, Fig. 7.9 presents an scenario, leveraging the attribute path shown in Fig. 7.5 in which the

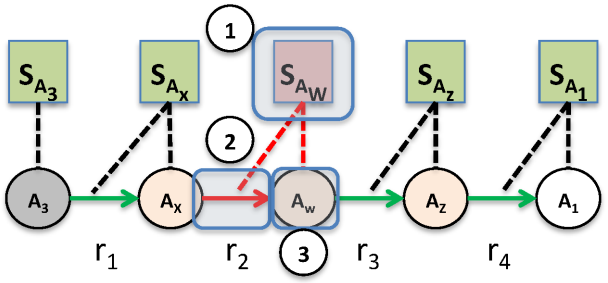

Figure 7.9: Different Cases for Trust Score Assignment.

trust score of the AD-Rule labeled as r_3 is unknown at the moment of calculating a trust score for the whole attribute path. In such a context, the trust score for r_3 can be obtained from the trust score assigned to its source entity S_{A_W} (1), or by a dedicated trust score assigned to r_3 itself (2), or to the actual attribute being produced by it: A_W (3).

As shown before in this chapter, peers may be allowed to keep a local repository for both source entities as well as specific AD-Rule implemented by other peers. For such a purpose, federations may implement their own naming convention system to prevent name clashes, e.g., by using *universal resource identifiers* (URI) (Berners-Lee *et al.* (2005)), thus covering cases (1) and (2) as depicted in Fig. 7.9. For case (3), a trivial approach would append to the recommendation message information about different attributes and their corresponding trust scores. Following the definition of attributes presented in Chapter 4.2.1, the name, the data type, as well as the value of a given attribute, along with a trust score, may be added to the recommendation message, allowing for peers to leverage a finer granularity level. As an example, a given peer may assign a higher degree of trust over the members of a certain research group over the rest. Leveraging our running example, members of a research group, labeled as G, may be fully trusted to initiate a data transfer request such as the one depicted in Fig. 2.1. A recommendation on such a group would then include an entry containing

the 3-tuple: <FederatedGroup, *research.group*, G> and a trust score.

However, this *tuple-based* approach may get complicated when many different tuples, each of them possibly depicting a different trust score, are to be added to a given recommendation message, due to the fact that many attribute-trust score relationships exist. For instance, going back to our running example, let us assume a recommendation is to be made on the federated attribute labeled as S, which depicts a representation of the size of a given data file to be transferred. If a recommendation is to be issued assigning a low trust value to files whose size stays within a certain range, e.g., from 0 to 1200 byes, a 3-tuple depicting each possible value may have to be added to the recommendation message, thus incurring in an excessive increase in the size of the message itself as well as in the processing time to parse and retrieve all of those tuples. Therefore, a direct representation of the mapping between large attribute *sets* and trust scores may not be a suitable approach.

As a solution to this problem, we propose leveraging the XACML language, also mentioned in Chapter 5.1.2, to represent mappings between attribute sets and trust scores, which are then included into an XACML policy file, which gets appended to a recommendation message, thus creating a *recommendation policy*. We aim to leverage the XACML constructs for the specification of policy targets for the definition of trust scores, as well as the constructs intended for specifying attribute-based constraints for policy rules in order to specify attribute sets. As an example, Fig. 7.10 presents a sample policy where a trust score of 0 is assigned to attributes depicting the data size of a given file whose value is greater than 10. The value of the trust score being assigned is included as a resource within the `Target` section (Lines 3-16) using an identifier that is expected to be known to all members of a given federation. The attribute set such a score is mapped to is then specified by means of a rule-based constraint as shown in Lines 18-29.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy>
<Target>
<Subjects><AnySubject/></Subjects>
<Resources>
<Resource>
<ResourceMatch MatchId="...:anyURI-equal">
<AttributeValue DataType=".../XMLSchema#anyURI">
edu.asu.sefcom.fedam.trust.TrustValue.0-0
</AttributeValue>
<ResourceAttributeDesignator AttributeId="..:resource-id"/>
</ResourceMatch>
</Resource>
</Resources>
<Actions><AnyAction/></Actions>
</Target>

<Rule Effect="Permit">
<Condition FunctionId="...:double-greater-than">
<Apply FunctionId="...:double-one-and-only">
<SubjectAttributeDesignator
AttributeId="data.size.S"
DataType=".../XMLSchema#double" />
</Apply>
<AttributeValue DataType=".../XMLSchema#double">
10.0
</AttributeValue>
</Condition>
</Rule>
</Policy>
```

Figure 7.10: An XACML Policy for Representing Trust.

Using XACML, peers may use their own tools for parsing and evaluating a recommendation policy. In addition, the expressiveness, the existing supporting tools, as well as the research literature on using XACML can be also leveraged, thus avoiding the need for introducing a new language for expressing trust score recommendations. Our approach, however, assumes that our recommendation policies are syntax-compliant when shared among participant peers, and also refer to attributes included within the output set of the recommended AD-Rule. As an example, in Fig. 7.9, the recommendation policy on the values of attribute A_W should not refer to any other attribute in the attribute constraint part implemented as an XACML rule, as shown in Fig. 7.10. Finally, we assume our recommendation policies are *conflict-free*, that is, no contradictory rules exist within them. As an example, we assume only a single trust score value is mapped to a single attribute, as defined by the attribute-based constraints implemented as XACML rules as shown before. An approach tailored to detect and remove such inconsistencies may include the work of (Hu *et al.* (2012)).

7.3.1 Architectural Depiction

A graphical depiction of our proposed architecture as well as a workflow for a trust management system for FAM is show in Fig. 7.11: as mentioned before, we aim for each participant peer to maintain a local storage repository for trust scores (1). In addition, peers are to create recommendation policies in the XACML format as shown in Fig. 7.10 by leveraging the information contained within their own repositories. As mentioned before, our recommendation policies may be shared with other peers upon request, e.g., by leveraging an *offline* client-server mode. In addition, we aim to support the dynamic sharing of recommendations following an *online* approach by allowing peers to include their recommendation policies as a part of the AD-Rule entries that are leveraged while performing the path discovery procedure described

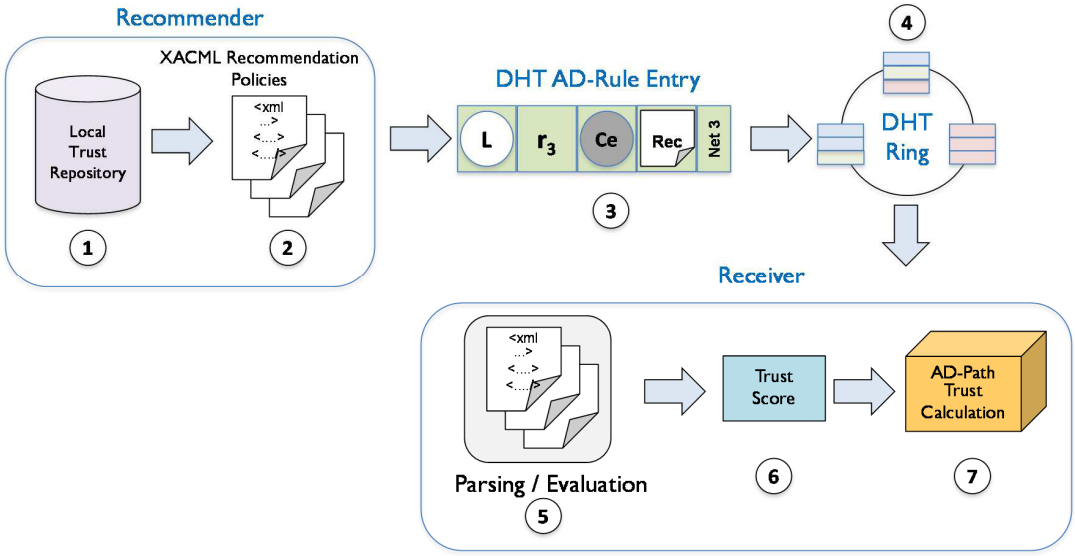

Figure 7.11: A Workflow for a Trust Management System for FAM.

in Chapter 5.2.1. As an example, Fig. 7.11 (3) contains an entry for the AD-Rule labeled as r_3 in Fig. 5.3, that has been modified to include a recommendation policy for the attributes it receives as an input (Ce). Later on, such an entry is to be stored within the DHT structure following the approach described in Chapter 5.2 (4). This way, when constructing paths for attribute provisioning, peers may also collect recommendations on the perceived trust of the AD-Rules depicted in such paths, thus avoiding an extra communication sequence to retrieve those, which may have a considerable impact in runtime performance. Upon receiving a recommendation policy, peers may parse an evaluate it using their own tools (5), thus retrieving the trust scores contained in them (6), which are later forwarded to a trust calculation procedure, following the algorithm depicted in Fig. 7.8.

The source code implementation of our proposed trust management framework comprises 26 Java classes and interfaces that contain 2555 lines of code, as reported by the `cloc` tool (Danial, Al (2015)). Our implementation is organized intro three main software modules: first, a *trust policy administration point* (TrustPAP) module

is in charge of maintaining a local repository for adding, deleting and retrieving trust scores, which is in turn implemented as a `MySQL` (Oracle Corporation (2015)) backend in the form of a relational database. In addition, this module is also in charge of parsing, syntax-checking, adding, deleting and retrieving XACML policies by leveraging a customized version of the XACML Sun Implementation (Sun Microsystems, Inc. (2015)). This way, our TrustPAP module provides support for the steps (1) and (2) as depicted in Fig. 7.11. Next, we have also implemented a *trust policy information point* (TrustPIP) module, which is in charging of preparing the entries for AD-Rule so they contain the XACML recommendation policies as described before in this chapter. Conversely, it also extracts policies contained in AD-Rule entries that have been received from other peers while performing a path discovery process, and provides parsing and syntax-checking capabilities for received XACML recommendation policies. Our TrustPIP module implements step (3) and some of (5) as described in Fig. 7.11. Functionality devised in step (4), e.g., adding and retrieving entries from a DHT structure, is implemented by the software modules described in Chapter 5.2. Finally, we have also provided a *trust policy decision point* (TrustPDP) that is in charge of providing a mapping between the attribute format as described in XACML policies, as shown in Fig. 7.10, and the format for attributes we have presented as a part of our FAM approach in Chapter 4.2.1. In addition, such a module performs the actual evaluation of XACML trust policies and implements the calculation of trust scores over AD-Graphs, as described in the algorithm shown in Fig. 7.8, thus providing support for steps (5) and (6) as shown in Fig. 7.10.

7.4 Experimental Evaluation

We conducted an experimental evaluation tailored to measure the impact on the runtime performance of our proposed trust management framework with respect to

the enforcement mechanism for FAM that is described in Chapter 5. Concretely, we aimed to measure the impact on execution time when calculating trust scores for attribute provisioning paths as described previously in this chapter, either by leveraging locally-stored scores from a repository or by leveraging the recommendations issued by other peers by means of XACML policies. In the rest of this chapter, we present our experimental methodology, our supporting tools as well as a description of the results obtained, including highlights and observed limitations.

Methodology. Initially, we conducted a series of experiments that leveraged simulated AD-Graphs to serve as attribute provisioning paths of different *size*, e.g., different number of composing AD-Rules. First, we conducted an experiment to measure the time taken for calculating an overall trust score for each of the aforementioned paths. For such a purpose, we introduced trust scores within a local repository for each of the source entities implementing the AD-Rules contained in each path under test. Next, we conducted an experiment in which the trust score for each AD-Rule is obtained from a *fixed* recommendation policy composed of three XACML rules, following the approach described in this chapter. Third, for comparison purposes, we conducted an experiment that performs no calculations on trust scores on the same set of paths as described before, thus implementing an approach similar to the one described in the experimental section shown in Chapter 5.2.3.

In an additional series of experiments, we also measured the execution time taken to construct attribute paths of different sizes when trust scores are obtained from recommendation policies depicting a varying number of composing XACML rules, in an effort to better measure the impact of parsing and evaluating different policies.

Supporting Tools. We conducted our experimental evaluation by customizing the implementation of our proposed enforcement mechanism for FAM as described in Chapter 5.1.2. We implemented a simulation engine that creates the AD-Graphs serving as attribute provisioning paths as described in the previous section, and leverages the implementation of our trust management system for FAM as described before in this chapter for adding, retrieving and parsing XACML recommendation policies as well as trust scores that are stored locally within our database repository. Entries for the AD-Rules composing the aforementioned paths where customized to include XACML recommendation policies and inserted into the DHT structure used for the experimental procedure conducted as a part of Chapter 5.2.3.

Results. Results for our first series of experiments are shown in Fig. 7.12. As described before, we implemented AD-Graphs depicting attribute provisioning paths of different sizes, and implemented three approaches on top of each of these paths: trust calculation with locally-stored scores, trust calculation with recommendation policies, and no trust calculation at all. For each experiment, we obtained the execution time taken to construct each path, denoted as *path discovery time* (PDT) as mentioned in Chapter 5.2.3. As it can be observed in Fig. 7.12, the PDT of our approach leveraging locally-stored scores, and the one depicting no trust calculations, stays within comparable means, and grows linearly as the number of composing AD-Rules increases, thus showing that the introduction of local repositories has no noticeable impact in runtime performance. In addition, the PDT observed for an approach when each AD-Rule in a given path contains an XACML recommendation policy, which is then parsed an evaluated for retrieving trust scores for calculations, depicts a manageable increase with respect to the two other options just described. As expected, such a difference increases with respect to an also increasing number of AD-Rules and

recommendation policies to process.

Results for an experiment varying the number of AD-Rules containing recommendation policies are shown in Fig. 7.13. This time, only a subset of such AD-Rules composing attribute paths of varying sizes, e.g., 5, 20 and 50 AD-Rules, where augmented with recommendation policies. Once again, a fixed recommendation policy containing 3 XACML rules was used to serve as a parameter of common reference. As expected, the overall PDT time observed increases as the number of AD-Rules from which a trust score must be obtained from as recommendation policy increases, due to the parsing and evaluation procedures as described before in this chapter.

Finally, Fig. 7.14 shows the results of an experiment intended to vary the number of composing XACML rules for our proposed recommendation policies. Once again, as the number of such rules increases, so it does the overall time taken to parse and evaluate a given policy as a result, thus ultimately affecting the obtained PDT. As with the previous experiment, increasing the number of composing AD-Rules has a direct linear impact on the overall execution time observed on each experiment instance.

Limitations. While the results observed for our experimental process provide evidence of the suitability of our approach for real-life deployments, still some limitations in both our proposed approach as well as in the experimental process arise. First, as observed in Fig. 7.13 and Fig. 7.14, the process of parsing and evaluating XACML recommendation policies has a noticeable impact at runtime. While we expect the number of composing AD-Rules contained in real-life attribute provisioning paths to be small, e.g., less than 6, and we also expect the number of such AD-Rules whose trust scores can be obtained from recommendation policies to be small as well, e.g., less than 3, still the processing of recommendation policies containing many different

XACML rules may be an important factor affecting runtime performance, as shown in the experimental results contained within Fig. 7.14. As a solution, in Chapter 9.3.2 we present a discussion on an alternative approach in which federated peers may agree on a more suitable representation for XACML policies that eludes the text-based parsing and evaluation procedure, e.g., a binary representation that contains well-defined bit segments that can be retrieved and processed more efficiently.

In addition, an extended experimental process may be needed to properly assess the benefits of recurring to a constraint-based language such as XACML instead of a direct tuple-based representation for recommendation policies, as described previously in this chapter. Even when the results shown in Fig. 7.14 suggest that leveraging a small number of constraint-based XACML rules for defining fine-grained attribute-trust score relationships may be a suitable option, an extended experimental process may be required to further corroborate such a claim.

Finally, even when our experimental process relied on simulated AD-Graph that may provide a good abstract representation of real-life settings, still a further experimental procedure taking into account recommendation settings obtained from realistic scenarios is required. As an example, several partnership organizations may in fact implement recommendation settings depicting different trust relationships, in a similar way to the ones depicted in Chapter 7.1.

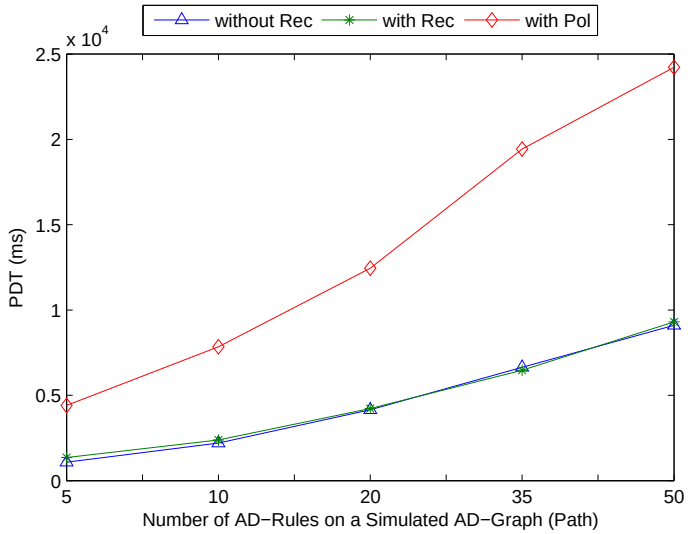

Figure 7.12: An Experimental Evaluation for a Trust Management System.

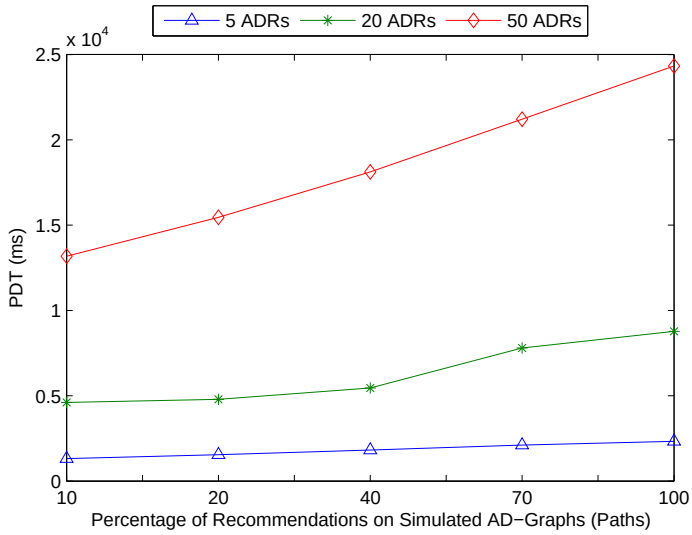

Figure 7.13: Varying Recommendation Policies in a Trust Management System.

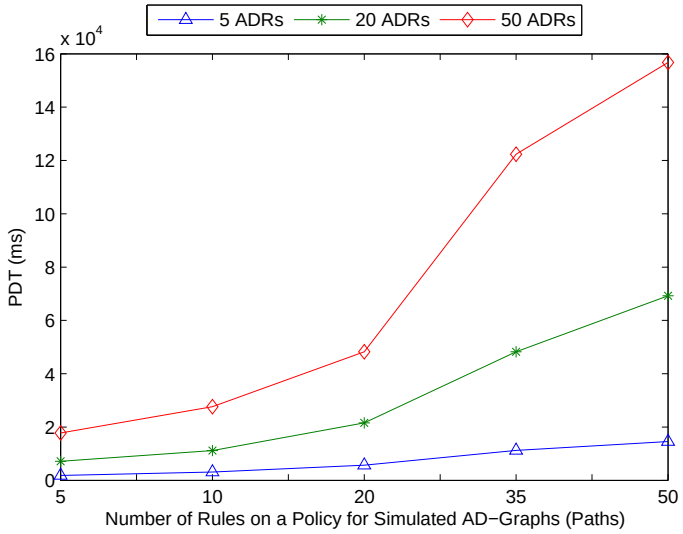

Figure 7.14: Varying XACML Rules for Policies in a Trust Management System.

Chapter 8

RELATED WORK

8.1 Attribute-based Access Control

Using the requirements for ABAC stated in Chapter 3.1 as a comparison criteria, we now examine existing approaches in the literature. One of the initial attempts in defining ABAC was proposed by (Wang *et al.* (2004)), who explored the use of logic programming and computable set theory for modeling the main features of ABAC, taking into account a web-based context. Our work depicts a similar approach by proposing the evaluation of attributes through functions, but does not depend on any particular supporting methodology for its implementation. In a similar context, (Yuan and Tong (2005)) explored the use of ABAC for the specification of access control policies for web services. Moreover, (Priebe *et al.* (2006)) presented an approach leveraging the concepts of *ontologies* and the *semantic web* in order to formalize the notion of ABAC. Another approach leveraging the concept of the semantic web for ABAC include the one of (Cirio *et al.* (2007)), who provide a model definition including attribute-based constraints. Such an approach, however, does not provide a formalization of ABAC that can be regarded as independent of any other supporting technologies. The work of (Zhu and Smari (2008)) and (Smari *et al.* (2009)) has provided a definition of both attributes and attribute-based access control constraints tailored for supporting collaborative software systems. In the context of grid computing, the approach presented by (Lang *et al.* (2009)˙ also provided an ABAC model mostly focused on the definition of attributes and the access control constraints. In addition, (Covington and Sastry (2006)), introduced the *contextual attribute access*

control (CABAC) model, which was realized in mobile applications. As in their approach, our model leverages the notion of attributes and their relationship with access control entities (including modeling the runtime context as an access entity). However, our approach takes a step further by describing the way such attributes are mapped to access rights (permissions) by means of our proposed AD-Rules and AD-Graphs. Recently, (Jin *et al.* (2012)) proposed a noticeable approach intended to formalize a series of ABAC model *families*. In addition, the relationship between ABAC and other well-known access control models was explored. We have been influenced by this paper with respect to the definition of attributes as described in Section 4.2.1. However, our approach introduces a notion of attribute derivations to capture the mapping between attributes and corresponding access rights. Finally, another interesting approach was presented by (Zhang *et al.* (2005)), who developed their *attribute-based access control matrix*, which extends classical theory in the field of access control to accommodate attributes as well as the notion of security state. However, it provides no definition for attribute-based constraints, which is considered in our approach by means of our proposed AD-Rules and AD-Graphs.

Table 8.1 shows a summary on a literature survey on recent papers addressing ABAC taking the aforementioned design goals as a criteria for comparison purposes. As shown, no approach proposed so far meets all the criteria we have identified as a part of this dissertation.

8.2 Approaches for Federated Security

The problem of providing security guarantees in inter-organizational settings has been largely addressed in literature. In particular, several *federated identity* (Chadwick (2009)) approaches have been introduced to allow partnering organizations to reuse locally-issued credentials when accessing resources located under the scope of an exter-

Table 8.1: A Survey on Recent Approaches for ABAC.

	Attributes	**Constraints**	**Model Formalization**	**Security State**	**Platform Independence**
Wang *et al.* (2004)	√	√	√	×	×
Yuan and Tong (2005)	×	×	×	×	×
Zhang *et al.* (2005)	√	×	√	√	√
Covington and Sastry (2006)	√	×	×	×	×
Priebe *et al.* (2006)	×	×	√	×	×
Cirio *et al.* (2007)	×	√	√	×	×
Zhu and Smari (2008)	√	×	×	×	×
Lang *et al.* (2009)	√	√	×	×	×
Smari *et al.* (2009)	√	√	×	×	×
Shen (2009)	×	×	×	√	×
Wei *et al.* (2010)	×	×	√	×	×
Jin *et al.* (2012)	√	×	√	×	√

nal security domain. As an example, OpenID (Recordon and Reed (2006)) and Shibboleth (Morgan *et al.* (2004)) have recently gained acceptance in both industry and academia respectively for user-credential sharing. Our approach builds on this idea by allowing participants to exchange federated *attributes*, thus potentially allowing for such attributes to serve as *tokens* granting access to shared resources, in an approach also inspired by Kerberos (Neuman and Ts'o (1994)), OAuth (Jones and Hardt (2012))

and more recently, Facebook Login (Facebook Inc. (2015)), strive to allow third-party applications to leverage the user credentials defined for the popular social network to access application-dependent resources. However, our approach goes a step further by providing stronger support for defining, evaluating and enforcing policies in both the local and federated contexts. In addition, our proposed attribute derivation schemes, e.g., AD-Rules and AD-Graph, are also novel in the context of federated inter-organization settings.

Moreover, our AD-Rules are inspired by the idea depicted in the credential-discovery protocol proposed by the RT Framework (Li *et al.* (2002)), which allows for credentials issued by independent domains to be located and leveraged for access management. Similar to the RAMARS Framework (Jin and Ahn (2009)), our AD-Rules are depicted in a graph-like structure that allows for user-defined attributes to be transformed into a set of widely-recognized credentials. However, the RAMARS framework assumes each security domain implementing the transformation functions may be *partially* trusted by modeling trust in the range [0,1]. In Chapter 7, we present a similar approach that differs in the way an overall trust score among peers providing entries in a given AD-Graph is calculated.

Our approach is also inspired by existing research-oriented infrastructures for sharing resources: the OSCARS system (Guok *et al.* (2006)) provides a way for organizations to reserve shared resources within high-performance networks, e.g., network bandwidth. In a similar context, the perfSONAR (Hanemann *et al.* (2005)) infrastructure allows for partnering organizations to obtain and share network-related information obtained within their administrative domains such that inter-organizational network problems can be better diagnosed and resolved.

In addition, recent approaches leveraging federated identity for sharing resources include the work of (Broeder *et al.* (2012)) and Globus Nexus (Ananthakrishnan *et al.*

(2013)). Moreover, (Klingenstein (2007)) and (Chadwick and Inman (2009)) incorporate the concept of end-user attributes with the federated identity. Our solution differs from this approaches as it includes attributes originated from different access control entities rather than considering credentials from end-users only.

A survey depicting a literature review of existing approaches for federated security and collaborations, which leverages the comparison criteria as defined in Chapter 3.3, is shown in Table 8.2.

8.3 Software Assurance

The work presented in Chapter 6.1 is related to other efforts in software security, most notably, architectural risk analysis and language-based security. Architectural risk analysis (McGraw (2006)) attempts to identify security flaws on the level of the software architecture and hence is unrelated to the source-code level addressed in this approach. Language-based security approaches in the sense of Jif (Sabelfeld and Myers (2003)) allow software to be verified against information flow policies rather than supporting specific security requirements for software construction. In addition, formal verification of RBAC properties has been already discussed in the literature (Hu and Ahn (2008)). These approaches are mostly focused on verifying the correctness of RBAC models without addressing their corresponding verification against an implementation at the source-code level.

The work closely related to ours involves the use of DBC, which was explored by (Dragoni *et al.* (2007)). In addition, (Belhaouari *et al.* (2012)) introduced an approach to the verification of RBAC properties based on DBC. Both approaches, while using DBC for checking RBAC properties, do not include the use of reference models to better aid the specification of DBC constraints in the security context. Moreover, no support is provided as customized source-code implementations,

Table 8.2: A Survey on Recent Approaches for Federated Security/Collaborations.

	Policy Discovery and Eval.	**Domain-Specific Functionality**	**Attribute Provisioning**	**Existing Infrastructure**	**Scalability and Independence**	**Decentralized Implementation**
Kerberos (Neuman and Ts'o (1994))	√	√	×	√	√	×
RT (Li *et al.* (2002))	×	√	×	×	√	√
Shibboleth (Morgan *et al.* (2004))	×	√	×	√	√	√
perfSONAR (Hanemann *et al.* (2005))	×	√	×	×	√	√
OpenID (Recordon and Reed (2006))	×	√	×	√	√	√
OSCARS (Guok *et al.* (2006))	×	√	×	×	√	√
(Klingenstein (2007))	×	√	×	√	×	√
(Chadwick and Inman (2009))	×	√	×	√	×	√
RAMARS (Jin and Ahn (2009))	√	√	×	√	√	√
(Broeder *et al.* (2012))	√	√	×	√	×	√
OAuth (Jones and Hardt (2012))	×	√	×	√	√	√
Globus Nexus (Ananthakrishnan *et al.* (2013))	×	√	×	√	×	×
Facebook Login (Facebook Inc. (2015))	×	√	×	√	×	×

Table 8.3: A Survey on Recent Approaches for Software Assurance.

	V&V Independent Impl.	**Proper Access Mediation**	**Integrating *In-House* Sys.**
(Cataño and Huisman (2002))	×	√	√
(Sabelfeld and Myers (2003))	×	√	×
(McGraw (2006))	×	√	×
(Dragoni *et al.* (2007))	×	√	×
(Hu and Ahn (2008))	×	√	×
(Lloyd and Jürjens (2009))	×	√	×
(Belhaouari *et al.* (2012))	×	√	×
(Mustafa and Sohr (2014))	×	√	√

which are in turn supported by he JML model capabilities discussed in our approach. Other works that apply a DBC approach based on JML in the security context are presented by (Lloyd and Jürjens (2009)) (biometric authentication system), (Cataño and Huisman (2002)) (smart card system), and (Mustafa and Sohr (2014)) (Android system services). These works, however, do not cover applications consisting of highly-customizable modules and do not leverage dynamic testing techniques for assertion-based verification.

A survey of related approaches for software assurance that takes into account the criteria defined in Chapter 3.4.1 is shown in Table 8.3.

8.4 Trust Management Systems

As detailed in Chapter 3, in this dissertation we have striven to provide a trust management system that can effectively allow for participant organizations within our proposed federations to trust previously-unknown peers for the purposes of resource sharing. In addition, we have also striven to accommodate for such a framework within the theoretical model as well as the enforcement mechanisms we have devised for our approach, as discussed in Chapter 5. With this in mind, we now present a description of several different approaches for trust management found in the literature. For each of them, we briefly detail their most noticeable features, at the same time we provide a comparison with the approach for trust management we have presented in Chapter 7, including similarities, points of inspiration, and differences.

Trust management has been largely explored in the literature. Over the years, specialized trust policies languages have been introduced in an effort to better capture the relationships between trustors and trustees, as introduced in Chapter 7.1. Noticeable examples include Ponder (Damianou *et al.* (2001)), KAos (Uszok *et al.* (2004)) and Rei (Kagal *et al.* (2003)). While such approaches may be extremely convenient for modeling complex trust relationships and related context, we have resorted to the use of XACML, the *de facto* language for authorization, as we are mostly concerned with the communication of recommendation scores rather than expressing complex trust relationships. In addition, we believe XACML provides a common ground in which existing literature and tools based on such a language can be further leveraged.

As introduced in Chapter 2.5, the definition of risk within a certain community represents an important factor for determining trust relationship between different members. In such a context, different approaches in literature have represented risk by introducing a mathematical model that combines scores obtained from different

dimensions. As an example, Regret (Sabater and Sierra (2001)) presents an approach tailored for an electronic marketplace where different dimensions such as overcharging customers, late product delivery as well as overall quality are combined together to produce a trust score over different vendors included in the system. In a similar approach, PeerTrust (Xiong and Liu (2004)) provides a decentralized approach for peer-to-peer networks where participants combine scores obtained from direct feedback, quantity of previous transactions as well as credibility. Moreover, GRAft (Hendrikx and Bubendorfer (2013)) allows for communities to provide a customized set of dimensions for risk representation. Later on, each community member may decide how to represent and combine those for a final trust calculation. In our approach, we have chosen to provide a single trust score to combine different dimensions such as fair use of shared resources, confidence on the implementation of enforcement mechanisms, as well as a degree of confidence in the implementation of our proposed AD-Rules. Whereas resorting to single combined score may be sufficient for most cases, specialized federations may find it convenient to provide a separate score for each of these dimensions. We will discuss an approach for such a purpose while we describe our plans for future work in Chapter 9.3.2.

Trust management systems in the literature have also struggled to provide frameworks for the introduction of possibly-untrusted newcomers to a given community, a.k.a., federations in our approach. In addition, different approaches have been introduced in order to establish and communicate recommendations on the observed behavior of both newcomers and old members. As an example, the Confidant framework (Buchegger and Le Boudec (2002)) provides an approach in which all members are initially fully-trusted, and recommendations are only issued when a misbehaving peer is detected, thus in fact implementing an alarm-based system. A similar approach is implemented by P-Grid (Aberer and Despotovic (2001)): negative rec-

ommendations, a.k.a., complaints, are issued as a response to malicious behavior, and are later distributed for storage to a subset of participants. Later, when a peer is interested in calculating a trust score on another peer, the framework can be queried to retrieved the set of stored complaints, which are then combined together using a mathematical approach. In our proposed trust framework, we allow for participants to issue negative recommendations that are to be later forwarded through the entries describing AD-Rules. When malicious behavior is detected, a given peer may update the entries stored in the distributed implementation based on DHT in such a way that a new negative score is introduced for the faulting peer. Later on, other peers will be notified of the complaint when constructing their own AD-Graphs following the approach described in Chapter 5.2.1. Our approach for the combination of trust scores is influenced by RAMARS (Jin and Ahn (2009)), which allows for so-called *trust chains* to be created when leveraging recommendation scores that are issued by other participants in a given community. This way, the level of trustiness assigned to recommenders may also influence the degree of confidence that is ultimately assigned to their recommended scores. We have captured this notion in Definition 3.

Another important factored mentioned in the literature is the support for the recalculation of trust scores over time. Ideally, peers should be allowed to update the trust scores assigned to other peers by taking into account a history of previous interactions. As an example, R^2Trust (Tian and Yang (2011)) combines a history of previous interactions with a given peer along with a set of recommendations obtained from others in order to provide a new trust score, which can be then used for future decisions. A similar approach is implemented by popular websites such as Amazon (Amazon Inc. (2016)) and Stackoverflow (Stackoverflow (2016)), which allow for end-users to provide feedback on a history of transactions. Later on, such history, along with the reputation given to the end-users themselves, is used to calculate a trust

score over certain resource, e.g., an item for purchase or a blog post detailing an answer to a technical issue, respectively.

Over the years, different architectural depictions have been proposed for trust management systems. As an example, the trust framework implemented by eBay (eBay (2016)), (Melnik and Alm (2002)) has been largely studied in the literature and depicts a centralized architectural style in which all scores and calculations are performed in a *logically* centralized location, e.g., a proprietary cloud implementing the same security domain. In contrast, several approaches have been introduced in order to provide decentralized architectures that leverage the distributed nature that is featured by most communities. As an example, XRep (Damiani *et al.* (2002)) and EigenTrust (Kamvar *et al.* (2003)) leverage DHT to exchange recommendation on trust scores that are later combined in a mathematical-based approach by participants. Moreover, RateWeb (Malik and Bouguettaya (2009)) provides a decentralized approach that allows for peers to obtain recommendations on different web services. Before utilizing a given web service, a peer may obtain recommendation score on it from other peers and may combine those to obtain an overall trust score, which is then used to determine if the web service should be invoked or not. In our approach, we have striven to leverage the distributed architecture depicted by our proposed DHT implementation as much as possible, allowing for peers to locally store and later recommend trust scores by adding them in the form of recommendation policies, which are stored and retrieved from the DHT by other peers when attempting to provision attributes.

Moreover, (Resnick *et al.* (2000)) provided an study on the requirements for successful trust recommendation systems: first, the participant peers must be long-lived, that is, they should stay in a given community for a considerable amount of time with respect to the lifetime of such community. Second, recommendation scores must be

captured, distributed and made available for use in the future. Third, recommendation scores must be ultimately used to guide decisions in the context of the enclosing community. We believe our proposed trust management framework, as depicted in Chapter 7, meets the three aforementioned requisites as follows: first, we expect participant organizations to stay within our proposed federations for a considerable amount of time, engaging in more collaboration projects over time due to the benefits entitled to the use of shared resources. Second, we provide means for the definition, storage and communication of trust scores via recommendations. Finally, we have also provided an approach for guiding the functionality of our attribute provisioning scheme by leveraging trust scores and recommendations, following Definitions 1, 2 and 3.

Finally, Table 8.4 presents a comparison between the approaches presented in this chapter and the requirements for a trust management system as introduced in Chapter 3. As shown, no approach found in the literature solves all the issues with respect to trust management in the context federations for access mediation as they are detailed in Chapter 3.4.2. Later on, in Chapter 9.3.1 and Table 9.4, a discussion is presented on how the approach introduced in this dissertation effectively fulfills such requirements.

Table 8.4: A Survey on Recent Approaches for Trust Management.

	Trusting Unknown Participants	**Incorporating Security Functionality**	**Fine-grained Trust Relationships**	**Providing a Mathematical Foundation**	**Leveraging Community Experience**	**Providing a Decentralized Approach**
Regret (Sabater and Sierra (2001))	√	√	√	√	√	×
Confidant (Buchegger and Le Boudec (2002))	√	√	√	×	√	√
XRep (Damiani *et al.* (2002))	√	√	×	×	√	√
EigenTrust (Kamvar *et al.* (2003))	√	√	×	√	√	√
P-Grid (Aberer and Despotovic (2001))	√	√	×	√	√	√
PeerTrust (Xiong and Liu (2004))	√	√	×	√	√	√
RateWeb (Malik and Bouguettaya (2009))	√	√	×	√	√	√
R^2Trust (Tian and Yang (2011))	√	√	×	√	√	√
GRAft (Hendrikx and Bubendorfer (2013))	√	√	√	×	√	√
Amazon (Amazon Inc. (2016))	√	√	√	√	√	×
eBay (eBay (2016))	√	√	√	√	√	×
Stackoverflow (Stackoverflow (2016))	√	√	√	√	√	×
Delegent (Firozabadi and Sergot (2003))	√	√	√	×	√	×
RAMARS (Jin and Ahn (2009))	√	√	×	√	√	√

Chapter 9

DISCUSSION AND FUTURE WORK

In this chapter, we present some discussion on the topics composing the main contributions of this dissertation, which have been presented in Chapters 4, 5 and 6. In addition, we also identify some opportunities for improvement and present some guidelines for future work that can effectively improve the suitability of our approach for production environments.

9.1 FAM Model and Implementation.

9.1.1 Addressing Challenges

The approach proposed in Chapter 4 provides a solution to the problems we have described in Chapter 3.1 as follows: first, we provide a well-defined description of attributes, e.g., their inner components, that is intended to solve the *attribute definition* problem, allowing for attributes to be defined in a local as well as in a federated context in a clear and unambiguous way. In addition, our proposed model also addresses the *attribute constraints* problem by modeling attributes as 3-tuples that can be used to define constraints on top of them, e.g., by restricting the data type as well as the set of values a given subset of attributes is expected to have in order to meet a certain constraint for an ABAC policy. In addition, our proposed AD-Rules may be also used to model constraints due to their definition as mathematical functions, as described in Chapter 4.2.3.

In a similar fashion, our model, as shown in Fig. 4.5 and Fig. 4.6, also provides a well-defined description of the way the inner components of ABAC, e.g., access enti-

ties, attributes and permissions, interact with each other, thus providing a solution to the *model formalization* problem described before in Chapter 3.1. The problem of providing a definition for modeling the *security state* of a given collaborative setting that includes resource sharing can be also solved by means of our proposed definition of attributes, and the AD-Rules and the AD-Graphs based on them. As an example, the AD-Graph shown in Fig. 4.7 depicts a case when the authorization to perform a security-sensitive operation such an inter-organizational data transferring is modeling by means of a federated attribute *Ta*, which is in turn produced by applying several AD-Rules to a well-defined set of both local and federated attributes belonging to the organizations depicted in such a transferring operation. In such a setting, attribute *Ta* serves as an access token that represents the `TransferFilePermission` depicted in Fig. 2.2, thus providing a representation of a state in the context of a given collaboration in which a security-sensitive operation can be carried on. Moreover, our proposed approach also meets the goal of being *independent of any supporting technology* as we have opted to propose a model formulation that can be implementing without requiring any basic platform and/or previous knowledge on any previous methodology other than the set theory formulations depicted in Fig. 4.5 and Fig. 4.6.

In addition, our proposed model also supports the specification, enforcement and evaluation of both intra and inter-organizational policies, by defining them as descriptions of attributes that are related to a given permission over a protected resource. Also, our model defines how policies relevant to a given access request can be located and evaluated in the context of mediating access to a given resource request. In the same context, our approach encourages the *reuse of existing infrastructure* as our proposed AD-Rules (and AD-Graphs), while formally defined within our model as function mappings within attributes, are expected to be implemented at the source level in customized ways by participant organizations, which can then leverage lo-

cal attributes originated in such existing security systems, in such a way that they can be later transformed into federated attributes. Our proposed AD-Rules and AD-Graphs also provide a strong foundation to support the *attribute derivation* features also explained in Chapter 3.1. Such a scheme is complemented in Chapter 5 when an approach for supporting the *path discovery* and *path traversal* procedures described in Chapter 4.2.4 is introduced. Finally, we believe our approach provides strong support for implementing authorization features that go far beyond the support for authentication that is the main concern in the inspiring existing frameworks for federated security described at the end of Chapter 3.1. A complete description matching each identified problem and our proposed solution is presented in Table 9.1.

As mentioned in the introduction of Chapter 5, our proposed enforcement mechanism is intended to address the challenges for the implementation of federated access management in the context of collaborations as described in Chapter 3.3. First, our proposed mechanism provides a solution to the *policy discovery and evaluation* problem by implementing the policy administration, evaluation and discovery layers discussed in Chapter 5.1.2. In addition, our approach also strives to *incorporate specific functionality* derived from a collaboration setting, e.g., a data file transfer, by means of our policy administration and evaluation layers, which can then be used to provide policies that mediate access to such specific tasks only to authorized entities. Moreover, our attribute discovery and derivation layers, as well as our DHT-based architecture provide support for implementing the *attribute provisioning* challenge also described before, by allowing for implementers to defined customized AD-Rules that can be later leveraged by end-users for transforming local and federated attributes into federated ones as needed in the context of a given collaboration. The challenge of *reusing existing infrastructure* is also addressed by means of our AD-Rules, which, as mentioned earlier in this chapter, are to be implemented in a customized way

Table 9.1: Addressing the Challenges Devised for an ABAC-based Model.

	Resource Identification	**Attribute Identification**	**Attribute Mapping**	**Attribute Discovery**	**Administration**	**Policy Retrieval**	**Attribute Provisioning**	**Policy Dispatch**	**Results Aggregation**
Attribute Definition	√	√	√	√			√		
Constraints Definition			√				√		
Model Formalization		√	√	√	√		√		
Definition of Security State	√	√	√	√			√		
Independence of Supporting Methodology		√	√	√		√	√	√	
Intra and Inter-organizational Policies	√	√	√	√	√	√	√	√	√
Reusing Existing Infrastructure		√	√	√			√		
Attribute Derivation		√	√	√	√		√		
No Native Support for Authorization		√	√	√	√	√	√		√

by each participant organization, thus allowing them to use existing security-related systems as a *backend* providing support for such custom-made functionality. Such a *platform independence* feature is also supported by our attribute discovery and DHT architecture as proposed in this chapter, which can be in turn supported by each participating organization by means of different underlying frameworks, assuming the basic protocols for each of those layers, e.g., path discovery and path traversal as stated in Chapter 5.2.1 and Chapter 5.2.2, are properly met.

Moreover, our DHT distributed implementation described in Chapter 5.2 may also

support the overall *scalability* of our approach by allowing to many new members to enter or exit our proposed federations without the need for storing information about memberships in a centralized node. This way, new members may only need to join the DHT ring for the purposes of path discovery and provide an implementation for the AD-Rules they make available to other peers. With many new members join a given federation, many different paths may be obtained as a result when attempting to provision a given attribute. Peers may then want to implement their own strategies for deciding which paths should be attempted to traverse. As an example, peers may leverage the trust framework presented in Chapter 7 for choosing the paths whose composing entries are provided by the peers whose degree of trust is found to be optimal. Finally, our decision of implementing our approach by means of custom-made AMA agents running on each participant organization, as well as the decision of organizing the path discovery procedure into a distributed architecture based on DHTs, provides strong support for addressing the challenge of *decentralized implementation*, as described at the end of Chapter 3.3. Table 9.2 presents a summary on the topics addressed in this section.

9.1.2 Future Work

Attribute Provisioning. As shown in Chapter 5.2.3, efficient provisioning of federated attributes is crucial for processing and resolving policies in a timely manner. The attribute provisioning scheme presented in Chapter 5.2 supports this goal by reducing the number of communication messages between participating domains to determine if a given AD-Graph depicts a path between a pair of attributes. Each participant organization should decide the number of times it will attempt to retrieve new entries from a DHT ring when constructing a given path. As an example, an organization may set a limit of three explorations of the DHT ring while trying to find

Table 9.2: Addressing the Challenges Devised for an Enforcement Mechanism.

	Policy Admin. Layer	Policy Evaluation Layer	Policy Discovery Later	Attribute Derivation	Attribute Discovery	DHT Implementation
Policy Discovery and Evaluation	√	√	√			
Incorporating Domain-Specific Functionality	√	√				
Attribute Provisioning				√	√	√
Reusing Existing Infrastructure				√		
Scalability and Platform Independence				√	√	√
Decentralized Implementation			√	√	√	√

a set of input attributes for AD-Rules that fall under the scope of its local domain. Setting a low limit of explorations might prevent participants from discovering a potential path in the AD-Graph. However, a large limit may increase the attribute provisioning time, thus possibly affecting the overall processing time of a given policy. In addition, due to the fact DHTs require participants to locally store only a subset of all the entries included in a given ring, our scheme allows participants to store only a subset of AD-Rules entries, thus potentially relieving them from storing information related to the complete AD-Graph. In this way, the process of adding and removing AD-Rules is significantly simplified, thus providing a means for modifying a given AD-Graph to better meet the specific goals devised for collaborations, e.g., adding new AD-Rules to handle user credentials from a new participating domain.

Future work may also focus on providing enhancements to our attribute provisioning technique. Concretely, we plan to explore alternative techniques for attribute provisioning by means of traversing the AD-Graphs. As an example, federated domains may want to avoid repeatedly traversing a given AD-Graph by *caching* previously-produced attributes for a certain time. In addition, other domains may find useful to introduce alternative *shortcut* paths within a given graph to replace frequently used ones, e.g. introducing local AD-Rules to replace external ones. Another approach may consider caching the paths themselves, assuming no frequent updates to the entries in a given DHT ring are made, thus possibly moving directly to the path traversal phase when provisioning attributes.

With this in mind, a *federated attribute reachability* process may take place as follows: in an initial step, peers locate the set of federated attributes within a given federation that are more likely to be provisioned at runtime by the entities, e.g., end-users, within they own local domain. Next, peers can locate the set of federated policies devised for the collaborations they take place in, and query the AD-Graph defined for such federation in a periodical way to retrieve all the paths leading to the attributes defined for such policies. Finally, such paths can be located and stored in a local repository for faster retrieval in the future. Such a process may also allow for peers to ensure their local attributes can be effectively transformed into the federated ones required for inter-organizational policies, thus avoiding further trouble that may arise in case no effective provisioning can take place at runtime.

Resource Allocation. As mentioned before, the approach proposed in this dissertation relies heavily in the successful implementation of our propose attribute provisioning scheme, as depicted in Chapter 5.2. Following the trust management scheme introduced in Chapter 7, some peers may become highly-trusted within the context of a given federation, thus possibly luring other peers to leverage the AD-Rules

they provide in a more frequent way than the ones provided by other less-trusted peers. In such a context, *popular* highly-trusted peers may introduce a resource allocation mechanism to allow for the AD-Rule they provide to handle an increasing number of requests in an efficient and timely way.

Delegation. Besides supporting resource sharing by means of attribute provisioning, we believe our approach provides support for delegation, privacy, and the *outsourcing* of access control mechanisms. As an example, end-users may delegate their access rights over resources they *own* by composing AD-Rules that take the local credentials from both the owner and the delegatee, producing a proper federated attribute as a result. Federated peers may also be relieved from implementing local AD-Rules by themselves by allowing other peers to produce federated attributes on their behalf, following an outsourcing approach based on a previously established trust relationship. In this way, domains need not implement a complete implementation mechanism for joining a given federation, thus encouraging the introduction of new federated peers.

Privacy. A basic privacy model may be implemented on top of our approach by allowing for sensitive information contained in locally-defined attributes not to be revealed to other organizational peers when producing federated attributes. For instance, in Fig. 4.7, sensitive information in attribute *Ce*, e.g., a user's full name, may be replaced by a *pseudonym* in the *L* attribute produced by the AD-Rule labeled as r_3, following an approach similar to the one depicted by (Iwaihara *et al.* (2008)). In such a setting, the local security domain (Net 1) may want to keep track of this process for accountability purposes. An alternative approach may allow for end-users to hide sensitive attributes at request time by incorporating techniques such as the *privacy-preserving attribute-based credentials* (PABC) proposed by (Camenisch *et al.* (2014)).

Policy Indexing and Conflict Resolution. Efficient discovery and retrieval of policies (as shown in Chapter 5.1.2) may benefit from the use of indexing techniques based on both policies and attributes on each federated peer, following an approach similar to the one presented by (Marouf *et al.* (2011)). In addition, a comprehensive policy specification framework is critical to detect and resolve conflicts that may arise between federated and local policies, or the intersection of the two, e.g., contradictory rules, following an approach similar to the one proposed by (Hu *et al.* (2012)) and (Ahn *et al.* (2010)).

Attribute-based Encryption. Recently, the introduction of *attribute-based encryption* (ABE) (Sahai and Waters (2005)) has provided confidentiality guarantees over fine-grained data. In ABE, *attributes* are defined over a protected resource, e.g., a text file F, and the set of cryptographic *keys* used to manage F. Later on, an *access structure*, e.g., an access tree, is constructed by leveraging the aforementioned attributes as *leaf* nodes and threshold gates, e.g., and, or, as the interior nodes. A user may be allowed to use a given key k to access a the protected resource, e.g., to decrypt F, if the set of attributes defined in k match the ones defined for F as depicted in the aforementioned access structure. Using ABE, different attributes may be leveraged to produce customized access structures . This way, different users depicting such attributes may be allowed access to protected data with different levels of granularity.

In the context of collaborations between independently-run organizations, ABE may be leveraged in order to provide confidential access to shared encrypted data, allowing for the creation of dedicated inter-organizational groups that share a common predefined set of attributes, following an approach similar to the one presented by (Li *et al.* (2013)). In addition, our proposed attribute derivation features could be also used to allow for members of a certain collaborative group to transform their

locally-issued credentials into the set of attributes depicted within a given ABE setting, such that further access to encrypted data can be successfully achieved. As an example, in Fig. 4.7 the locally-issued credential *Ce* can be potentially transformed into the federated attribute G, which depicts membership to a given collaborative inter-organizational group. A similar setting may allow for locally-issued attributes to be transformed into ABE attributes in an automated way by crafting dedicated AD-Rules that can perform such derivations within the context of each participant organization.

9.2 Assurance-based Models and Construction

9.2.1 Addressing Challenges

Table 9.3 presents a summary of how the approach we have presented in Chapter 6.1 provides a solution to the challenges discussed in Chapter 3.4.1. We believe the different features contained within our proposed assertion-based security models may solve the aforementioned issues as follows: first, our proposed structural and compositional specifications provide a convenient way to precisely describe how an independent implementation of our approach should be constructed, as it was shown in Chapter 6.4. In addition, our model data structures and the specifications for method customizations also provide a well-defined approach for supporting not only customized functionality that is to be added to the basic functionality intended for our approach, but also the integration of existing *in-house* systems, as it has been also discussed in Chapter 6.1. Finally, our client-based invariants and contracts may also support the development and reuse of customized implementations by precisely defining how already-existing components, which may implement security functionality as well, are to be leveraged by other software modules when an integrated system-wide

Table 9.3: Addressing the Challenges Devised for Assurance and Conformance.

	Structural Specifications	**Compositional Specifications**	**Model Data Structures/Methods**	**Client-based Invariants/Contracts**	**Method Customization**
Supporting Independent Implementations	√	√	√	√	√
Supporting Customized Functionality			√	√	√
Integration of *In-House* Systems		√	√		√

implementation is constructed, thus potentially preventing the problems for system integration, e.g., API reutilization, described in the introductory text of Chapter 6.1.

9.2.2 Future Work

Extended Testing Procedure. As mentioned at the end of Chapter 6.4, an extended testing procedure may be required to further explore the capabilities of our proposed assertion-based security model to detect existing security vulnerabilities at the source code level. Following the approach presented by (Rubio-Medrano *et al.* (2013a)), a combination of both *dynamic* techniques, e.g., automated testing, as well as *static* ones, e.g., *theorem proving* (Flanagan *et al.* (2002)), may allow for further vulnerabilities to be detected. In addition, future work may also focus on leveraging our proposed assertion-based models along with well-established techniques for data

flow analysis, such as the ones presented by (Harrold and Rothermel (1994)), who strove to provide testing procedures that can take into account the way data is processed by a certain source code under test. As an example, an in-deep analysis may take into account the different values depicted by attributes in an effort to assess the proper implementation of customized AD-Rules, following the approach depicted in Fig. 6.8.

Refining DBC/JML Specifications. As it was also mentioned during the description of our conformance testing procedure in Chapter 6.4, resolving discrepancies between the observed behavior of our proposed implementation and its corresponding DBC/JML specifications allowed us to engage in a very productive reasoning process, which ultimately resulted in our specifications getting updated to better reflect the desired behavior for modules implementing our FAM approach. With this in mind, we believe our assertion-based models may provide a convenient framework for different actors within the software development process, e.g., architects, designers, coders and testers, to fully engage into discussions on expected security-related runtime behavior, in such a way that our assertion-based models can be further *refined* over time, allowing for more effective descriptions to be created as a result. Future work may then focus on providing a methodology for different actors to leverage, extend, modify and test changes in assertion-based models, such that refinements can be introduced, verified and validated without breaking any existing functionality or introducing newer vulnerabilities not previously identified.

Supporting Other Specification Languages. Finally, we believe our proposed assertion-based security models should be extended to include specification languages other than JML, which is mostly designed to work with the Java programming lan-

guage. As an example, the *object constraint language* (OCL) (Warmer and Kleppe (1998)), has been extensively explored in the literature for providing well-defined descriptions of software structures and runtime behavior. Future work may then strive to provide an OCL framework in such a way that software architects and designers can produce assertion-based security models than can be leveraged for verification and validation in other programming languages, in an approach similar to the one we have described in Chapter 6.4.

9.3 Trust Management Framework

9.3.1 Addressing Challenges

Table 9.4 presents a summary on how the approach presented in Chapter 7 provides a solution to the challenges for trust management as depicted in Chapter 3.4.2. Initially, we believe the different features presented for our trust management framework support the challenge of providing a way to trust participant peers with whom no previous interaction has been recorded. In addition, our approach strives to influence security-related functionality by assigning and calculating trust scores to the process of path discovery, as shown in Chapter 7.2. In addition, the introduction of our XACML-based recommendation policies, as well as their inclusion within our distributed architecture based on DHTs may also allow for peers to share trust scores at finer levels of granularity, which may be extremely convenient as newer and more advance collaboration settings are introduced within peers. In a similar context, federated peers may also leverage our mathematical-based approach for combining trust scores, as introduced in Definitions 1, 2 and 3. Finally, our approach may also allow for peers to leverage the experience collected at the community scale by leveraging our proposed recommendation system, which may also allow them to exchange rec-

Table 9.4: Addressing the Challenges Devised for a Trust Management System.

	Trust Use Cases	Trust Scores	Trust Calculation for AD-Graphs	Recommendation System	Implementation with AD-Rules- DHTs
Trusting Unknown Participants	√	√	√	√	√
Incorporating Security-based Functionality		√	√		√
Supporting Fine-grained Trust Relationships	√	√		√	√
Providing a Mathematical Foundation		√	√		
Leveraging Previous Community-wide Experience		√		√	
Providing a Decentralized Approach			√	√	√

ommendations in a decentralized way by means of the support for the use of the DHT structures as we have detailed in Chapter 5.2 and Chapter 7.3.

9.3.2 Future Work

Supporting Additional Dimensions. As detailed during the description of related work on trust management presented in Chapter 8.4, previous approaches in the literature have considered different dimensions when it comes to calculating a unified trust score in the context of a given community. With this in mind, future work

may focus on allowing for federations to define their own customized dimensions to be included within trust calculations. As an example, a given federation may define resource sharing and implementation of AD-Rules as the most important dimensions to consider, and may also give an specific calculation weights to each of them, in such a way that future trust calculations such as the one presented in Definition 1 can take such dimensions into account. In addition, extra dimensions may also need to be supported within our proposed recommendation policies. As an example, the format for our XACML recommendation policies as shown in Fig. 7.10 can be updated to define different trust dimensions and allow for mapping them to attribute values as well.

Transitive Trust. As shown in Chapter 8.4, our model for recommendations allow for peers to leverage the recommendations issued by other trusted peers within a given federation. As our proposed federations grow with respect to the number of participants, direct *one-to-one* recommendations may be difficult to find in the first place. As a solution, *transitive* recommendations may be introduced allowing for peers to follow a *friends of my friends* approach similar to the model depicted in popular social networks, e.g., Facebook (Facebook Inc. (2015)). This way, federated peers may implement schemes in which different trust scores are assigned according to the number of *friend* hops a given recommendation has been passed upon.

Leveraging History of Transactions. In the approach for trust management we have presented in Chapter 7, participant peers are allowed to maintain their own local scores on the level of trust assigned to other peers within a given federation. However, such scores may evolve over time as a response to a history of shared transactions. As an example, trust scores may need to be updated to reflect an improved perception

on the behavior of a given peer, which whom several successful interactions involving resource sharing may have been carried on in the past. Conversely, the perception of misbehavior may cause the score assigned to a given peer to be decreased as a consequence. With this in mind, future work may focus on allowing for peers to leverage a history of past transactions, as well as a history of the received recommendations involving a certain peer, in order to suggest possible updates to the local trust score that is to be used for future calculations.

Supporting Different Policy Formats. As noticed in the experimental results depicted in Fig. 7.12 and Fig. 7.14, the process of parsing an evaluating our proposed recommendation policies based on the XACML language has a noticeable impact on runtime performance. As such policies become more detailed over time, e.g., providing richer descriptions of the mapping between attributes and trust scores, processing policies containing a large number of XACML rules may become a performance bottleneck, which may ultimately affect the adoption of such a schema in production environments. With this in mind, future work may provide alternative representations of our XACML recommendation policies, following an approach as the one discussed by (Milutinovic (2008)), in such a way that federated peers may be able to effectively parse, evaluate and retrieve trust scores at runtime in a faster way, thus alleviating the performance detriments just described as a result.

Comparison with Existing Trust Languages. As it was also mentioned in Chapter 8.4, several languages tailored for specifying trust relationships have been proposed in the literature. In our approach, we have chosen XACML to serve as a language for communicating trust scores rather than a language for specifying the different trust relationships that may arise in the context of our proposed FAM ap-

proach, e.g., the ones defined in Chapter 7. Future work may then focus on exploring the suitability of such languages for the purposes of federated access management, and evaluating their suitability to serve as a media for the communication of richer trust relationships among participants, so that not only trust scores can be communicated, but also extra information on trust settings that may help peers to decide what other peers are to be trusted within the context of a given federation.

Enhancing Attribute Discovery with Game Theory. Finally, as our proposed federations for FAM increase in the number of participating peers, we also expect the number of AD-Rules provide to increase as a consequence. In such a setting, many different options may be available to peers when constructing attribute provisioning paths by means of the process described in Chapter 5.2.1. As defined in Chapter 7, not all peers in large federations may be fully-trusted, therefore, peers may be presented with several competing options for constructing AD-Graphs for attribute provisioning. With this in mind, future work may implement an approach based on *game theory* (Aumann (1989)), which has been also explored in the context of computer security (Roy *et al.* (2010)) (Manshaei *et al.* (2013)). This way, peers may leverage an extended approach that may better help them to evaluate different options when it comes to constructing attribute paths, which, as mentioned before in this dissertation, has an important influence in the process of mediating access to resources as proposed in our approach.

Chapter 10

CONCLUSIONS

In this dissertation, we have presented an hypothesis based on the use of security-related properties, a.k.a., attributes, for the convenient, flexible, efficient and effective access mediation to resources being shared between independently-run collaborating organizations. For such a purpose, we have provided theoretical as well as experimental evidence that supports the main contributions introduced in Chapter 1: first, we have provided a well-defined theoretical model for attribute-based access mediation based on allowing participants to engage in federations. Also, we have included a precise description on how attributes from different organizations can be transformed (derived) into other ones, referred as federated attributes, which are to be commonly understood and implemented by all participants, providing solid ground for inter-organizational exchange of security-related information.

In addition, we have described an approach that allows for participants to publish, discover, derivate and communicate attributes that will be later used for specifying, evaluating and enforcing policies relevant both in the local and in the federated context. For such a purpose, we have proposed a distributed setting based on the concept of DHT rings and a client-server architecture, thus allowing for newer participants to fully leverage existing *in-house* security-related systems at the same time a timely integration into a given federation implementing our approach is favored.

Also, in order to support correct implementations of our approach, we have introduced our so-called *assertion-based security models*, which leverage software specifications depicting assertions and the DBC programming paradigm to provide rich and concise descriptions of the most important modules and functionalities intended

for our approach. In addition, we have provided evidence showing how our assertion-based models can be leveraged for the purposes of conformance testing as well as for locating and later correcting non-trivial security vulnerabilities that may exists in customized implementations of our approach due to source-code-related pitfalls.

As our proposed federations are expected to accommodate for increasing numbers of participants and collaborative projects, we have also introduced an approach for trust management that allows peers to exchange recommendations on perceived trust scores in a fine-grained detail. For such a purpose, we have also provided a mathematical foundation for combining scores obtained from different peers and have shown how a combined result can be used when performing attribute provisioning and resource sharing duties. Also, we have provided experimental evidence showing that the execution time overhead introduced by trust-related operations remains under manageable means and can be successfully used at runtime when interacting with unknown and possibly untrusted peers.

Finally, for each of the contributions provided in this dissertation, we have presented comparisons detailing the way our proposals solve the problems discussed in Chapter 3. In addition, we have provided a comprehensive review of related work, and have detailed the influences, similarities and differences between our approach and the ones found in the literature, in such a way that the contributions and relevance of our work can be better explained and understood. With the same purpose, we have presented a discussion on the topics addressed in this dissertation and presented several lines of future work, which is intended to address observed shortcomings in our approach at the same time it enhances its suitability for being successfully deployed in practice.

REFERENCES

Aberer, K. and Z. Despotovic, "Managing trust in a peer-2-peer information system", in "Proceedings of the Tenth International Conference on Information and Knowledge Management", CIKM '01, pp. 310–317 (ACM, New York, NY, USA, 2001).

Ahn, G.-J., "Discretionary access control", in "Encyclopedia of Database Systems", pp. 864–866 (Springer, 2009).

Ahn, G.-J., H. Hu, J. Lee and Y. Meng, "Representing and reasoning about web access control policies", in "Proc. of the 34th Computer Software and Applications Conference (COMPSAC)", pp. 137–146 (IEEE, 2010).

Amazon Inc., "Amazon", http://www.amazon.com/ (2016).

Ananthakrishnan, R., J. Bryan, K. Chard, I. Foster, T. Howe, M. Lidman and S. Tuecke, "Globus nexus: An identity, profile, and group management platform for science gateways", in "Proc. of 2013 IEEE International Conference on Cluster Computing (CLUSTER)", pp. 1–3 (2013).

Armbrust, M., A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia, "A view of cloud computing", Commun. ACM **53**, 4, 50–58 (2010).

Aumann, R. J., "Game theory", in "Game Theory", pp. 1–53 (Springer, 1989).

Belhaouari, H., P. Konopacki, R. Laleau and M. Frappier, "A design by contract approach to verify access control policies", in "17th Int'l Conf. on Engineering of Complex Computer Systems (ICECCS)", pp. 263 –272 (2012).

Berners-Lee, T., R. Fielding and L. Masinter, "Rfc 3986: Uniform resource identifier (uri): Generic syntax", The Internet Society (2005).

Blaze, M., J. Feigenbaum and J. Lacy, "Decentralized trust management", in "Proceedings of the 1996 IEEE Symposium on Security and Privacy", SP '96, pp. 164– (IEEE Computer Society, Washington, DC, USA, 1996).

Broeder, D., R. Wartel, B. Jones, P. Kershaw, D. Kelsey, S. Luders, A. Lyall, T. Nyronen and H. J. Weyer, "Federated identity management for research collaborations", Tech. rep., CERN (2012).

Buchegger, S. and J.-Y. Le Boudec, "Performance analysis of the confidant protocol", in "Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing", MobiHoc '02, pp. 226–236 (ACM, New York, NY, USA, 2002).

Burdy, L., Y. Cheon, D. Cok, M. Ernst, J. Kiniry, G.-T. Leavens, K. Leino and E. Poll, "An overview of JML tools and applications", in "Proc. 8th Int'l Workshop on Formal Methods for Industrial Critical Systems (FMICS 03)", pp. 73–89 (2003).

Burdy, L., Y. Cheon, D. R. Cok, M. D. Ernst, J. R. Kiniry, G. T. Leavens, K. R. M. Leino and E. Poll, "An overview of jml tools and applications", Int. J. Softw. Tools Technol. Transf. **7**, 3, 212–232 (2005).

Camenisch, J., A. Lehmann, G. Neven and A. Rial, "Privacy-preserving auditing for attribute-based credentials", in "Proc. of European Symposium on Research in Computer Security (ESORICS)", pp. 109–127 (2014).

Cataño, N. and M. Huisman, "Formal specification of Gemplus's electronic purse case study", in "FME 2002", vol. LNCS 2391, pp. 272–289 (Springer, 2002).

Chadwick, D. W., "Federated identity management", in "Foundations of Security Analysis and Design V", pp. 96–120 (Springer, 2009).

Chadwick, D. W. and G. Inman, "Attribute aggregation in federated identity management", IEEE Computer **42**, 5, 33–40 (2009).

Cheon, Y., "Automated random testing to detect specification-code inconsistencies", in "Proceedings of the 2007 International Conference on Software Engineering Theory and Practice", (2007).

Cheon, Y., G. Leavens, M. Sitaraman and S. Edwards, "Model variables: cleanly supporting abstraction in design by contract: Research articles", Softw. Pract. Exper. **35**, 6, 583–599 (2005).

Cirio, L., I. F. Cruz and R. Tamassia, "A role and attribute based access control system using semantic web technologies", in "Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems - Volume Part II", OTM'07, pp. 1256–1266 (Springer-Verlag, Berlin, Heidelberg, 2007).

Covington, M. J. and M. R. Sastry, "A contextual attribute-based access control model", in "Proceedings of the 2006 international conference on On the Move to Meaningful Internet Systems: AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET - Volume Part II", OTM'06, pp. 1996–2006 (Springer-Verlag, Berlin, Heidelberg, 2006).

Damiani, E., D. C. di Vimercati, S. Paraboschi, P. Samarati and F. Violante, "A reputation-based approach for choosing reliable resources in peer-to-peer networks", in "Proceedings of the 9th ACM Conference on Computer and Communications Security", CCS '02, pp. 207–216 (ACM, New York, NY, USA, 2002).

Damianou, N., N. Dulay, E. Lupu and M. Sloman, "The ponder policy specification language", in "Proceedings of the International Workshop on Policies for Distributed Systems and Networks", POLICY '01, pp. 18–38 (Springer-Verlag, London, UK, UK, 2001).

Danial, Al, "CLOC: Counting Lines of Code", http://cloc.sourceforge.net/ (2015).

Doupé, A., L. Cavedon, C. Kruegel and G. Vigna, "Enemy of the state: A state-aware black-box web vulnerability scanner", in "Proceedings of the 21st USENIX Conference on Security Symposium", Security'12, pp. 26–26 (USENIX Association, Berkeley, CA, USA, 2012).

Dragoni, N., F. Massacci, K. Naliuka and I. Siahaan, "Security-by-contract: Toward a semantics for digital signatures on mobile code", in "Public Key Infrastructure", vol. 4582 of *LNCS*, pp. 297–312 (Springer Berlin, 2007).

D'silva, V., D. Kroening and G. Weissenbacher, "A survey of automated techniques for formal software verification", Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **27**, 7, 1165–1178 (2008).

eBay, "eBay", http://www.ebay.com/ (2016).

Europe's National Research and Education Networks (NRENs), "Geánt Project Home", http://www.geant.net/ (2015).

Facebook Inc., "Facebook Login", https://www.facebook.com/about/login/ (2015).

Ferrari, E., *Access Control in Data Management Systems* (Morgan and Claypool Publishers, 2010).

Firozabadi, B. S. and M. Sergot, "Revocation in the privilege calculus", in "In: Workshop on Formal Aspects of Security and Trust (FAST2003) at FM2003", pp. 39–51 (2003).

Flanagan, C., K. R. M. Leino, M. Lillibridge, G. Nelson, J. B. Saxe and R. Stata, "Extended static checking for Java", in "Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation", PLDI '02, pp. 234–245 (2002).

Forum, O. G., "An Open Global Forum for Advanced Distributed Computing", https://www.ogf.org/ (2015).

Georgiev, M., S. Iyengar, S. Jana, R. Anubhai, D. Boneh and V. Shmatikov, "The most dangerous code in the world: validating ssl certificates in non-browser software", in "Proceedings of the 2012 ACM conference on Computer and communications security", CCS '12, pp. 38–49 (ACM, New York, NY, USA, 2012).

Girardin, F., F. Calabrese, F. D. Fiore, C. Ratti and J. Blat, "Digital footprinting: Uncovering tourists with user-generated content", Pervasive Computing, IEEE **7**, 4, 36–43 (2008).

Guok, C., D. Robertson, M. Thompson, J. Lee, B. Tierney and W. Johnston, "Intra and interdomain circuit provisioning using the oscars reservation system", in "Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on", pp. 1–8 (2006).

Hanemann, A., J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, D. M. Swany, S. Trocha and J. Zurawski, "Perfsonar: A service oriented architecture for multi-domain network monitoring", in "Service-Oriented Computing - ICSOC 2005", vol. 3826 of *Lec. Notes in Computer Science*, pp. 241–254 (Springer, 2005).

Harrold, M. J. and G. Rothermel, "Performing data flow testing on classes", in "ACM SIGSOFT Software Engineering Notes", vol. 19, pp. 154–163 (ACM, 1994).

Hendrikx, F. and K. Bubendorfer, "Malleable access rights to establish and enable scientific collaboration", in "eScience (eScience), 2013 IEEE 9th International Conference on", pp. 334–341 (2013).

Hendrikx, F., K. Bubendorfer and R. Chard, "Reputation systems: A survey and taxonomy", Journal of Parallel and Distributed Computing **75**, 184 – 197 (2015).

Hoare, C. A. R., "An axiomatic basis for computer programming", Communications of the ACM **12**, 10, 576–580 (1969).

Hu, H. and G.-J. Ahn, "Enabling verification and conformance testing for access control model", in "Proc. of the 13th ACM Symp. on Access Control Models and Technologies", pp. 195–204 (2008).

Hu, H., G.-J. Ahn and K. Kulkarni, "Detecting and resolving firewall policy anomalies", IEEE Transactions on Dependable and Secure Computing **9**, 3, 318–331 (2012).

Hu, V. C., D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller and K. Scarfone, "Guide to attribute based access control (abac) definition and considerations", NIST Special Publication **800**, 162 (2014).

Iwaihara, M., K. Murakami, G.-J. Ahn and M. Yoshikawa, "Risk evaluation for personal identity management based on privacy attribute ontology", in "Conceptual Modeling-ER 2008", pp. 183–198 (Springer, 2008).

Jia, Y. and M. Harman, "An analysis and survey of the development of mutation testing", IEEE Transactions on Software Engineering **37**, 5, 649 –678 (2011).

Jin, J. and G.-J. Ahn, "Authorization framework for resource sharing in grid environments", Grid and Distributed Computing **63**, 148–155 (2009).

Jin, X., R. Krishnan and R. Sandhu, "A unified attribute-based access control model covering dac, mac and rbac", in "Proceedings of the 26th Annual IFIP WG 11.3 conference on Data and Applications Security and Privacy", DBSec'12, pp. 41–55 (Springer-Verlag, Berlin, Heidelberg, 2012).

Jones, M. and D. Hardt, "The oauth 2.0 authorization framework: Bearer token usage", Tech. rep., RFC 6750, October (2012).

Kaffille, S. and K. Loesing, "Open Chord", http://sourceforge.net/projects/open-chord/ (2015).

Kagal, L., T. Finin and A. Joshi, "A policy language for a pervasive computing environment", in "Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks", POLICY '03, pp. 63– (Washington, DC, USA, 2003).

Kamvar, S. D., M. T. Schlosser and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks", in "Proceedings of the 12th International Conference on World Wide Web", WWW '03, pp. 640–651 (ACM, New York, NY, USA, 2003).

Klingenstein, N., "Attribute aggregation and federated identity", in "Proceedings of the 2007 International Symposium on Applications and the Internet Workshops (SAINT)", pp. 26–26 (2007).

Lang, B., I. Foster, F. Siebenlist, R. Ananthakrishnan and T. Freeman, "A flexible attribute based access control method for grid computing", Journal of Grid Computing **7**, 2, 169–180 (2009).

Lee, C. Y., "An algorithm for path connections and its applications", IRE Transactions on Electronic Computers **EC-10**, 3, 346–365 (1961).

Li, B., Z. Wang and D. Huang, "An efficient and anonymous attribute-based group setup scheme", in "2013 IEEE Global Communications Conference (GLOBECOM)", pp. 861–866 (2013).

Li, N., J. Mitchell and W. Winsborough, "Design of a role-based trust-management framework", in "Proc. of the 2002 IEEE Symposium on Security and Privacy", pp. 114–130 (2002).

Lloyd, J. and J. Jürjens, "Security analysis of a biometric authentication system using UMLsec and JML", in "MoDELS", vol. 5795 of *Lecture Notes in Computer Science*, pp. 77–91 (Springer, 2009).

Malik, Z. and A. Bouguettaya, "Rateweb: Reputation assessment for trust establishment among web services", The VLDB Journal—The International Journal on Very Large Data Bases **18**, 4, 885–911 (2009).

Manshaei, M. H., Q. Zhu, T. Alpcan, T. Bacşar and J.-P. Hubaux, "Game theory meets network security and privacy", ACM Computing Surveys (CSUR) **45**, 3, 25 (2013).

Marouf, S., M. Shehab, A. Squicciarini and S. Sundareswaran, "Adaptive reordering and clustering-based framework for efficient xacml policy evaluation", IEEE Trans. Serv. Comput. **4**, 4, 300–313 (2011).

McGraw, G., *Software Security: Building Security In* (Addison-Wesley, 2006).

Melnik, M. I. and J. Alm, "Does a seller's ecommerce reputation matter? evidence from ebay auctions", The Journal of Industrial Economics **50**, 3, 337–349 (2002).

Milutinovic, S., "The need for the use of xacml access control policy in a distributed ehr and some performance considerations", Medical and Care Compunetics **5**, 137, 346 (2008).

Monsanto, C., J. Reich, N. Foster, J. Rexford and D. Walker, "Composing software-defined networks", in "Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation", nsdi'13, pp. 1–14 (USENIX Association, Berkeley, CA, USA, 2013).

Morgan, R. L., S. Cantor, S. Carmody, W. Hoehn and K. Klingenstein, "Federated Security: The Shibboleth Approach", EDUCAUSE Quarterly **27**, 4, 12–17 (2004).

Mustafa, T. and K. Sohr, "Understanding the implemented access control policy of android system services with slicing and extended static checking", International Journal of Information Security pp. 1–20 (2014).

Needham, R. M., "Denial of service", in "Proceedings of the ACM Conference on Computer and Communications Security (CCS)", pp. 151–153 (ACM, 1993).

Neuman, B. and T. Ts'o, "Kerberos: an authentication service for computer networks", Communications Magazine, IEEE **32**, 9, 33–38 (1994).

Nordic Council of Ministers, "Nordic Infrastructure for Research & Education (NORDUnet)", https://www.nordu.net/ (2015).

OASIS Standard, "eXtensible Access Control Markup Language (XACML) Version 3.0. (2013, January 22)", http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html (2013).

Open Grid Forum, "Network Services Interface (NSI)", https://redmine.ogf.org/projects/nsi-wg (2015).

Oracle Corporation, "MySQL Community Edition Server", https://dev.mysql.com/downloads/mysql/ (2015).

Paci, F., R. Ferrini, A. Musci, K. Steuer and E. Bertino, "An interoperable approach to multifactor identity verification", IEEE Computer **42**, 5, 50–57 (2009).

Priebe, T., W. Dobmeier and N. Kamprath, "Supporting attribute-based access control with ontologies", in "Proceedings of the First International Conference on Availability, Reliability and Security", ARES '06, pp. 465–472 (IEEE Computer Society, Washington, DC, USA, 2006).

Rackspace, I., "OpenStack Cloud Operating System", https://www.openstack.org/ (2015).

Recordon, D. and D. Reed, "Openid 2.0: A platform for user-centric identity management", in "Proceedings of the Second ACM Workshop on Digital Identity Management", DIM '06, pp. 11–16 (ACM, New York, NY, USA, 2006).

Rentsch, T., "Object oriented programming", ACM Sigplan Notices **17**, 9, 51–57 (1982).

Resnick, P., K. Kuwabara, R. Zeckhauser and E. Friedman, "Reputation systems", Commun. ACM **43**, 12, 45–48 (2000).

Roberts, G., T. Kudoh, C. Guok and J. MacAuley, "NSI Policy", https://redmine.ogf.org/dmsf/nsi-wg (2015).

Rosenblum, D. S., "A practical approach to programming with assertions", IEEE Trans. Softw. Eng. **21**, 1, 19–31 (1995).

Roy, S., C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya and Q. Wu, "A survey of game theory as applied to network security", in "System Sciences (HICSS), 2010 43rd Hawaii International Conference on", pp. 1–10 (IEEE, 2010).

Rubio-Medrano, C. E., G.-J. Ahn and K. Sohr, "Verifying access control properties with design by contract: Framework and lessons learned", in "Proceedings of the IEEE International Computer Software and Applications Conference (COMPSAC)", pp. 21–26 (2013a).

Rubio-Medrano, C. E., C. D'Souza and G.-J. Ahn, "Supporting secure collaborations with attribute-based access control", in "Proceedings of the IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)", pp. 525–530 (IEEE, 2013b).

Ruohomaa, S. and L. Kutvonen, *Trust Management Survey*, pp. 77–92 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2005).

Sabater, J. and C. Sierra, "Social regret, a reputation model based on social relations", SIGecom Exch. **3**, 1, 44–56 (2001).

Sabelfeld, A. and A. C. Myers, "Language-based information-flow security", IEEE J. Selected Areas in Communications **21**, 1, 5–19 (2003).

Sahai, A. and B. Waters, "Fuzzy Identity-Based Encryption", Advances in Cryptology – EUROCRYPT 2005 pp. 457–473 (2005).

Sandhu, R. S., E. J. Coyne, H. L. Feinstein and C. E. Youman, "Role-based access control models", IEEE Computer **29**, 2, 38–47 (1996).

Shen, H., "A semantic-aware attribute-based access control model for web services", in "Proc. of the 9th Int. Conference on Algorithms and Architectures for Parallel Processing", ICA3PP '09, pp. 693–703 (Springer-Verlag, 2009).

Smari, W. W., J. Zhu and P. Clemente, "Trust and privacy in attribute based access control for collaboration environments", in "Proc. of the 11th International Conference on Information Integration and Web-based Applications & Services", iiWAS '09, pp. 49–55 (ACM, New York, NY, USA, 2009).

Stackoverflow, "Stackoverflow", http://www.stackoverflow.com/ (2016).

Stoica, I., R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications", in "Proc. of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications", pp. 149–160 (ACM, New York, NY, USA, 2001).

Sun, W., X. Yuan, J. Wang, D. Han and C. Zhang, "Quality of service networking for smart grid distribution monitoring", in "Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on", pp. 373–378 (IEEE, 2010).

Sun Microsystems, Inc., "Sun's XACML Implementation", http://sunxacml.sourceforge.net/ (2015).

Tarjan, R., "Depth first search and linear graph algorithms", SIAM Journal on Computing (1972).

The Apache Software Foundation, "Apache CXF", http://cxf.apache.org/ (2015).

Tian, C. and B. Yang, "R2 trust, a reputation and risk based trust management framework for large-scale, fully decentralized overlay networks", Future Generation Comp. Syst. **27**, 1135–1141 (2011).

Trompert, H. and J. MacAuley, "NSI Authentication and Authorization", https://redmine.ogf.org/dmsf/nsi-wg (2015).

Upadhyaya, S., "Mandatory access control", in "Encyclopedia of Cryptography and Security (2nd Ed.)", edited by H. C. A. van Tilborg and S. Jajodia, pp. 756–758 (Springer, 2011).

US Department of Energy, "Energy Sciences Network (ESnet)", http://www.es.net/ (2015).

Uszok, A., J. M. Bradshaw and R. Jeffers, *KAoS: A Policy and Domain Services Framework for Grid Computing and Semantic Web Services*, pp. 16–26 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2004).

Wang, L., D. Wijesekera and S. Jajodia, "A logic-based framework for attribute based access control", in "Proceedings of the 2004 ACM workshop on Formal methods in sec. eng.", FMSE '04, pp. 45–55 (ACM, New York, NY, USA, 2004).

Warmer, J. B. and A. G. Kleppe, "The object constraint language: Precise modeling with uml (addison-wesley object technology series)", (1998).

Wei, Y., C. Shi and W. Shao, "An attribute and role based access control model for service-oriented environment", in "Control and Decision Conference (CCDC), 2010 Chinese", pp. 4451–4455 (2010).

Xiong, L. and L. Liu, "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities", IEEE Trans. on Knowl. and Data Eng. **16**, 7, 843–857 (2004).

Yuan, E. and J. Tong, "Attributed based access control (abac) for web services", in "Proceedings of the IEEE International Conference on Web Services", ICWS '05, pp. 561–569 (IEEE Computer Society, Washington, DC, USA, 2005).

Zhang, X., Y. Li and D. Nalla, "An attribute-based access matrix model", in "Proceedings of the 2005 ACM symposium on Applied computing", SAC '05, pp. 359–363 (ACM, New York, NY, USA, 2005).

Zhu, J. and W. Smari, "Attribute based access control and security for collaboration environments", in "Aerospace and Electronics Conference, 2008. NAECON 2008. IEEE National", pp. 31–35 (2008).

VITA

Carlos E. Rubio Medrano was born in Chihuahua City, Mexico, in 1982. His research interests lay on the intersection of computer security, formal specifications, and the verification and validation of software modules. He received a BS degree from the Instituto Tecnológico de Chihuahua II, Mexico, in 2005, and the MS degree from the University of Texas at El Paso in 2008. He spent a couple of years in the PhD Program in Computer Science at the University of Arizona, Tucson, from 2008-2010, before joining Arizona State University in 2012, where he was affiliated to the Laboratory of Secure Engineering for Future Computing (SEFCOM) and the Center for Cybersecurity and Digital Forensics (CDF). He is an avid reader and a huge fan of the contemporary fantasy novel series *A Song of Ice and Fire* and its corresponding TV adaptation *Game of Thrones*. In late 2016, an idea of his inspiration was featured by *PhD Comics*, a popular comic strip on academic matters.

Figure 10.1: Comic Strip: A New Way to Review Academic Papers.