Supporting Secure Collaborations with Attribute-based Access Control†

Carlos E. Rubio-Medrano, Clinton D'Souza and Gail-Joon Ahn
Ira A. Fulton Schools of Engineering
Arizona State University
Tempe, Arizona, USA, 85282
{crubiome,cvdsouza,gahn}@asu.edu

***Abstract*—Attribute-based access control (ABAC) has been regarded in recent years as an effective way for providing security guarantees in collaboration environments, due to its alleged flexibility and efficiency for meeting the access control requirements of *heterogeneous* organizations. Despite the growing interest in ABAC, there still need consensus on a reference model that comprehensively describes all necessary components and functions, in such a way non-trivial security properties can be effectively taken into account. In order to overcome this limitation, we propose an abstract model that includes a precise definition of attributes and relevant core components. In addition, we introduce the notion of *security tokens* that serve as a layer of association between attributes and access rights. We also validate our results by presenting both a case study and a comparison with existing approaches that have been previously proposed in the literature.**

***Keywords*—attribute-based, access control, collaboration**

I. Introduction

Access control has been recognized as an effective way to provide security assurance of *mission-critical* software systems, by restricting access to the sensitive resources that may be consumed during execution, e.g., confidential files, communication channels, etc. For collaborative systems, the proper enforcement of access control rules becomes crucial, as the overall safety of the collaborative environment may rely on an access control that can effectively allow users (human agents, automated processes) from different organizations to access the resources required for collaboration, while still preventing unauthorized access to resources. In recent years, *attribute-based access control* (ABAC) [1], [2] has obtained considerable attention as a flexible and effective means for managing the crucial and non-trivial task in specifying access control requirements of collaborative systems, as the use of attribute-based constraints may better accommodate the specification of inter-domain access control policies among diverse organizations, which in turn may seamlessly implement their own *heterogeneous* access control models and mechanisms [3]. Despite its alleged benefits, there still exists no reference model of ABAC in the literature, in such a way relevant security properties in software systems can be properly verified. In order to alleviate this challenge, this paper presents an abstract model for ABAC intended to provide a basis for the rigorous analysis of access control configurations by means of ABAC policies. We first start by describing a sample use case involving attribute-based constraints in the collaboration domain, in an effort to locate some common patterns that better describe the main features desired for our proposed model. Next, we aim to define the main ABAC component: *attributes*, which we use extensively throughout the rest of our work. Based on this, we provide the main features of our proposed model, which introduce the concept of *security tokens*. These first-class objects are intended to provide an abstract representation of the set of relevant security states devised for software systems, in such a way access rights can be associated with states to better meet access control requirements. Security tokens establish an association between the aforementioned attributes and permissions, which are commonly described in the literature as a well-defined representation of access rights, relating protected resources with an operation that can be performed upon them. We describe how attributes are associated with access control entities, e.g. users, protected resources, etc. and how they can be turned into security tokens. We provide an empirical evaluation of our model by describing an illustrative case study in the collaboration domain and by presenting a comparison with existing approaches in the literature, in such a way the novelty and the benefits introduced by our approach are articulated. The rest of this paper is organized as follows: we introduce the problem being addressed in Section II, which has influenced our proposed ABAC model described in Section III. In order to illustrate the applicability of our approach, Section IV presents an empirical evaluation of our approach and exhibits a set of interesting access control challenges that are efficiently addressed by means of our ABAC model. We then discuss the future direction of our work and conclude the paper in Section V.

II. Problem Definition

In ABAC, a given access control request, e.g. reading a confidential data file, is granted upon the satisfaction of constraints involving some security-relevant properties, also known as *attributes*, that are exhibited by the access control *entities* involved in the request. Commonly, such access control entities include *actors*, i.e., human agents or computer processes running on behalf of them; *targets*, i.e., protected resources such as data files or network ports, and any applicable *context*, i.e., the running environment where a given software system is executed, such as an operative system or a cloud setting. In this section, we introduce a simple use case example in the collaboration domain that involves the specification of access control constraints based on attributes obtained from

†All correspondences should be addressed to: Dr. Gail-Joon Ahn at gahn@asu.edu.

COLLABORATECOM 2013, October 20-23, Austin, United States

DOI 10.4108/icst.collaboratecom.2013.254168

different entities, in an effort to properly identify how attributes are used for defining constraints and how constraints are in turn related to access rights (permissions). For illustrative purposes, consider a setting when *actors* may request access to a *shared* data file. Such an access right is represented by a permission named *readFilePermission*, which is only granted if the requesting actor happens to be the *owner* of the file, or if the file is labeled for sharing purposes (by naming the file as "*shared.txt*"). Such a constraint depicts a collaboration setting when an access to a shared resource is granted to a collaborative actor only for working hours. A corresponding use case is depicted in Fig. 1: the constraint C_1 restricts access to a given file only if its *name* attribute has a value equal to the "*shared.txt*" and the *time* attribute exhibited by the context environment fits within a certain range. Moreover, the access control constraint C_2 restricts access to files only when the *name* attribute presented by the requesting actor is the same as the *ownername* attribute defined for the requested file. This use case depicts a scenario when attributes are retrieved from all the involved access control entities, and several constraints are defined in order for the *readFilePermission* to be granted. Based on this sample case, some noticeable patterns emerge: first, different attributes exhibited by different access control entities may be used to define constraints, and not all of the entities involved may need to provide attributes in order for an access control decision to be made, as noticed in the C_1 constraint, where the entity requesting access is required to show no attributes for the constraint guarding the requested permission. Second, the same set of attributes exhibited by a given access control entity may be used to define more than one constraint, as shown in C_1 and C_2, where attributes from the *target* are used to define two different constraints. Third, constraints may be in turn composed of different *sub-constraints* and may be evaluated simultaneously while making an access control decision (C_1). Finally, many different constraints, which may be constructed upon different attributes from different access control entities, may be used to guard access to the same permission. The patterns discussed above provide an insight on how flexible an scheme based on attributes and constraints should be for modeling access control requirements: as many different options are available, policy designers may choose the ones that better fit the needs devised for a given collaboration setting. However, at the cost of flexibility, we must also consider such diversity may also cause unexpected challenges when analyzing some non-trivial security properties of a given ABAC setting. As an example, consider the case of performing a *risk analysis* [4] on the ABAC setting depicted in Fig. 1: in order to determine under which situations *readFilePermission* may be granted. Such an analysis may involve the analysis of each constraint guarding the *readFilePermission*, and attributes on each entity, including the range of possible values such attributes may take and the number of possible combinations that may be obtained from them. Such a process gets complicated by the fact that no standardized definition of both attributes and constraints exists in the literature, nor a standardized policy language for a reference ABAC model exists. These problems may turn the security analysis process cumbersome in real-life systems where diverse constraints may be in place for guarding a given permission.

III. Model Definition

A. General Description

In order to support access control patterns depicted in Section II, we first define the main components of our proposed model:

- *actors* are users (i.e. human agents) or subjects (i.e. computer processes) acting on behalf of users;
- *targets* are the protected resources within a software system;
- *requests* are initiated by actors when access to a specific target is needed; and
- *context* is the running (executing) environment, e.g. operative system, supporting platform, etc., where a given action is issued and/or served.

Besides the core ABAC component *attributes*, we also introduce the concept of *security tokens*: first-class objects that are intended to provide an abstract representation of a set of security states that may be relevant to the domain-specific properties in collaborative systems. Moreover, these security tokens are helpful to represent the attributes exhibited by access control entities. Attributes are related to security tokens through *token provisioning functions* (TP-Functions), which map each attribute to a corresponding security token, and are intended to model the constraints shown in Section II. They also produce security tokens as a result and allow to define the relationship between permissions and attributes. It must be noticed that the precise definition of such TP-Functions falls in the scope of the collaborative system domain. Fig. 2 shows a visual representation of our proposed model, which leverages a similar representation shown in [5]. In the diagram, the sets of either access control entities or elements (e.g. attributes) are modeled as circles connected by either unidirectional or bidirectional arrows, which can also be single-headed or double-headed. Unidirectional arrows depict functions, whereas bidirectional ones are used for relations. For instance, attributes are related to its corresponding access control entity by means of the *attribute assignment* (AA) relation. Moreover, double-headed arrows depict multiplicity of appearances of members of sets (denoted by circles) in the relation depicted by the arrow, whereas single-headed ones may depict a single appearance. As an example, the AA relation depicts an access control entity is related to several attributes, whereas an attribute may be related only to a single access control entity. Attributes are also related to security tokens through our proposed TP-Functions, which are depicted as unidirectional arrows in the diagram. Moreover, TP-Functions also relates security tokens with other security tokens. Following this approach, a security token that is produced by means of a TP-Function is *provisioned*. Security tokens are related to access rights (permissions) by means of the *permission assignment* (PA) relation. Permissions are in turn depicted as a combination of a protected source (target) and an operation that can be performed on it. Following the convention for double-headed arrows introduced earlier, a security token may be related to one or more permissions, and a given permission may be related to one or more security tokens. Finally, it must be noticed that our model is intended to be *independent* of any supporting technology or methodology other than the concepts we have enlisted in this section.

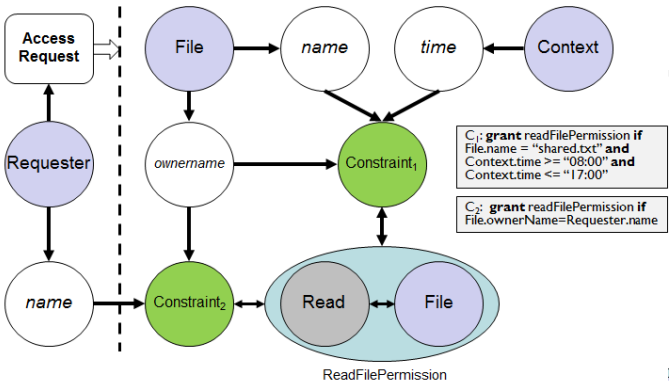

Fig. 1. A combined use case depicting ABAC in the collaboration domain.

B. Definition of Attributes

As described previously, attributes are defined to be *security-relevant* properties that are exhibited by access control entities, namely, actors, targets, policies and any applicable context. Their physical nature and the way those attributes are collected from the access control entities remains dependent on the collaborative application domain. From the use case and patterns discussed in Section II, attributes are expected to have at least the following inner components: an *identifier* (*id* for short), which is later used for defining constraints on them, e.g. the attribute *name* defined for a given file (target); a *value*, which is used when evaluating constraints, e.g. "*shared.txt*"; and a data *type*, which restricts the nature and the range of the value defined for the attribute, e.g. the data type *String*. In addition, for brevity, we also introduce the concept of *attribute families*, which are the sets of attributes where all members share the same *type* and *id* components. With this in mind, we proceed with the following definition:

Definition 1: An *attribute* is a 3-tuple of the form <*type*, *id*, *value*>, where

- *type* is a well-defined data type relevant to the collaborative application domain, e.g. String, Integer, Date, Boolean, etc.
- *id* is a unique identifier, which is required to be non-empty for the purpose of an access control decision. The format of the *id* component, as well as its associated semantics, is based on the scope of the collaborative application domain.
- *value* is a data value of the attribute in the nature and range defined by *type*.

Illustrative examples of attributes are as follows: <String, *file.name*, "*shared.txt*">, <String, *actor.name*, "*Carlos*">, <Integer, *actor.age*, 100>, <Date, *system.date*, "*10-10-3*">, <Boolean, *file.isOpen*, true>, etc.

Definition 2: An *attribute family* is a set of attributes where all elements share the same *type* and *id* components.

The number of possible elements of an attribute family is bounded by the range of values defined by the *type* component. In the rest of this paper, we use the term *fam-a*(*id*) to refer to a family of attributes identified by a given *id* component.

C. Definition of Security Tokens

As described previously, we aim to provide a description of the main features of ABAC by introducing the concept of *security tokens*. Informally, such tokens are obtained by *processing* the attributes from access control entities through the aforementioned TP-Functions, and are intended to provide an abstract representation of the security states within a software system (e.g. a collaborative application), on which access rights can be granted *safely*. As an example, TP-Functions may provide functionality intended to validate a given attribute, by inspecting its value component and producing a proper security token as a result. Thus, a validated attribute may put the system in a secure state, and access rights can be safely granted as a result. As described in Section III-A, permissions can be assigned to security tokens, which then serve as a layer of association between attributes and permissions. Such a layer helps identify the attributes that may be ultimately involved in granting a given permission, as well as the set of constraints represented by TP-Functions that may be involved in such process. Moreover, our model also allows for TP-Functions to take security tokens as an input, or may also take both attributes and security tokens as an input altogether to produce security tokens as a result, as depicted in Fig. 2. We define *security token* as follows:

Definition 3: A *security token* is a tuple of the form <*id*, *value*>, where *id* is a unique identifier, whose format as well as associated semantics remain in the scope of the collaborative application domain, and *value* is the devised value of the token, if any, which may also be in the scope of the application domain, and may not necessarily *match* a given value depicted by the attribute (or set of attributes) the token is originated from.

Note that our definition of security tokens does not include the concept of data type, which is included in the definition of attributes, in order to allow for an enhanced flexibility in the definition of the range of values. Examples of security tokens include: <*jobPosition*, manager>, <*jobPosition*, employee>, <*isUnderage*, true>, <*isSuperUser*, false>, etc. With attributes, security tokens are grouped into token families, which have a similar definition to the ones defined for attributes. A token family groups a set of security tokens that share the same *id* component as follows:

Definition 4: A *security token family* is a set of security

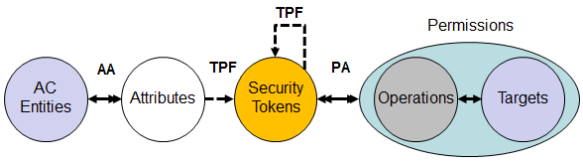

Fig. 2. Graphical depiction of a model based on *security tokens* for ABAC.

tokens where all members share the same *id* component.

As with attribute families, we denote a security family identified by *id* as *fam-t*(*id*). Examples of security token families include: *fam-t*(*jobPosition*) = {<*jobPosition*, manager>, <*jobPosition*, employee>} and *fam-t*(*isUnderage*) = {<*isUnderage*, true>, <*isUnderAge*, false>}.

D. Definition of Token-Provisioning Functions

As introduced in Section III-A, our TP-Functions represent a core component of our model that is intended to provide a conceptual foundation to the constraints depicted in the use case discussed in Section II. TP-Functions are expected to provide a strong mapping between a given attribute defined in an attribute family and a single security token defined in a security token family. On the other hand, not all tokens in a given security token family are expected to be originated from an attribute through a given TP-Function. Fig. 3 shows a representation of a pair of TP-Functions mapping attributes from the attribute family identified by *fam-a*(*a*) to security tokens in the token families *fam-t*(*t*) (TPF_1) and *fam-t*(*t'*) (TPF_2). Fig. 3 illustrates two interesting properties of our proposed TP-Functions: first, TP-Functions are said to be *non-injective*[1], as two or more elements from a TP-Function input set (domain) may be mapped to the same element in the output set (codomain). As an example, consider the case of attributes a_1 and a_2, from the attribute family *fam-a*(*a*), shown in Fig. 3, which are both mapped to the same element t_1 in *fam-t*(*t*) by TPF_1. Second, TP-Functions are said to be *non-surjective*[2], as none of elements in the output (codomain) set are required to have a corresponding element in the input (domain) set. As an example, Fig. 3 shows the security token t_3 belonging to the *fam-t*(*t*) family has no originating attribute from *fam-a*(*a*) by TPF_1.

Definition 5: A *token-provisioning function* (TP-Function) is a *non-bijective* as well as *non-surjective*, mapping sets of attribute families and security token families to security token families.

The combination of families of attributes and security tokens as an input to TP-Functions can be achieved by taking the cartesian product of all the families involved. Moreover, as depicted in Section III-A, security tokens are associated with access rights (permissions) by means of the PA relation. As shown in Fig. 3, TP-Functions can be chained together to produce a graph-like structure showing how attributes and security tokens are used to produce security tokens. Such a structure is defined as follows:

[1]A function f: A → B is said to be *injective* or *one-to-one*, if ∀ a, a' ∈ A, $f(a) \neq f(a')$.

[2]A function f: A → B is said to be *surjective* or *onto*, if ∀ b ∈ B, ∃ a ∈ A, $f(a) = b$.

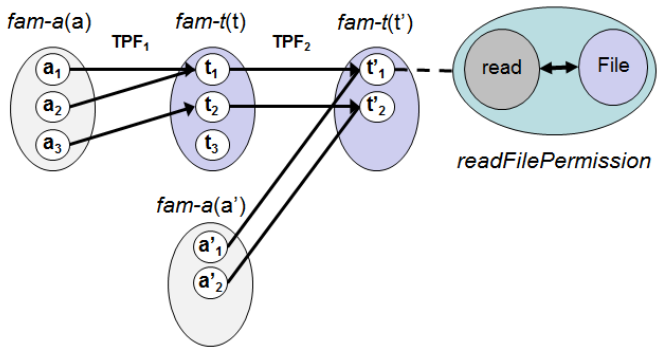

Fig. 3. An example of TP-Functions, token families and an associated permission.

Definition 6: A *token-provisioning graph* (TP-Graph) is a directed, *weakly* connected, and possibly cyclic graph, whose *vertices* (*nodes*) represent either families of attributes or families of security tokens, and its *edges* (*arcs*) represents token-provisioning functions (TP-Functions).

TP-Graphs are *directed*, since TP-Functions represent unidirectional edges due to their nature. Moreover, TP-Graphs are also *weakly* connected, as there is no requirement for all nodes (families) to be connected to each other. Finally, TP-Graphs are also possibly *cyclic*. Leveraging this definition, security tokens are obtained by *traversing* a given TP-Graph. Fig. 3 shows an example of a TP-Graph that has been annotated with a permission, which is in turn associated with a given security token t_1. The process of granting the depicted *readFilePermission* is summarized as follows: at runtime, the access control entities involved in the request must exhibit attributes a_1 and a_2. An attribute a_1 is turn into the security token t_1 by means of the TPF_1 function. Token t_1, in combination with attribute a_1 (from family *fam-a*(*a*)) is also turned into the security token t'_1, which is related to the aforementioned *readFilePermission* by means of an entry in the PA relation. As explained in Section III-A, a permission is granted if a requesting actor is able to *provision* a token that is associated with the PA relation. Following Definition 6, security tokens are *provisioned* by traversing a TP-Graph. Besides providing a stronger definition of the way security tokens are generated within our model, TP-Graphs allow for the development of security analysis techniques based on graph theory. Finally, the process of applying TP-Functions to attributes/security tokens (by traversing a given TP-Graph) is named as *token provisioning*. We also call the set of security tokens that can be potentially provisioned from a TP-Graph as the *token repository*.

IV. Evaluation

A. Case Study

In order to illustrate the access control model proposed in Section III, we performed a case study with a *document management system* (DMS). Such an application is intended for end-users to store and share document files, e.g. text files, pictures, etc., while still enforcing a set of access control rules to protect the confidentiality and integrity of the documents managed by the system. Users are allowed to upload documents into the system, in such a way documents are *owned*

by the user who uploaded them in the first place. Users are also allowed to *share* documents they own with other users by specifying the following access modes: *read* (R), *write* (W) and *execute* (X). As an example, a user u_1 may share a document d with another user u_2 by granting him/her the *read* access mode. Therefore, u_2 is allowed to inspect d, but not to change its contents. In addition, three different categories of end-users are defined, namely: *Employee*, *Departmental Manager* and *CEO*. Users with the *Employee* category are grouped in departments led by one or more users holding the *Department Manager* category, which in turn oversee users with the *CEO* category. The access control requirements for each category are as follows:

- Users with the *Employee* category share *full* access privileges (RWX) only with their corresponding *Department Manager* and the *CEO* by default.
- Users with the *Department Manager* category share *full* access privileges with the *CEO* as well, but not with *Employee* .
- Users with the *CEO* category share no access privileges with any other users by default.

Finally, the access control privileges defined for document *owners* remain intact despite the job category defined for a given user. This case study shows a couple of interesting features in terms of access control: first, it clearly depicts an approach involving access control constraints based on attributes. Second, derived from the aforementioned set of access control requirements based on organizational jobs, an access control model depicting the well-known *role-based access control* (RBAC) [5] immediately comes to mind. However, as depicted in [6], applying attributes (along with their corresponding access control constraints) to an RBAC setting is not a straightforward task. With this in mind, our case study presents an interesting alternative by leveraging an ABAC setting modeled with the approach we have described in previous sections.

Fig. 4 shows a representation of a model configuration depicting our case study. The set of access control entities includes the set of users of the DMS (actors), as well as all the documents that are uploaded into the system (targets). The operations include the ones described earlier and the set of permissions is specified by relating the operations with the set of documents uploaded to the DMS. Attributes can be grouped into the following families: *userID*, *jobCategory* and *deptID*, which are related to the set of users, and *docID*, *ownerID* and *sharingModes*, which are related to documents. Families of security tokens include *userID*, *docID*, *deptID*, *jobCategory*, *docID*, *docID*, *ownerID* and *ownerDeptID*, which are obtained from processing the corresponding attributes by means of the TP-Functions f_1, f_2, f_3, f_4, f_5 and f_7 respectively. In addition, the families of security tokens *shared-Owner*, *shared-Dept-Manager*, *shared-CEO* and *shared* represent the set of security states in the DMS where permissions can be safely granted, and are produced by functions f_6, f_8, f_9 and f_{10}, which derive the access control constraints for our case study. As an example, the TP-Function identified by f_9 compares the security tokens *userID* and *ownerID* then returns the *shared-Owner* if their *value* components match. Otherwise, an *empty* security token is returned. If the *shared-Owner* is provisioned

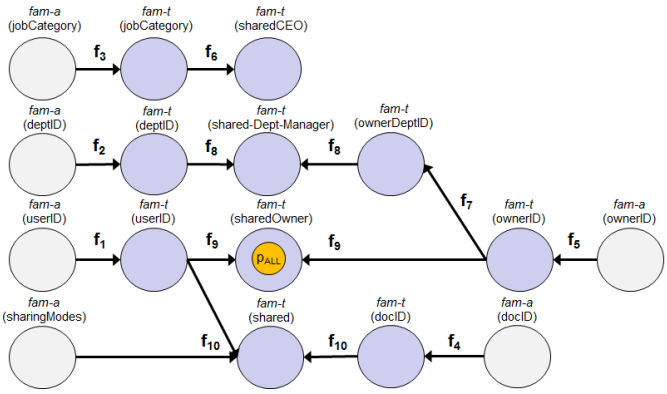

Fig. 4. A TP-Graph created from the TP-Functions depicted in our case study.

by a requesting actor, the associated permission p_{ALL}, which allows complete access to a document within the DMS, is granted as a result. A similar convention is applied in the definition of all other TP-Functions within our case study.

B. Comparison with Existing Approaches

This section describes a brief comparison of our approach with existing approaches in the literature. We first articulate a comparison criteria based on the characteristics of ABAC: first, as we aim to provide a definition of the main ABAC component *attributes*, we check how this important component has been defined and considered in other existing approaches. Next, we explore the support offered by existing approaches in specifying access control constraints based on the concept of attributes, which we have described in Sections III-A and III-D. Following with the same idea, we also compare our approach with existing ones with respect to the definition of the overall access control model. Next, we explore the support for the notion of *security state* (Sections III-A and III-C) offered by the literature, in such a way a relationship between attributes and access rights are properly identified. Finally, as our approach is intended to be *independent* in terms of the supporting methodology, as described in Section III-A, we determine whether such a feature has been provided by other approaches. Using this comparison criteria, we examine existing approaches in the literature: one of the initial attempts in defining ABAC was proposed by Wang et al. [1], 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. Another approach was presented by Yuan et al. [2], who explored the use of ABAC for the specification of access control policies for web services. Their approach, while providing no formal definition of ABAC, depicts useful use cases that have influenced the ones described in Section II. Moreover, Priebe et al. [7] 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. [8], who provide a model definition including attribute-based constraints. The work of Zhu and

TABLE I. A COMPARISON WITH EXISTING APPROACHES FOR ABAC.

Paper	Attributes	Constraints	Model Def.	Sec. State	Independence
[1]	√	√	√	×	×
[2]	×	×	×	×	×
[13]	√	×	√	√	√
[11]	√	×	×	×	×
[7]	×	×	√	×	×
[8]	×	√	√	×	×
[10]	√	√	×	×	×
[9]	√	×	×	×	×
[3]	√	√	×	×	×
[14]	×	×	×	√	×
[15]	×	×	√	×	×
[12]	√	×	√	×	√
Ours	√	√	√	√	√

Smari [9] [3] 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. [10] also provided an ABAC model mostly focused on the definition of attributes and the access control constraints. An approach close to ours was introduced by Covington and Sastry [11], who presented *contextual attribute access control* (CABAC) model which was realized in mobile applications. However, our approach takes a step further by describing the way such attributes are mapped to access rights (permissions) by means of TP-Functions and TP-Graphs. Recently, a noticeable approach was proposed by Jin et al. [12], whose approach is 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 III-B. However, our approach introduces a notion of security token and TP-Functions to capture the mapping between attributes and corresponding access rights. Finally, another interesting approach was presented by Zhang et al.[13], who presented 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 the proposed TP-Functions and TP-Graphs. Table I summarizes our findings and comparison results.

V. CONCLUDING REMARKS

In this paper, we have presented a model for ABAC and introduced a novel approach based on the concept of security tokens for establishing an association between attributes and access rights. Moreover, we have presented a case study depicting a set of interesting access control features, as well as an empirical comparison with existing approaches in the literature, which show our proposed model constitutes a promising result and is suitable for supporting security properties of collaborative systems and applications. Our future work will focus on the following enhancements: first, we aim to provide both a reference model and prototype testbed along with more realistic experiments. Second, we plan to explore how security tokens can be leveraged to model any security states in software and systems. As an example, our TP-Graphs may be used to model the set of states as a *security automata* as depicted in [16]. Third, we plan to develop a suitable language to specify access control policies based on our model.

VI. ACKNOWLEDGMENT

This work was partially supported by the grant from US Department of Energy (DE-SC0004308) and by the graduate fellowship from Mexico CONACYT (210775).

REFERENCES

[1] L. Wang, 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 security engineering*, ser. FMSE '04. New York, NY, USA: ACM, 2004, pp. 45–55.

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

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

[4] G. Stoneburner, A. Y. Goguen, and A. Feringa, "Sp 800-30. risk management guide for information technology systems," Gaithersburg, MD, United States, Tech. Rep., 2002.

[5] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[6] D. R. Kuhn, E. J. Coyne, and T. R. Weil, "Adding attributes to role-based access control," *Computer*, vol. 43, no. 6, pp. 79–81, Jun. 2010.

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

[8] L. Cirio, 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*, ser. OTM'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 1256–1266.

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

[10] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "A flexible attribute based access control method for grid computing," *Journal of Grid Computing*, vol. 7, no. 2, pp. 169–180, 2009.

[11] M. J. Covington 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*, ser. OTM'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 1996–2006.

[12] X. Jin, 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*, ser. DBSec'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 41–55.

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

[14] H. Shen, "A semantic-aware attribute-based access control model for web services," in *Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing*, ser. ICA3PP '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 693–703.

[15] Y. Wei, 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*, 2010, pp. 4451–4455.

[16] F. B. Schneider, "Enforceable security policies," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 1, pp. 30–50, Feb. 2000.