The ISC Int'l Journal of Information Security
ISeCure

January 2014, Volume 6, Number 1 (pp. 3–22)
http://www.isecure-journal.org

Access Control in Ultra-Large-Scale Systems Using a Data-Centric Middleware

Saeed Shokrollahi [1,*], Fereidoon Shams [1], and Javad Esmaeili [1]
[1] *Department of Computer Engineering, Shahid Beheshti University, Tehran, Iran*

A R T I C L E   I N F O.

Article history:
Received: 13 March 2014
Revised: 31 May 2014
Accepted: 24 June 2014
Published Online: 28 June 2014

Keywords:
Access Control, Colored-Petri-Nets Model, Middleware, Data-Distribution-Service Middleware, Ultra-Large-Scale Systems.

A B S T R A C T

The primary characteristic of an Ultra-Large-Scale (ULS) system is ultra-large size on any related dimension. A ULS system is generally considered as a system-of-systems with heterogeneous nodes and autonomous domains. As the size of a system-of-systems grows, and interoperability demand between sub-systems is increased, achieving more scalable and dynamic access control system becomes an important issue. The Attribute-Based Access Control (ABAC) model is a proper candidate to be used in such an access control system. The correct deployment and enforcement of ABAC policies in a ULS system requires secure and scalable collaboration among different distributed authorization components. A large number of these authorization components should be able to join different domains dynamically and communicate with each other anonymously. Dynamic configuration and reconfiguration of authorization components makes authorization system more complex to manage and maintain in a ULS system. In this paper, an access control middleware is proposed to overcome the complexity of deployment and enforcement of ABAC policies in ULS systems. The proposed middleware is data-centric and consists of two layers. The lower layer is a Data-Distribution-Service (DDS) middleware used for loosely-coupled-communication among authorization components. The upper layer is used for secure configuration and reconfiguration of authorization components. An executable model of the proposed middleware is also represented by a Colored-Petri-Net (CPN) model. This executable model is used to analyze the behavior of the proposed middleware.

1 Introduction

The scale of system-of-systems is growing in the recent years, dramatically. Ultra-Large-Scale (ULS) systems have extraordinary scale in many dimensions, such as number of policy domains and enforceable mechanisms, number of connections among components, number of overall interactions, amount of data, and number of people involved [1]. The consequence of the scale of ULS systems makes a serious need to have a scalable and dynamic access control that accepts, denies or restricts access to the systems. In addition, all attempts made to access the system and processes involved could be logged in detail by the access control system. The logged data may be audited

* Corresponding author.
Email addresses: s_shokrollahi@sbu.ac.ir (S. Shokrollahi), f_shams@sbu.ac.ir (F. Shams), j-esmaeili@sbu.ac.ir (J. Esmaeili).
ISSN: 2008-2045 © 2014 ISC. All rights reserved.

and the result of the audition could be used in new *decision-making* processes, while the access control system is in progress. Various access control models are in use, including Mandatory Access Control [2], Discretionary Access control [3], and Role Based Access Control [4, 5]. These models are usually effective in unchangeable distributed systems that deal only with a set of known *subjects* which access a set of known *resources* [6, 7]. However, Attribute Based Access Control (ABAC) model provides more flexibility, granularity, and dynamicity [6–14]. Therefore, ABAC model is a proper candidate to be used in ULS systems. In ABAC model, access decisions are based on attributes of the *subjects* and *resources*.

In ABAC model a resource could be accessed by a *subject* while the *subject* is unknown. In such models, the enforcement of policies usually requires collaboration among different distributed authorization components. For example, Policy Administration Point (PAP), Policy Enforcement Point (PEP), Policy Decision Point (PDP), and Policy Information Point (PIP) components are respectively used for administration, enforcing policies, making decisions, and providing a collection of attributes of related entities. The *logging* process could be also done by an authorization component, which is called Policy Logging Point (PLP) in this paper.

- The weakness of ABAC model is the complexity of administration and maintenance in open and large-scale distributed systems [8, 9]. In addition, an authorization system based on this model has to have some desirable features as follows, in order to be widely used in ULS systems efficiently. A large number of authorization components should be able to join the authorization system dynamically and communicate with each other anonymously. These components might be located in distributed and heterogeneous nodes and different autonomous domains.
- *Multi policy*, *multi decision*, and *multi information* capabilities in a domain or among several domains should be supported. These capabilities allow each component to receive several policies, several decisions, and several information data from other components and properly combine them to make a decision, final-decision, or needed-information, respectively.
- Flexible and loosely-coupled dependencies between authorization components, to reduce the administration overhead, are necessary.
- The capability of dynamic and correct *reconf* of components is important.
- The integration of new types of components into the authorization system should be done easily. For example, adding a new component for specifying policies, performing obligations, verifying certificates and using ontology.
- The component failures are considered as normal and frequent events. These failures should be tolerated and tackled by authorization system.

To overcome the complexity of deployment and enforcement of ABAC policies in ULS systems and provide the mentioned desirable features, a two-layer Access Control (AC) Middleware is presented in this paper. The lower layer, named Data-Distribution Layer (DDL), is a Data-Distribution-Service (DDS) middleware used for loosely-coupled-communication among authorization components. DDS is a data-centric middleware [15]. However, in most of distributed access control systems, authorization components communicate with each other via a message-centric middleware [11]. A data-centric middleware focuses on shared-data and system-state-information, not on component-specific-message sets. General-messages are used by the data-centric middleware to exchange the system states. Moreover, a data-centric middleware is aware of data being shared among components. This awareness leads to a set of capabilities, such as Quality of Services (QoS) policies in DDS. The QoS policies of DDS are applied by the AC middleware to reduce the complexity of communication among the authorization components. The AC middleware exploits the data-centric and publish-subscribe characteristics of DDS, which increase the scalability, elasticity, availability, location and network independence, auditability, and manageability of distributed systems [16, 17].

The upper layer, named Authorization Layer (AL), is used for *conf* and *reconf* of authorization components, independent of the lower layer. In the AC middleware the notion of combining ABAC policies, combining decisions and combining information received from several components is used to achieve *Multi policy*, *multi decision*, and *multi information* capabilities. Publish-subscribe architecture that is used in DDS, may cause new threats to the authorization system [18]. To reduce these threats, the upper layer helps authorization systems to provide a mechanism to allow only authorized components to read and write data in DDS. The provided mechanism exploits the Open Architecture for Secure Interworking Services (OASIS) model [19–21]. An executable model of the AC middleware is also represented by a Colored-Petri-Net (CPN) model, using CPN tools software.

The rest of this paper is organized as follows: Section 2 describes some related works about using the ABAC and OASIS models in large-scale systems. Section 3 gives background information about principles of the DDS middleware and the OASIS model. Section 4 describes the proposed AC middleware. Section 5 represents an executable model of the proposed middleware by a CPN model. Section 6 presents an analysis of the proposed middleware. Section 7 concludes the paper and discusses possible future work.

2 Related Work

The ABAC model [22] specifies the access permission based on attributes of three entities named *subject*, *resource*, and *environment*. The *subject* submits its access request to the access control system for desired *resource*. The *environment* represents the required context information to be used in *decision-making* process. The context information is not related to a specific *subject* or *resource*. Several research articles show that the ABAC model provides more flexibility, granularity, and dynamics compared with other access control models [6–13]. Some of these articles have introduced a new access control model or architecture based on ABAC model. Singh *et al.* [12] proposed a scalable and flexible authorization model for large-scale distributed systems based on ABAC model. In the proposed model, a policy decision point has been considered as a combination of different policy decision points. In this article, a policy information point has also been implemented as a combination of different policy information points. In another article, a multiple-policy model within large-scale device collaboration systems, based on ABAC model, is introduced by Liang *et al.* [10]. In their proposed model, each single policy includes a priority and the final decision is made based on the combination of several policies with different priorities.

In a recent work, the strength and weakness of ABAC model are listed by Verma *et al.* [9]. The list shows that in a large open system, ABAC model has more granularity, flexibility, and dynamics compared with role-based access control model. The article also represents that the authorization system based on ABAC model is complex to manage and maintain.

To overcome the aforementioned complexity and improve the scalability, availability, and flexibility of the authorization system, some researchers have used a message-centric or a publish-subscribe architecture for communication among access control components [11, 18]. In order to reduce the overhead of administration, and to increase the availability of authorization system, and improving the software development process, a publish-subscribe model is used among policy decision points and policy enforcement points by Wei *et al.* [18]. In another article, a distributed authorization middleware for attribute-based policy evaluation is presented by Goovaerts *et al.* [11]. They use a lifecycle and dependency management of authorization components to guarantee that configurations are consistent with respect to deployed policies and applications. In this article a message-centric middleware is used for message delivery among enforcement, decision, and information components.

Some researchers have used the OASIS model in their work and show that OASIS is a promising approach for securing large-scale publish-subscribe middlewares and multi-domain systems [21, 23–25].

To best of our knowledge, in our work, DDS has been used for the first time in an access control middleware. The OASIS model has also been used for the first time in a data-centric middleware.

3 Background

A brief introduction to DDS and OASIS model is presented in this section.

3.1 Data Distribution Service Middleware

The Object Management Group (OMG) DDS specification [15] provides a data-centric publish-subscribe communication standard for wide variety of computing environments. DDS is a scalable, platform-independent, and location-independent middleware, using a peer-to-peer communication model [26]. Over the last few years, DDS has been extended to improve its efficiency over the world-area network and improve its scalability for ULS systems [27–29].

Figure 1 illustrates the main entities of DDS. In this figure, there is a domain consisting of two domain-participants. A domain-participant is an entity that represents an application's participation. The domain-participants are bound together within a same domain, since more than one domain may exist.

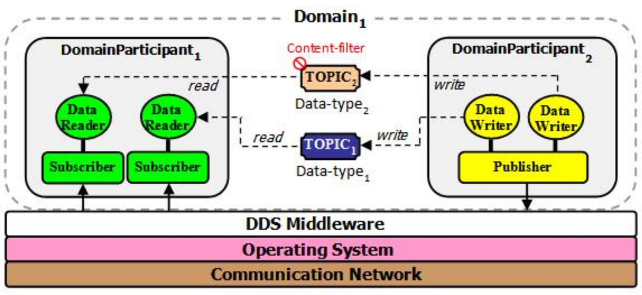


Figure 1. The DDS concepts and entities.

Each domain provides a virtual communication environment for domain-participants having the same domain identity. A domain-participant acts as a manager for publishers and subscribers entities. A publisher is a container to group together individual data-writers. Similarly, a subscriber is a container that groups together individual data-readers. A data-reader can obtain its subscribed data in the DDS middleware, via listener-based asynchronous or waitset-based synchronous mechanism. A topic is a data holder, which makes connection of data-writers and data-readers possible. Each topic has a defined data type and is identified by a name. DDS can interpret the exchanged data-samples of topics, and provides data-caching and *content-based-filtering*. A data-sample can be some data value, and an instance is some data-samples with the same key value. Communication via topics is anonymous and transparent. It means that publishers and subscribers are not concerned about how and who creates, reads, and writes topics. When a topic published by a data-writer matches the topic subscribed by a data-reader, the communication may occur.

The DDS middleware also supports parametric QoS policies. A QoS is a set of characteristics that controls some aspects of the behavior of the DDS service. These QoS policies can be configured at various levels of granularity such as topics, publishers, data-writers, subscribers, and data-readers. All of these issues are handled by the DDS middleware, to simplify the development of distributed applications, and relieve the applications from the burden of transmitting and managing data.

3.2 OASIS

The OASIS model [19–21, 23] provides an architecture for role-based access control in open, distributed environments. In OASIS, roles are autonomously managed in a decentralized way. Each service in this model is responsible for the classification of its users into named roles, using a formal logic to specify precise conditions for entering each role. The specification of conditions comprises a set of rules, where a role-activation-rule for a target role, named r, takes the form:

$$r_1, \cdots, r_i, a_1, \cdots, a_j, e_1, \cdots, e_k \vdash r \tag{1}$$

Where r_x are prerequisite roles, a_y are appointment certificates, and e_z are environmental constraints. The environmental constraints restrict the activation of a role in a specific time and location. If a user holds all the r_x , a_y, and e_z in such activation-rule, a Role Membership Certificate (RMC) is issued for the user to enter the target role. A principal is a user that has satisfied the conditions for possession of a role. An RMC is an encryption-protected capability, which includes the role name, the identity of the principal to which it was issued, and a reference to the issuing service. Each issuing service creates and maintains a Credential Record (CR) for each RMC. The CR indicates the predicates against which the RMC was issued and lists all other services which have issued RMCs to the principal based on this CR. Any predicate that must remain true for the principal to remain active in the role is tagged as a role-membership-condition. Such predicates are monitored, and their violation triggers revocation of the role and related privileges from the principal.[A1] A mechanism is used to achieve rapid revocation of the dependent RMCs issued by other services. An authorization rule for some privilege *pr* takes the form:

$$r, e_1, \cdots, e_k \vdash pr \tag{2}$$

According to such rules the privilege *pr* is assigned to the role r, if all the environmental constraints are satisfied. An authorization policy comprises a set of such rules. OASIS roles and rules are parameterized that allows fine-grained policy requirements to be expressed and enforced. Another OASIS concept is appointment, by which a principal that has an appointment certificate may delegate a role that does not itself possess.

4 The Proposed Access Control Middleware

A two-layer AC Middleware is proposed to make an attribute-based authorization system more suitable for ULS systems. Figure 2 shows an overview of the proposed middleware, which is used by authorization components of domains to authorize the *subjects* of a ULS system to access the *resources* of the system. A *subject* can be a user, service or any other entity on behalf of a user. A *resource* can be a data, service, software or any other peripheral, or processing devices. In our basic model, an access control policy for a *resource* is specified and done at two sides, the authorization system side and the *resource* owner side. In the authorization system side, each administrator specifies an independent, domain-internal ABAC policy that defines the access rights granted to the *subjects*, and the *resource* owner has no control over that. In the basic model a centralized ABAC policy specification is assumed, and the *resource* owner trusts the administrator to specify the ABAC policies. A decentralized policy specification model is introduced later, as an extension of the basic model. In the owner side, the

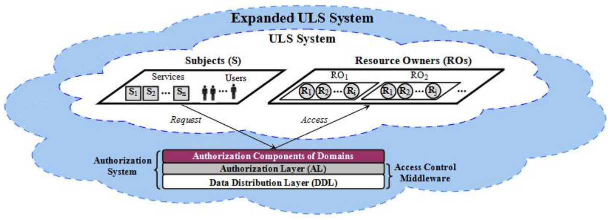


Figure 2. An overview of the proposed access control middleware.

resource owner authorizes the related domains of the authorization system to access its resources. However, each resource owner may independently have its own access control model, which is an issue that our model is not concerned about. The proposed middleware may be used by more than one domain.Figure 3 shows more details of the AC middleware used by two domains.

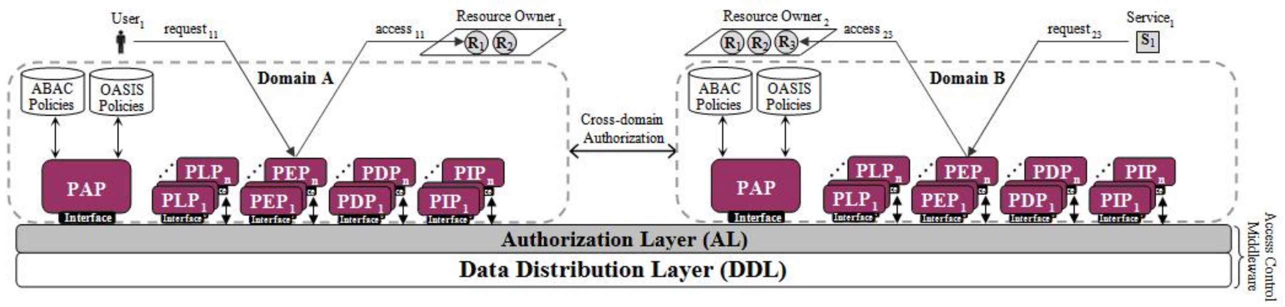


Figure 3. The proposed access control middleware, for two domains.

Each domain consists of several components. One administrator component, called PAP, which administrates the other components of the domain. The rest of the components in the domain are classified in four component types: PEP to enforce ABAC policies, PDP to make decisions, PIP to provide a collection of attributes of related entities such as subjects and resources, and PLP to log authorization processes. The *multiple-instances* of each type of component may exist in the domain. For example PEP$_1$ to PEP$_n$ instances of PEP component type. The number of instances in each type may dynamically vary according to special requirements.

In the proposed middleware, the lower layer, named DDL, is a DDS used for loosely-coupled and platform-independent communication among the authorization components. DDS uses a publish-subscribe architecture that allows authorization systems to easily provide *multi-information* and *multi-decision* capability. DDS is also a data-centric middleware consisting of many QoS policies that can reduce the complexity of communication among the authorization components and improve their development process.

The upper layer, named AL, is used for *conf* and *reconf* of the authorization components, independent of the lower layer. Publish-subscribe architecture that is used in DDS may cause two threats to the authorization systems [18]. First, a malicious component can attempt to lower the performance of an authorization system. For example, a malicious publisher component can send a large number of junk data-samples to DDS to increase its load, and a malicious subscriber component can register a large number of junk subscriptions in DDS to increase its time in finding an interested subscriber. Second, a malicious component can subscribe to any ongoing request and publish false response. To prevent these attacks, the AL layer helps authorization systems to provide a mechanism to allow only authorized components to read and write data in the DDL layer, and also receive the related *conf* and *reconf* information. The provided mechanism exploits the OASIS policies that are defined by the PAP components.

When a *subject* from one domain is allowed to access some *resources* in other domains, we have the concept of cross-domain access. To enable this concept in the AC middleware, a component should be allowed to join other domains. Hence, a *subject* from one domain can be authorized by other domains. This kind of authorization, called cross-domain authorization, is achieved by exploiting the OASIS model in the AC middleware. Each PAP component has two policies repositories, named OASIS and ABAC, to hold its OASIS and ABAC policies.

The details of the proposed AC middleware are presented in the following three subsections. Section 4.1 discusses the administration of authorization components including the *conf* and *reconf* processes, and explains the consisting elements of the *conf* and *reconf* information. Section 4.2 discusses the needed processes for enforcing policies such as *dependency-recognition*, *decision-making*, and *information-providing* processes. Section 4.3 discusses the logging process of authorization activities.

4.1 Administration of Authorization Components

The PAP component of a domain is responsible for creating and sending required *conf* and *reconf* information of the authorization components. In the AC middleware, authorization components need initial *conf-information* to join a domain. Since the *confs* may change after the join process, reliable and dynamic *reconf-information* is also needed. The *conf* and *reconf* information of a component consist of *topics-list*, *ABAC-policies*, *combining-algorithms*, and *priorities* of policies, decisions, and information. Each component gets the related *conf* and *reconf* information based on its roles. The PAP component defines and exploits the OASIS policies to classify the authorization components into named roles. All the roles and their role-activation-rules are held in a table, called Component Roles Table (see Table 1), which is saved in the OASIS repository. In this table, each type of component has a conceptual role, e.g., PEP has Enforcement Role (ER), PDP has Decision Role (DR), PIP has Information Role (IR) and PLP has Logging Role (LR). On the basis of the OASIS model, the conceptual role could be parameterized to create new roles, e.g., ER(1) to ER(4) roles for PEP component type. Note that the parameterized roles for each component type are conceptually the same in action, similar to a function which is applied with different arguments. Each row of the table holds a component type followed by a sequence of needed roles for the component type. An RMC is issued for an authorization component to enter each target role. A component should hold all the prerequisites roles and certificates prescribed in the role-activation-rule to receive the RMC certificate, and to be considered as a principal component. An authorization component may join several

Table 1. Component roles table.

Component Type	Sequence of needed roles				
PEP	ER(1)	ER(2)	ER(3)	ER(4)	⋯
PDP	DR(1)	DR(2)			
PIP	IR(1)	IR(2)			
PLP	LR(1)	LR(2)	LR(3)		

domains with proper certificates from a decentralized trust management and RMC certificates from PAP components. This kind of join process, supported by OASIS, provides an authorization, previously called *cross-domain* authorization. For *cross-domain* authorization, the PAP components must specify the proper role-activation-rules based on the issued certificates.

The description of applying a decentralized trust management is presented by Pesonen *et al.* [21]. The key issue in the cross-domain authorization is to map the external subject's privileges to the corresponding privileges in the local domain. In the ABAC model, attributes reflect subject's privileges. Therefore, the AC middleware exploits the attribute-mapping-approach presented by Long *et al.* [30].

4.1.1 Conf and Reconf Information

The PAP component of a domain creates and sends *conf* and *reconf* information of the authorization components. Each principal component, based on its RMC certificates that indicate its roles, gets the related *conf* and *reconf* information. The *conf* and *reconf* information consist of *topics-list*, *ABAC-policies*, *combining-algorithms*, and *priorities* of policies, decisions, and information, which are explained in the following three subsections.

4.1.1.1 Topics List

The DDL layer handles the actual distribution of data on behalf of the AL layer and authorization components. The topics defined in the DDL layer are basic connection points between the authorization components. The authorization components use the topics for different communication types such as *conf*, *reconf*, *decision-making*, *information-providing*, and *policy-specification* which are shown in the first column of Table 2. Each type of topics has some attributes, placed in the first row of the table, and a brief explanation of those attributes is as follows.

The content of each topic is represented in some structured fields, similar to tuples. For example, the topic-content of RPEP has two fields: *reconf-information* (RE) and *expected-time* (ET). The content-filtering is used to ensure that the data-samples of a topic are sent to only one specific corresponding component, by which the performance of the AC middleware could be improved. The publisher and subscriber attributes of a topic show which component types can write or read that topic. For example, the PAP components can write and the PEP components can read the RPEP topic. The topic multiplicity attribute indicates that whether a topic type can have *single-instance* or *multiple-instances* in a domain. As an example, RPEP type can have $RPEP_1$ to $RPEP_n$ instances. The PAP component defines multiple instances of a topic to create proper dependencies among the authorization components, and prevents the components from receiving the unrelated data. Suppose we have two instances of PEP component, PEP_1 and PEP_2, and *reconf-information* of these two instances are different, then the PAP component can define two instances of RPEP topic, $RPEP_1$ and $RPEP_2$, so that PEP_1 is allowed to read only $RPEP_1$, and PEP_2 is allowed to read only $RPEP_2$. In this case, the *reconf-information* of PEP_1 cannot be read by PEP_2. A *topics-list* of a role determines which topics can be accessed to read or write by the components that possess the role. The *topics-list* consists of two lists for write and read permissions, heading with *write-list* and *read-list* labels. The *topics-list* may have all types of topics except JOIN and CONF which are always used in joining process. Figure 4 shows a typical example of a *topics-list* that allows the components that possess the role ER(1) to write three topics and read two topics. The description of type of these topics is presented in Table 2. The name and type of all the needed topics in a domain are defined by the PAP component of the domain.

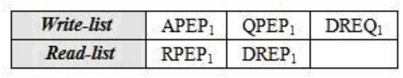
A *topics-list* of a role determines which topics can be accessed to read or write by the components that possess the role. The *topics-list* consists of two lists

Table 2. All topic types used in the proposed access control middleware.

Topic Attribute / Type of Communication	Topic type	Topic-content	Content-filtering	Publisher	Subscriber	Topic multi-plicity
Conf	JOIN	Join-request	NO	PEP,PDP, PIP,PLP	PAP	Single
	CONF	Initial conf-information	Yes	PAP	PEP,PDP, PIP,PLP	Single
Reconf	RPEP	Reconf-information of PEP	NO	PAP	PEP	Multiple
	APEP	Apply-time of PEP	NO	PEP	PAP	Multiple
	RPDP	Reconf-information of PDP	NO	PAP	PDP	Multiple
	APDP	Apply-time of PDP	NO	PDP	PAP	Multiple
	RPIP	Reconf-information of PIP	NO	PAP	PIP	Multiple
	APIP	Apply-time of PIP	NO	PIP	PAP	Multiple
	RPLP	Reconf-information of PLP	NO	PAP	PLP	Multiple
	APLP	Apply-time of PLP	NO	PLP	PAP	Multiple
Decision-making	QPEP	Required-attributes of PEP	NO	PEP	PDP,PLP	Multiple
	DREQ	Decision-request	Yes	PEP	PDP,PLP	Multiple
	DREP	Decision-reply	Yes	PDP	PEP,PLP	Multiple
Information-providing	QPDP	Required-attributes of PDP	NO	PDP	PIP,PLP	Multiple
	IREQ	Information-request	Yes	PDP	PIP,PLP	Multiple
	IREP	Information-reply	Yes	PIP	PDP,PLP	Multiple
ABAC-policy-specification	SPOL	Specified-policy	NO	PAP,PSP	PAP	Single
	APOL	Apply-time of specified-policy	Yes	PAP	PAP,PSP	Single

for write and read permissions, heading with *write-list* and *read-list* labels.

The *topics-list* may have all types of topics except JOIN and CONF which are always used in joining process. Figure 4 shows a typical example of a *topics-list* that allows the components that possess the role ER(1) to write three topics and read two topics. The description of type of these topics is presented in Table 2. The name and type of all the needed topics in a domain are defined by the PAP component of the domain. The AL layer of the AC middleware creates the



Write-list	$APEP_1$	$QPEP_1$	$DREQ_1$
Read-list	$RPEP_1$	$DREP_1$	

Figure 4. Topics list of ER(1) role.

structure of the topics, and sets the *content-filtering* attribute for desired topics. The AL layer also sets the needed QoS policies of the topics. The QoS policies control the distribution of data, to reduce the complexity of administration. Table 3 shows the QoS policies for each type of topics, which are defined by the PAP components. The QoS policies that are shown in the first column of Table 3 are as follows:

Durability: presents the lifetime of data-samples of each topic type. The *volatile* value specifies that once data-sample is published, it is not maintained by the DDL layer for delivery to late-joining-components. The *persistent* value indicates that the DDL layer stores the data-sample persistently to make it available to late-joining-components even after restarting of shutting down the whole authorization system.

History: specifies the number of data-samples of each topic type that must be stored for authorization components.

Ownership: determines which data-writer owns the write-access to a topic when there are multiple data-writers and the ownership policy value is exclusive. In case of shared values, data-writers can concurrently update the topic.

Destination-order: defines the order of changes made by authorization components to instances of each topic type, based on the source or destination *time-stamps*. The AC middleware uses *source-time-stamp* values in order to provide data-samples in the same order for the subscriber components.

Reliability: shows the level of reliability associated

Table 3. Quality of service policies of the topic types.

QoS \ Topic type	Join	CONF	QPEP, QPDP	DREQ, DREP, IREQ, IREP	RPDP, RPEP, RPIP, RPLP	APDP, APEP, APIP, APLP
Durability	persistent	persistent	persistent	volatile	persistent	persistent
History	keep-all	keep-all	keep-last-one	keep-last-one	keep-all	keep-all
Ownership	shared	Exclusive	shared	shared	shared	shared
Destination-order	source-time-stamp	source-time-stamp	source-time-stamp	source-time-stamp	source-time-stamp	source-time-stamp
Reliability	reliable	reliable	reliable	reliable	reliable	reliable
Time-based-filter	zero	zero	zero	zero	zero	zero
Deadline	infinite	infinite	infinite	infinite	infinite	infinite
Lifespan	infinite	infinite	infinite	LS	infinite	infinite
Latency-budget	LB_3	LB_3	LB_2	LB_2	LB_1	LB_2
Transport-priority	TP_3	TP_3	TP_2	TP_2	TP_1	TP_2

with data diffusion of each topic type. The *reliable* value specifies that a data-sample should be resent until its acknowledgement is received.

Time-based-filter: sets a minimum period of time, in which a data-reader can receive at most one data-sample and the rest of them are filtered. The zero value means that the data-reader receives all data-samples and no one is filtered.

Deadline: sets a maximum time in which an update for instances must be published. The deadline policy of all topic types is infinite by default, otherwise the PAP components may set it.

Lifespan: sets the period of time during which a data-sample is valid. The lifespan policy of the DREQ, DREP, IREQ, and IREP topic types, which contain decision and information request and reply, is set to the LS value. It means that the LS value is the maximum duration that each request or reply is valid.

Latency-budget: specifies the maximum acceptable delay from time a data-sample is written by a publisher component until it is inserted in data-cache of a subscriber component. The AC middleware uses three levels of latency-budget which indicates the urgency associated with transmitted-data of each topic type: LB_1, LB_2, and LB_3. These three levels comply with the following formula.

$$LB_3 > LB_2 > LB_1 \geq 0 \qquad (3)$$

Transport-priority: allows the DDL layer to optimize the transporting of data-samples of topic types with different priorities. The AC middleware uses three levels of priority which indicates the data-samples importance: TP_1, TP_2, and TP_3. These three levels comply with the following formula.

$$TB_1 > TB_2 > TB_3 \qquad (4)$$

More description of attributes of topics and the applications of the QoS policies will be presented in the next sections.

4.1.1.2 ABAC Policies

In the AC middleware, the *ABAC-policies* of a domain are centrally specified by the PAP component of the domain. The policies are stored in the *ABAC-policies* repository, and distributed to the related PDP components. The PAP components exploit the XACML standard [31] to specify the *ABAC-policies*. Each *ABAC-policies* includes five fields: identity, change-time-stamp, policy-target, rule-set, and rule-combining-algorithm. The identity field is a unique label assigned to each policy in the authorization system. The label consists of a domain name followed by a unique number in the domain. The change-time-stamp field indicates the last time that the policy has been updated. The policy-target field identifies a set of subjects, resources, actions, and environments to which the policy is applicable. The rule-set is a set of rules. Each rule itself consists of three elements named rule-target, rule-condition, and effect. The rule-target element has the same structure as the policy-target. The rule-condition element specifies restrictions on the attribute values in a request, and the effect element indicates that the request should be permitted (P) or denied (D). The rule-combining-algorithm field is used to resolve conflicts among applicable rules with different effects. If a request does not satisfy both the rule-target and rule-condition elements, the decision response would be not-applicable (NA). If an error occurs in the evaluation process of a rule, the decision response would be indeterminate (IN).

An extension of the basic model is shown in Figure 5. In this extension, unlike the basic model, a decentralized policy specification is provided. In the extended model, each domain has Policy Specification Point (PSP) type. The PSP component in a domain specifies the *ABAC-policies* of the domain or the ABAC-policies required by the other domains. A PAP component may receive policies specified by different languages. Therefore, the received policies should be compiled to produce the specification policies conformant with XACML. The compiler assigns and attaches the identity and change-time-stamp fields to each produced policy. Note that the extended model allows the resource owners to specify their own *ABAC-policies*, which could be used later by the PAP components. In the extended model, each domain can have several redundant instances of the PAP component to increase the availability of the PAP component. In Table 2, the last type of communication row shows the topic types needed for the extended model. Ardagna *et al.* [32] provide a formalization of concepts that have to be supported for enforcing the new access control paradigm needed in open systems. They illustrate how the concepts can be deployed in the XACML standard by exploiting its extension points for the definition of new functions. This extension of XACML can be used in the above extended model. Bonnati *et al.* [33] propose an algebra for composing independent authorization specifications, possibly in different languages and according to different policies. This work can also be used in the extended model to compose and compile different policies.

4.1.1.3 Combining Algorithms and Priorities

An authorization system, which is applied in a ULS system, needs to combine the result of multiple policies to make a decision. The multiple decisions and multiple information, received from different components, should also be combined to make the final-decision and provide the needed-information, respectively. The AC middleware exploits three types of combining-algorithms: Policy Combining-algorithm (PCA), Information Combining-algorithm (ICA), and Decision Combining-algorithm (DCA). The descriptions of these combining-algorithms are as follows.

a. Policy Combining-algorithm (PCA): this type of combining-algorithm is applied by the PDP components to combine the result of multiple *ABAC-policies* to make a decision. The identity and change-time-stamp of polices, which are combined to make a decision, are returned along with the decision in order to check consistency and integrity of policies. The *ABAC-policies* for the PDP components may have different policy-priorities. The policy-priorities are used by PCAs to combine the result of policies that have the highest priority.

The PAP components use a tuple of four decision-elements E_P, E_D, E_{NA}, and E_{IN}, to express PCAs. Each decision-element expresses a condition that is not overlapped with conditions of other elements in the tuple. Hence, the conditions of elements could be evaluated in any order. By satisfying E_P, E_D, E_{NA}, and E_{IN}, the result of combination will be Permit, Deny, Not-applicable, or Indeterminate, respectively. This tuple, named decision-tuple, is inspired by the PCL Language [34]. PCL is based on automata theory and linear constraints, and it has been integrated with XACML. The four decision-elements of decision-tuple can be filled by different approaches. In this section two approaches are explained.

In the first approach, the tuple can be filled by four regular expressions over $\Sigma = \{P, D, NA, IN\}$, which expresses a set of acceptable strings that yields the corresponding result, such as P in EP. For example, the first-applicable PCA can be expressed using regular expressions as follows. Remind that P, D, NA, and IN denotes Permit, Deny, Not-applicable, and Indeterminate decisions, respectively.

$$\begin{aligned}(E_P = NA^*P\Sigma^*, E_D = NA^*D\Sigma^*,\\ E_{NA} = NA^*, E_{IN} = NA^*IN\Sigma^*)\end{aligned} \tag{5}$$

The first approach cannot directly express the counting-based PCAs such as the weak-majority PCA, in which a decision (P or D) wins if it has more votes than the opposite. In the second approach, the tuple can be filled by four linear constraints. The linear constraint is a predicate consisting of variables ♯P, ♯D, ♯NA, and ♯IN using relational, conjunctive and disjunctive operators. The ♯P, ♯D, ♯NA, and ♯IN variables represent the number of policies that return P, D, NA, and IN decision, respectively. For example by using the second approach, the deny-override PCA can be expressed as the following decision-tuple.

$$\begin{aligned}(E_P = \sharp P > 0 \wedge \sharp D = 0 \wedge \sharp IN = 0,\\ E_D = \sharp D > 0, E_{NA} = \phi, E_{IN} = \sharp D = 0 \wedge \sharp IN > 0)\end{aligned} \tag{6}$$

b. Information Combining-algorithm (ICA): this type of combining-algorithm is applied by the PDP components to combine information received from different PIP components. The PDP components may receive the information-responses with different information-priorities from PIP components of different domains. The information-priorities are used by ICAs to combine the information-responses that have the highest priority. If no response received from PIP components, the related ICA sends a Null message to the PDP component. In this case, the PDP component sends indeterminate result for the related evaluation. ICA is used for the following purposes.

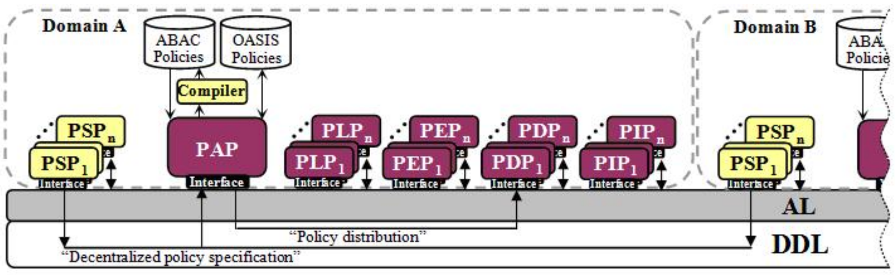


Figure 5. Decentralized policy specification.

- **Information Availability**: to achieve higher information-availability, the information-request of a PDP component is published to several redundant PIP components. The redundancy of PIP component increases the chance of receiving at least one acceptable information-response on time, even some of the PIP components fail. Regarding this purpose, ICA returns the first information-response and discards the other responses.
- **Information Composition**: sometimes the entire required information to make a decision is not provided by just one PIP component. Hence, regarding this purpose, ICA merges different parts of needed information, which is received from different PIP components.
- **Information Consistency and Integrity**: when a PDP component receives information-responses from several redundant PIP components, the information inconsistency and non-integrity problems may occur. These possible problems should be checked and resolved by ICA. To resolve the problems, ICA selects at least n, $n \geq 2$, information-responses and compares change-time-stamps and values of information-responses. An information-response may have several fields and each field has its own change-time-stamp and value. If the information-responses have different change-time-stamps, it means some fields are not updated yet, and there is inconsistency problem. In this case, ICA returns the information-response with the most recent change-time-stamp. If the information-responses have the same time-stamps, but the different values, it means that there is a non-integrity problem and ICA cannot return any response.

c. Decision Combining-algorithm (DCA): this type of combining-algorithm is applied by the PEP components to combine decisions received from different PDP components. The PEP components may receive the decision-responses with different decision-priorities from the PDP components of different domains. The decision-priorities are used by DCAs to combine the decision-responses that have the highest priority. DCA is used for the following purposes.

- **Decision Availability**: the redundant PDP components provide more decision availability. In this case, DCA returns the first decision-response of PDP components as a final-decision.
- **Decision making**: when decision-responses are received from different PDP components, the final-decision is made by combining the responses. In this case, DCA is expressed by a decision-tuple that is filled by four linear constraints. For example, the at-least-two-permit DCA is expressed as the following decision-tuple.

$$(E_P = \sharp P > 1, E_D = \sharp P < 1 \wedge \sharp D > 0, \quad (7)$$
$$E_{NA} = \phi, E_{IN} = \sharp P < 1 \wedge \sharp D = 0 \wedge \sharp IN > 0)$$

- **Decision Consistency and Integrity**: DCA is used to check and resolve the inconsistency of decisions and non-integrity of decisions. These problems may occur when a PEP component receives decision-responses from several redundant PDP components. In this case, DCA selects at least n, $n \geq 2$, decision-responses and compares identities, change-time-stamps, and values of decision-responses. If the decision-responses have the same identities and different change-time-stamps, it means some policies are not changed yet and there is an inconsistency problem. In this case, DCA returns the decision-response with the most recent change-time-stamp. If the decision-responses have the same identities and the same change-time-stamps but the different values, it means that there is a non-integrity problem and DCA cannot return any response.

To summarize the issues discussed in this section, Table 4 shows the types of combining-algorithms and priorities for each enforcement role and decision role.

4.1.2 Conf Process of an Authorization Component

When an authorization component wants to join a domain, it needs an initial conf-information that is defined and issued by the PAP component of the

Table 4. Combining algorithms and priorities of roles.

Role Type	Combining algorithms	Priorities
Enforcement Role	Decision Combining-algorithm	Decision-priorities
Decision Role	Policy Combining-algorithm	Policy-priorities
	Information Combining-algorithm	Information-priorities

domain.

An example of a conf process for one PDP component, named PDP$_i$, is shown in Figure 6. The join-request of PDP$_i$ is sent to the AL layer of PDP$_i$ (AL.PDP$_i$). By receiving the join-request, AL.PDP$_i$ creates a data-writer for the JOIN topic and writes an instance of this topic, which consists of five fields: certificate (CE), domain-name (DN), identity (ID), requested-role (RR), and active-roles (AR) of the component. After the write operation, AL.PDP$_i$ creates a data-reader for the CONF topic, waiting to receive the conf-information. The join-request is received by the AL layer of the PAP component (AL.PAP), and is passed to the PAP component. The PAP component checks the join-request based on its OASIS policies, and replies the related conf-information. Based on the replied conf-information, AL.PAP writes an instance of the CONF topic, which consists of seven fields: domain-name (DN), identity (ID), role-certificate (RC), topics-list (TL), combining-algorithms (CA), priorities (PR), and policies (PO). Note that the PEP, PIP, and PLP components have no policy repository, so the policies fields for them are set to the empty value. Since the CONF topic has content-filter on domain-name and identity fields, only one component whose domain-name and identity matches the content of the CONF instance receives the instance. After

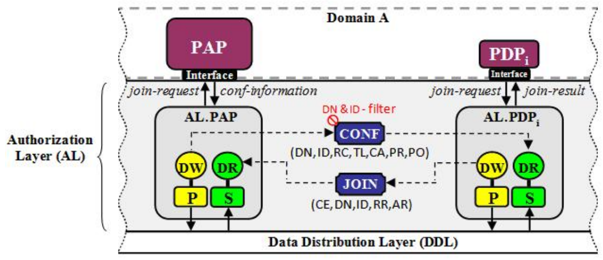


Figure 6. An example of a *conf* process for one PDP component.

receiving the conf-information, AL.PDP$_i$ deletes the data-writer and data-reader, and according to the conf-information creates publishers, data-writers, subscribers, data-readers, combining-algorithms, and policy-repository of PDP$_i$. Finally, AL.PDP$_i$ sends a join-result to PDP$_i$ to inform the result of join-request.

Referring to Table 3, the ownership value for the CONF topic is set to exclusive value. It means that if a domain, in the extended model of AC middleware, has several redundant instances of the PAP component, the conf-information can only be written by one instance of the PAP component. The durability value for the JOIN and CONF topics is set to persistent, and the history value for these topics is set to keep-all, as well. Therefore, when the DDL layer receives instances of the JOIN and CONF topics, it is not concerned about the possible failure of the AL layer during the conf process.

4.1.3 Reconf Process of an Authorization Component

The roles of an authorization component and the conf-information of a role may be changed by the PAP components, dynamically. Hence, the authorization components must be reconfigured in these cases.

An example of a *reconf* process for two PDP components, named PDP$_i$ and PDP$_j$, is shown in Figure 7. A reconf-information and its expected-time made by the PAP component are sent to AL.PAP. The time that the PAP component expects that the reconf-information can be applied, is called expected-time. Hence, the apply-time of a reconf-information cannot be less than its expected-time. By receiving the reconf-information for PDP$_i$ and PDP$_j$, AL.PAP writes an instance of the RPDP$_k$ topic, which consists of two fields: reconf-information (RE) and expected-time (ET). Note that the RPDP, RPEP, RPIP, and RPLP topics are used by AL.PAP to send reconf-information to the PEP, PDP, PIP, and PLP components, respectively. The instance of RPDP$_k$ is read by AL.PDP$_i$ and AL.PDP$_j$, and the expected-time is passed to PDP$_i$ and PDP$_j$. If there is a request whose sending-time is greater than the expected-time, then the request will be suspended. When the reconf-

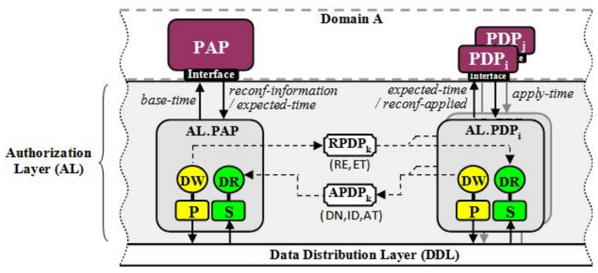


Figure 7. An example of a *reconf* process for two PDP components.

information is applied, AL.PDP$_i$ and AL.PDP$_j$ send a reconf-applied message to the related components. Each related component sends back an apply-time in response. Consequently, AL.PDP$_i$ and AL.PDP$_j$ write an instance of the APDP$_k$ topic, which consists of three fields: domain-name (DN), identity (ID), and apply-time (AT). Note that AL.PAP component receives apply-times of the related PEP, PDP, PIP, and PLP components via the APEP, APDP, APIP, and APLP topics, respectively. AL.PAP returns the minimum of all received apply-times for each reconf-information to the PAP component. This returned value is referred as base-time. The decision and information requests whose sending-times are greater than the base-time for a new reconf-information are considered as new requests. Hence, the related PDP and PEP components that have already applied the new reconf-information should reply new requests, otherwise, an inconsistency problem may occur. By setting the values of latency-budget and transport-priority of topics according to Table 3 and formula Equation 3 and Equation 4, it is expected that the mentioned inconsistency problem is decreased. As mentioned before, the combining-algorithms will resolve this inconsistency problem, as well.

A reconf-information may cause revocation of some roles from an authorization component. Based on the mechanism used in OASIS, the AL layer revokes the dependent RMCs issued by the PAP components of other domains.

4.2 Enforcing Access Control Policies

The PEP components are mainly responsible for enforcing ABAC-policies, based on decisions made by the PDP components. To make decisions, the PDP components may need the information provided by the PIP components.

The PAP components define the connection points between PEP and PDP components by creating the DREQ and DREP topics, for making the final-decision about requests which are received from subjects. The process of making the final-decision is called decision-making process. Similarly, the PAP components define the connection points between PDP and PIP components by creating the IREQ and IREP topics, for providing the needed information. The process of providing the needed information is called information-providing process.

The PAP components define the connection points between PEP and PDP components by creating the DREQ and DREP topics, for making the final-decision about requests which are received from subjects. The process of making the final-decision is called decision-making process. Similarly, the PAP components define the connection points between PDP and PIP components by creating the IREQ and IREP topics, for providing the needed information. The process of providing the required information is called information-providing process. The PAP components define the DREQ, DREP, IREQ, and IREP topics, based on the attribute dependencies of components. The creation of these topics may be ignored if there is not any dependency between components. These attributes can be attributes of ABAC-policies plus any attribute defined by the PAP components. For example, a PAP component can assign a binary field as a new attribute of components, to show the sensitivity-level of the components. The components with different sensitivity-levels have no attribute dependency. Since the attribute dependencies are changed dynamically, the related topics must be reconfigured, subsequently. This sort of changes increases the complexity and cost of administration. To reduce this complexity and cost, the AC middleware uses dependency-recognition processes, by which the PDP and PIP components recognize attribute dependencies.

4.2.1 Dependency Recognition Process

Two types of dependency between authorization components are as follows. The first type is named attribute-dependency, which refers to dependency between PDP and PIP components. This dependency is recognized by the PIP components, based on their provided-attributes and required-attributes of PDP components. Required and provided attributes are expressed in terms of the attributes of subjects and resources. For example in an e-bank system, owner and balance are two attributes of an account. An example of a dependency-recognition process for one PDP component and one PIP component, named PDP$_i$ and PIP$_j$, is shown in Figure 8 Figure 8. After joining of PDP$_i$ or every time that the required attributes of PDP$_i$ are changed, the new required attributes of PDP$_i$ are sent to AL.PDP$_i$. Similarly, after joining of PIP$_j$ or every time that the provided attributes of PIP$_j$ are changed, the new provided attributes of PIP$_j$ are sent to AL.PIP$_j$. By receiving the new required attributes, AL.PDP$_i$ writes an instance of the QPDP$_k$ topic, which consists of three fields: domain-name (DN), identity (ID) and required-attributes (RA). The instance of QPDP$_k$ is read by AL.PIP$_j$, and the required attributes of PDP$_i$ are compared with the provided attributes of PIP$_j$. If there is no attribute dependency between them, PDP$_i$ is added to the filter-list of PIP$_j$, which contains components that should be filtered by content-filtering of the IREQ$_k$ topic for PIP$_j$. Therefore, information-requests of PDP$_i$ are notsent to AL.PIP$_j$. A PDP component is removed from the filter-list of a PIP component, if an attribute dependency between the filtered component and the PIP component is recognized in the next comparison. By receiving new provided attributes of PIP$_j$, the new provided attributes of PIP$_j$ and the last required attributes of PDP$_i$ are compared to update filter-list.

When PDP_i wants to leave the domain, $\mathrm{AL.PDP}_i$ writes an instance of QPDP_k, whose RA field is empty. It means that $\mathrm{AL.PIP}_j$ must remove PDP_i from the filter-list and delete its previous required attributes.

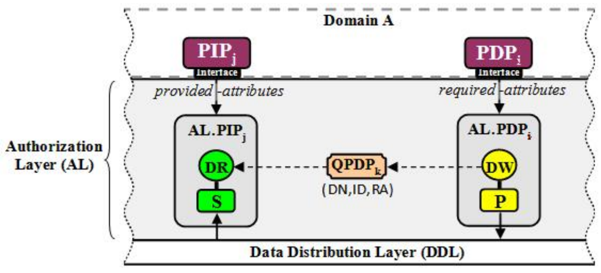


Figure 8. An example of a *dependency-recognition* process for one PDP component and one PIP components.

The second type of dependency is named action-dependency, which refers to dependency among PEP and PDP components. This dependency is recognized by the PDP components, based on their provided-actions and required-actions of PEP components. Required and provided actions are expressed in terms of the actions on resource types of a policy-domain-model.

For example, in the policy- domain-model of an e-bank system, deposit and withdraw are two actions on an account. The dependency-recognition processes in two types of dependency are essentially the same, while in the second type, the QPDP topics are used instead of the QPEP topics.

4.2.2 Decision Making and Information Providing Processes

When a PEP component receives a request from a subject to access some resources, the decision-making process is started to make the final-decision about the request.

An example of a decision-making process for one PEP component, named PEP_m , and two PDP components, named PDP_i and PDP_i, is shown in Figure 9. When PEP_m receives a request from a subject, a decision-request is sent from PEP_m to $\mathrm{AL.PEP}_m$, and then $\mathrm{AL.PEP}_m$ writes an instance of the DREQ_k topic. This topic consists of three fields: domain-name (DN), identity (ID), decision-request (RQ). If PEP_m is not in the filter-list of PDP_i and PDP_j, the instance of DREQ_k, which has been written by PEP_m, is read by $\mathrm{AL.PDP}_i$ and $\mathrm{AL.PDP}_j$. Consequently, the decision-request is passed to PDP_i and PDP_j and each PDP component sends back a decision-reply in response, based on its ABAC-policies and provided information. Both $\mathrm{AL.PDP}_i$ and $\mathrm{AL.PDP}_j$ writes an instance of the DREP_k topic in reply to PEP_m. The replied topic consists of three fields: domain-name (DN), identity (ID), decision-reply (RP). Since DREP_k has content-filter, the instance of DREP_k is received only by $\mathrm{AL.PEP}_m$. Based on decision-combining-algorithm of PEP_m, the final-decision is made and sent to PEP_m.

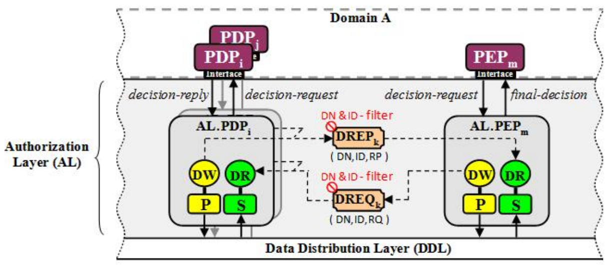


Figure 9. An example of a *decision-making* process for one PEP component and two PDP components.

Referring to Table 3, the durability value of the DREQ and DREP topics is set to volatile value, and the lifespan value of them is set to a positive integer value, called LS. This value shows the maximum duration that a decision-request or decision-reply is valid, starting from the write operation of each of them.

When a PDP component receives a decision-request from a PEP component, an information-providing process is started to provide the needed information. The process of decision-making and Information-providing are essentially the same, while in the later, the IREQ and IREP topics are used instead of the DREQ and DREP topics, respectively.

4.3 Logging of Instances

The PLP components are responsible for logging instances of the QPEP, DREQ, DREP, QPDP, IREQ, and IREP topics. The PAP components determine that which of these topics should be subscribed and logged by each PLP component. The logged instances may be audited and the result of audition could be used in new decision-making processes, while the AC middleware is in progress.

5 An Executable Model of the AC Middleware

The OASIS policies and the *conf* and *reconf* information are defined by administrators. These definitions should be correct and consistent, and make the behavior of the AC middleware more efficient. Evaluation of the new definitions and their effects on behavior and performance of the AC middleware is a huge and complicated work for administrators. Therefore, an executable model of the AC middleware helps the administrators to see whether the new definitions are correct, consistent and efficient. Constructing a CPN model of the AC middleware could be a promising approach to provide such an executable model.

CPN is a well-proven formal language for constructing models of complex systems and analyzing their properties [35, 36]. A CPN model is an executable representation of a system consisting of the states of the system and the events that cause the system to change its state. By the CPN model, it is possible to analyze different scenarios and explore the behavior of systems [35]. CPN Tools [37] is a visual and graphical software tool that is applied to create, edit, simulate, and analyze the CPN models.

Enterprise-integration patterns are appropriate to design a middleware system conceptually, before actually implementing it. Fahland *et al.* [38] introduced an approach to translate each of these patterns into a CPN model. The approach is used to investigate middleware system designs in early stages of development. In another article, Abidi *et al.* [39] proposed publish-subscribe models using CPN Tools. We use these two works to represent an executable model of the AC middleware. The model consists of a main page which represents the overall form of a domain of the AC middleware, and several related subpages. The needed input files are created by the administrator of the domain and will represent the initial marking of the CPN model. The input files contain the components that want to join the domain, the rules of OASIS policies, and the elements of *conf* and *reconf* information. Lets consider a simple scenario of a domain with the input files containing the fields that are shown in Table 5. Snapshots of execution of this scenario by the CPN tools are displayed in the following figures. The main page of the proposed model, called MAIN, is shown in Figure 10.

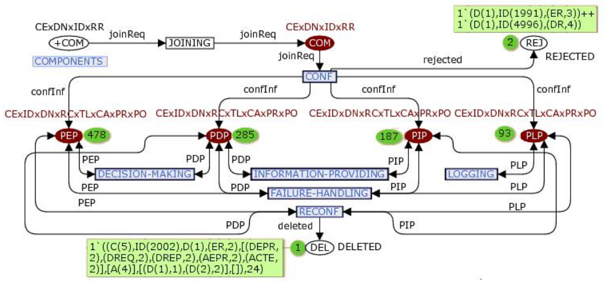


Figure 10. The MAIN page.

This page has six substitution transitions, named CONF, RECONF, DECISION-MAKING, INFORMATION-PROVIDING, LOGGING, and FAILURE-HANDLING, which are associated with six subpages with the same name. A short explanation of selected part of some subpages is given in this paper to have a taste of corresponding executable model. Each substitution-transition is similar to a function-call, while the body of that function is represented in the corresponding subpage. The main page

Table 5. Fields of input files of a simple scenario.

Fields of input files	Number of instances of each field
Authorization component	1046
OASIS rule	320
Topics-list	320
ABAC-policy	100
Combining-algorithm	120
Priority	120
Reconf-information	20

helps the administrator to monitor and analyze the components that have joined and left the domain, as well as the components whose join-requests have been rejected. The number of tokens for the PEP, PDP, PIP, and PLP places are shown by the circled numbers next to each place. These circled numbers indicate that the domain currently has 478, 285, 187, and 93 instances of PEP, PDP, PIP, and PLP components, respectively. By clicking on each of these circled numbers, the content of tokens can be seen, which consists of eight fields: certificate, identity, domain-name, role-certificate, topics-list, combining-algorithms, priorities, and policies. Each token of the REJ place consists of three fields: domain-name, identity, and requested-role of the components whose join-requests are rejected. The circled number for the REJ place shows that the join-requests of two components have been rejected. For example, the first token of the REJ place is (D(1),ID(1991),(ER,3)), which shows its three corresponding fields. Each token of the DEL place consists of nine fields: certificate, identity, domain-name, role-certificate, topics-list, combining-algorithms, priorities, policies, and time-stamp. The circled number for this place shows that just one PEP component has left the domain, at time-stamp 24.

The CONF subpage is shown in Figure 11. This subpage is used to simulate the conf process of the AC middleware. The subpage has three substitution-transitions, named JOIN-PUBLISHING, CONFIGURING, and CONF-PUBLISHING. The JOIN-PUBLISHING transition specifies the active-roles of components, which are needed to make join instances.

The CONFIGURING transition checks the join-requests based on the OASIS policies and prepares the related conf-information. The CONF-PUBLISHING transition sends the conf-information to the AL layer of components and updates the active-roles of the components.

The circle number for the SUB place shows that four components have currently subscribed to the CONF topic, to receive the conf-information. The join

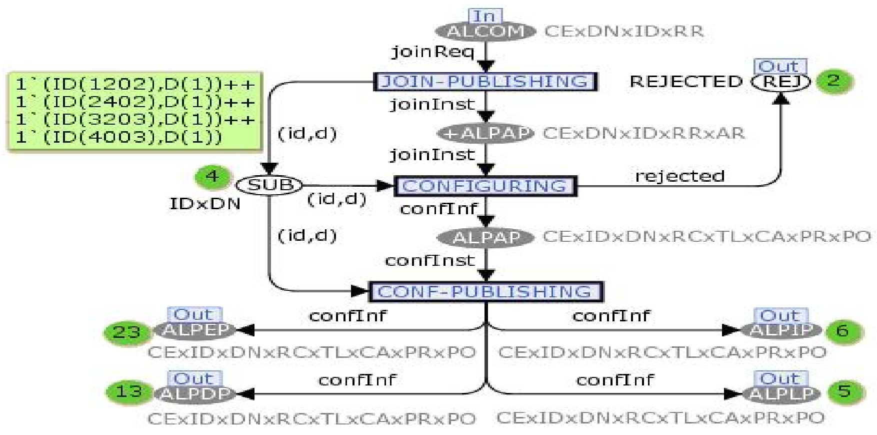


Figure 11. The CONF subpage.

instances that are received by the CONFIGURING subpage are processed based on OASIS policies, and the related conf information are sent to the ALPAP place. If a join instance is not acceptable, a rejected message is sent to the REJ place. The inconsistent and incomplete OASIS polices is detected if a join instance received by the CONFIGURING subpage does not exit from the subpage.

The RECONF subpage is shown in Figure 12. This subpage is used to simulate the reconf process of the AC middleware. The subpage has one substitution-transition, named RECONFIGURING. This transition first sends a reconf-information, read from input file, to the ALPAP place and receives the apply-times of that change from the APPLYTIME place, consequently. In one page of CPN we cannot have places with the same name. Hence, to have several places with the same concept, we annotate the corresponding name of the concept by one or more than one '+' at the beginning of the name. The data produced and displayed by the RECONF subpage helps the administrator to monitor the effects of the changes on the authorization components and the whole authorization system.

The DECISION-MAKING subpage is shown in Figure 13. This subpage is used to simulate the decision-making process of the AC middleware. The subpage has five substitution-transitions, named D-REQUESTING, ED-RECOGNITION, DREP-MAKING, DREP-WAITING, and D-COMBINING. The D-REQUESTING transition sends the decision-requests of PEP components to the REQ place. The ED-RECOGNITION transition updates the list of dependencies between the PEP and PDP components, by receiving a new attribute from the RAPEP or PAPDP place, and the old list from the ED-DEPENDENCYLIST place. The DREP-MAKING transition receives tokens from the SPDP place and makes a decision-reply about the request field of each token, and the decision-reply is sent to the DREP place. The DREP-WAITING transition sets the maximum response time of decision-requests. The circled number for the SUB place shows that five decision-requests are waiting for decision-replies. When the response time expires, the DREP-WAITING transition sends the decision-requests to the UNSUB place, in order to make the final-decision for each request. The D-COMBINING transition makes the final-decision based on the replies received from the REPS place, using the decision combining-algorithm of the related PEP component, and the final-decisions are sent to the ALPEP place. The circled number for the ALPEP place currently shows that two final-decisions have been received. Each token of this place contains the attributes and elements of the components, such as identity, domain-name, combining-algorithms, policies and information. The data displayed by these tokens helps the administrator to find the inconsistent and incomplete conf and reconf elements. Data in these tokens are related to the components that have been used to make the final-decision.

Each token of ++ALPLP, +ALPLP, and ALPLP places contains the instance of QPEP, DREQ, and DREP topics, respectively. The *LOGGING* subpage uses these tokens to simulate the logging process.

The INFORMATION-PROVIDING transition in the main page is used to simulate the information-providing of the AC middleware. The INFORMATION-PROVIDING subpage is similar to DECISION-MAKING subpage. The FAILURE-HANDLING transition in the main page is also used to simulate the failures of the components, as well as handling of the failures.

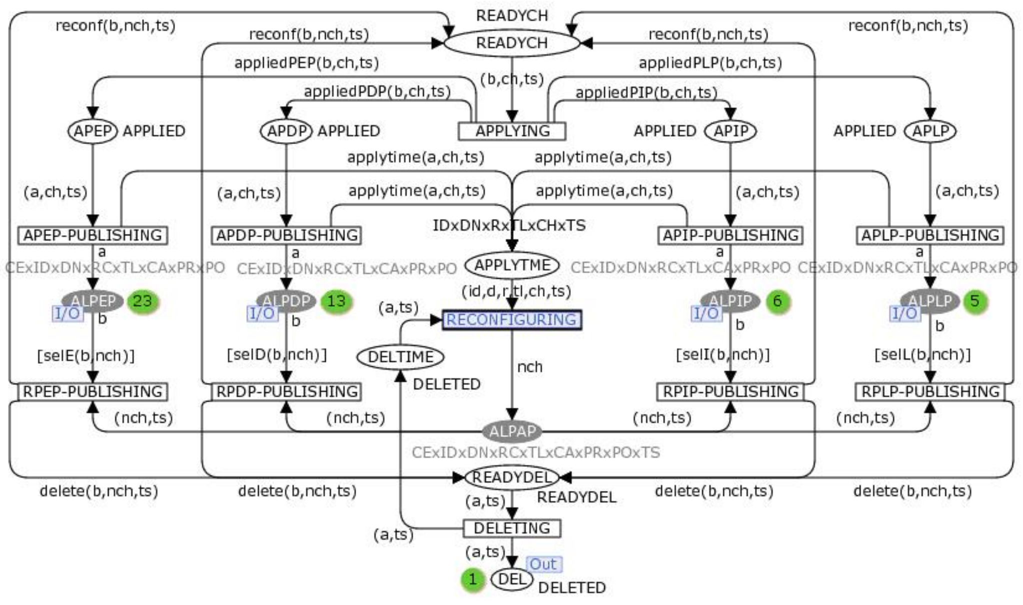


Figure 12. The RECONF subpage.

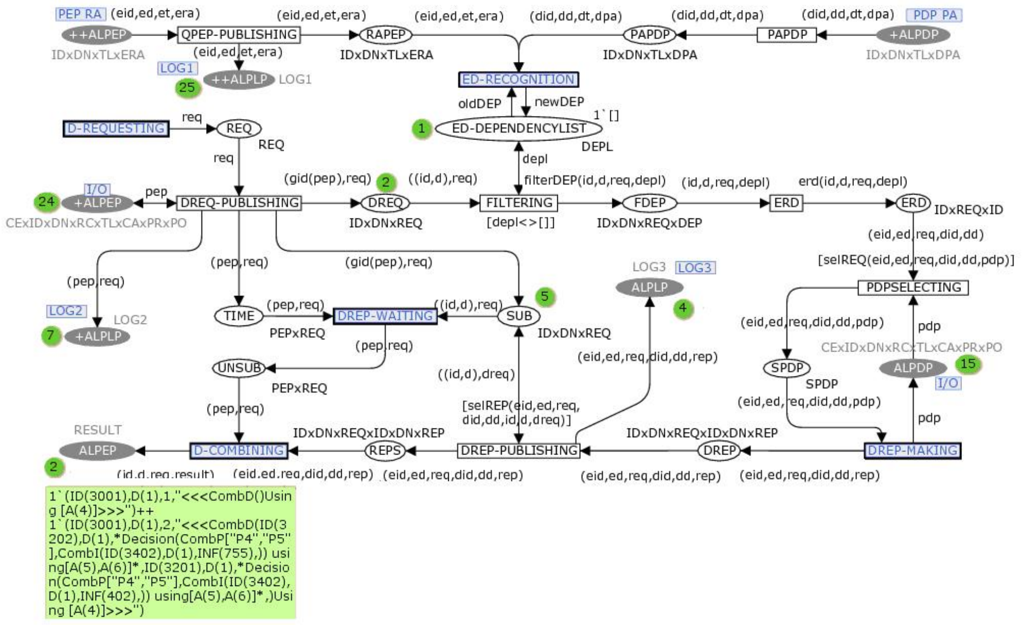


Figure 13. The DECISION-MAKING subpage.

6 Performance Analysis

In the CPN model of the AC middleware there is a subpage called RESPONSETIME. Execution of this subpage displays some parameters that could be used, interactively, to evaluate the response-time of access requests. The response-time of an access request is the total time that it takes from receiving the request by a PEP component until the component sends back the access response. This time depends on three issues.

(1) The maximum-response-time of decision requests, related to the received access request. Note that, the PEP component sends these decision requests to the related PDP components in order to make a final-decision. The response-time of a decision request is the total time that it takes from receiving the request by a PDP component until the component sends back the decision response.

(2) The total data-latencies of the DREQ and DREP topics that are used for sending decision requests and receiving decision responses. The data-latency of a topic is the amount of time that a data-sample of that topic travels from

Table 6. Response time of six access requests using RTI connext DDS

Number of PDP components	Latency of decision request	Latency of decision response	Maximum response time of decision requests	Execution-time	Response time of an access request
1	$91\mu s$	$91\mu s$	$250\mu s$	$20\mu s$	$453\mu s$
200	$97\mu s$	$92\mu s$	$250\mu s$	$20\mu s$	$459\mu s$
400	$98\mu s$	$92\mu s$	$250\mu s$	$21\mu s$	$461\mu s$
600	$101\mu s$	$92\mu s$	$250\mu s$	$21\mu s$	$464\mu s$
800	$102\mu s$	$92\mu s$	$250\mu s$	$22\mu s$	$466\mu s$
888	$103\mu s$	$92\mu s$	$250\mu s$	$22\mu s$	$467\mu s$

its publisher to its subscriber. This latency depends on the number of its subscribers, the size of its data-samples, and the publication rate of its data-samples. Note that, the decision response times of PDP components include the data-latencies of the IREQ and IREP topics.

(3) The execution-time of combining algorithms and other related activities, such as exchanging messages, used in the AL layer.

Multiple implementations of DDS, such as RTI connext TM DDS [40], are now available. RTI connext DDS analysis demonstrates that the number of subscribers has no significant impact on sustainable throughput of a topic. Going from 1 to 888 subscribers has less than 13% impact on maximum throughput of a topic. Some parts of analysis of RTI connext DDS 4.3 [41] have been used in our CPN model, to simulate six access requests. The result of this simulation is shown in Table 6 . In this simulation, the rate of decision requests or the rate of decision responses is considered 100000 requests or responses per second. The size of each request or response is 200 bytes. The maximum number of subscribers is 888. The rate, size, and the number of subscribers used in our simulation are equal to those in the RTI's analysis. These equalities allow us to use the latency values reported by RTI. The table shows that going from 1 to 888 PDP components in the AC middleware has less than 4% impact on the response time of access requests.

In RTI connext DDS, new topics can be added to a system without impacting latency on other topics. So the topic-multiplicity attribute in the DDL layer can improve the performance of the AC middleware. RTI connext DDS also provide writer-side content-filtering attribute, in which a data-sample of a topic will not be sent to a subscriber if that topic has been filtered by the subscriber in advance. In this case, the publication of unnecessary data-samples is reduced, and the performance of the AC middleware is increased.

7 Conclusion and Future Work

The proposed AC middleware is a DDS-based access control middleware that makes ABAC model more suitable for a ULS system. A large number of authorization components can join the middleware dynamically and communicate with each other securely and anonymously. The middleware supports *multi policy*, *multi decision*, and *multi information* capability for a single domain or among several domains. The middleware provides flexible and loosely-coupled dependencies among authorization components. The capability of dynamic and correct*reconf* of components is also supported by the middleware. The integration of new types of components into the middleware is done easily. An executable model of the middleware is also represented by a hierarchical CPN model, using CPN Tools. The data produced and displayed by the executable model helps the administrators to evaluate the new definitions and their effects on behavior and performance of the middleware.

Currently, we are designing a rich service for access control, which exploits ideas applied in the proposed middleware. We also generalized this middleware to be used for integration of components in a ULS system.

8 Acknowledgment

The corresponding author thankfully acknowledges the helpful discussions with Professor Pierangela Samarati and her valuable comments on the paper. The author also likes to appreciate the hospitality and technical cooperation of Prof. Samarati's group at "Security, Privacy, and Data Protection Laboratory", Computer Science Department of University of Milan, Crema, Italy.

References

[1] L. Northrop et al., "Ultra-Large-Scale Systems: The Software Challenge of the Future," Carnegie Mellon Software Engineering Institute, Ultra-Large-Scale Systems Study Report, 2006.

[2] D. Bell and L. LaPadula, "Secure computer system: Unified exposition and multics interpretation," Technical Report ESD-TR-75-306, The Mitre Corporation, March 1976.
[3] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in Operating Systems," Communications of ACM, vol.19, no.8, pp.461-471, ACM, 1976.
[4] D. Ferraiolo and R. Kuhn, "Role-Based Access Control," In Proceedings of 15th NIST-NCSC National Computer Security Conference, pp. 554-563, Baltimore, MD: ACM, 1992.
[5] R.S. Sandhu, E.J. Coyne, H.L. Feinstein and C.E.Youman, "Role-based Access Control Models," IEEE Computer, vol. 29, no. 2, pp. 38-47, IEEE 1996.
[6] A.R. Khan, "Access control in cloud computing Environment," ARPN Journal of Engineering and Applied Science, vol.7, no.5, pp 613- 615, PK: ARPN, May 2012.
[7] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "Attribute based access control for grid computing," Math. Comput. Sci. (MCS) Div., Argonne Nat. Lab., Argonne, IL, Preprint ANL/MCS-P1367-0806, August 2006.
[8] T. Priebe, W. Dobmeier, and N. Kamprath, "Supporting Attribute-based Access Control in Authorization and Authentication Infrastructures with Ontologies," Journal of Software, vol.2, no.1, pp.27-38, Academy Publisher, February 2007.
[9] S. Verma, S. Kumar, and M. Singh, "Comparative analysis of Role Base and Attribute Base Access Control Model in Semantic Web," International Journal of Computer Applications, vol.46, no.18, pp.1-6, USA: FCS, May 2012.
[10] Feng Liang, Haoming Guo, Shengwei Yi, and Shilong Ma, "A Multiple-Policy supported Attribute-Based Access Control Architecture within Large-scale Device Collaboration Systems," Journal of Networks, vol.7, no.3, pp.524-531, Academy Publisher, March 2012.
[11] Tom Goovaerts, Lieven Desmet, and Wouter Joosen, "Scalable Authorization Middleware for Service Oriented Architectures," In Proceedings of the Third international conference on Engineering secure software and systems (ESSoS'11), February 2011, Springer-Verlag, pp.221-233.
[12] S. Singh, K. Singh, and H. Kaur, "Design and Evaluation of Policy Based Authorization Model for large scale Distributed Systems," International Journal of Computer Science and Network Security (IJCSNS), vol.9, no.11, pp.49-55. Nov 2009.
[13] E. Damiani, S. De Capitani di Vimercati, and P. Samarati, "New Paradigms for Access Control in Open Environments," In Proceeding of 5th IEEE International Symposium on Signal Processing and Information, 2005, Greece:IEEE, pp. 540-545.
[14] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, and P. Samarati, "Access Control Policies and Languages," International Journal of Computational Science and Engineering (IJCSE), vol. 3, n. 2, pp. 94-102, Inderscience Publishers, 2007.
[15] Object Management Group (OMG), "Data Distribution Service for Real-Time Systems Specification," March 2004.
[16] R. Joshi, "Data oriented Architecture: A loosely Coupled Real time SOA," Real-Time Innovations, Inc, CA, Tech. Rep. Aug 2007.
[17] R.Joshi, "Data-Centric Invocable Services: A Core Design Pattern for Building Scalable Distributed Real-Time Systems," In Proceedings of the 2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, Dec 2012, IEEE Computer Society, pp.1-7.
[18] Q. Wei, M. Ripeanu, and K. Beznosov, "Authorization Using the Publish-Subscribe Model," In Proceedings of IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2008), , December 2008, NSW, IEEE, pp.53-61.
[19] J. Bacon, K. Moody, and W. Yao, "Access control and trust in the use of widely distributed services," In Middleware 2001, Lecture Notes in Computer Science 2218, pp. 295-310, Springer, November 2001.
[20] J. Bacon, K. Moody, and W. Yao, "A Model of OASIS Role-Based Access Control and its Support for Active Security," ACM Transactions on Information and System Security (TISSEC), vol.5, no.4, pp.492-540, USA: ACM, Noveber 2002.
[21] L. I. Pesonen, D. M. Eyers, and J. Bacon, "Access control in decentralised publish/subscribe systems," Journal of Networks, vol.2, no.2, pp.57-67, Academy Publisher, 2007.
[22] E. Yuan, J.Tong, "Attribute Based Access Control (ABAC) for Web Services," In Proceeding of IEEE Conference on Web Services (ICWS 2005), 2005, IEEE, pp.561-569.
[23] A. Belokosztolszki, D. M. Eyers, P. R. Pietzuch, J. Bacon, and K. Moody, "Role-based access control for publish/subscribe middleware architectures," In Proceeding of international workshop on Distributed event-based systems, 2003, USA: ACM, pp.1-8.
[24] J. Bacon, D. M. Eyers, J. Singh, and P. R. Pietzuch, "Access control in publish/subscribe systems," In Proceeding of 2nd International. Conference on Distributed event-based systems (DEBS),

2008, USA: ACM, pp.23-34.

[25] J. Bacon ,D. M.Eyers, K. Moody, and L. Pesonen, “Securing publish/subscribe for multi-domain systems,” In Proceeding of 6th International Middleware Conference, vol.3790, 2005, Springer, pp.1-20.

[26] N. Wang , D. Schmidt, H. Hag, and A. Corsaro, “Toward an adaptive data distribution service for dynamic large-scale network-centric operation and warfare (NCOW) systems,” In Proceedings IEEE Military Communications (MILCOM’8), 2008, IEEE, pp.1-7.

[27] Jose M. Lopez-Vega, Javier Povedano-Molina, Gerardo Pardo-Castellote, Juan M. Lpez-Soler, “A content-aware bridging service for publish/-subscribe environments,” Journal of Systems and Software, vol.86, no.1, pp.108-124, Elsevier, 2013.

[28] RTI, “Real-Time Innovations (RTI) DDS Data Distribution Service”, http://www.rti.com/.

[29] Sung Yoon Chae, Sinae Ahn, Kyungran Kang, Jaehyuk Kim, Soohyung Lee, and Won-tae Kim, “Fast Discovery Scheme Using DHT-Like Overlay Network for a Large-Scale DDS,” In Proceedings of FGIT-CA/CES3, 2011, Springer, pp. 128-137.

[30] Y.H. Long, Z.H. Tang, and X. Liu, “Attribute mapping for cross-domain access control,” In Proceedings of Computer and Information Application (ICCIA), 2010, IEEE, pp. 343-347.

[31] OASIS. extensible access control markup language (xacml) version 3.0. Technical report, OASIS Standard, 2010. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf.

[32] C. Ardagna, S. De Capitani di Vimercati, S. Paraboschi, E. Pedrini, P. Samarati, and M. Verdicchio, “Expressive and Deployable Access Control in Open Web Service Applications,” IEEE Transactions on Service Computing (TSC), vol. 4, n. 2, pp. 96-109, IEEE, April-June 2011.

[33] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati, “An Algebra for Composing Access Control Policies,” ACM Transactions on Information and System Security (TISSEC), vol. 5, n. 1, pp. 1-35, ACM, February 2002.

[34] N. Li, Q. Wang, W. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin, “Access Control Policy Combining: Theory Meets Practice,” In Proceeding of 14th ACM Symposium on Access Control Models and Technologies, 2009, ACM, pp. 135-144.

[35] W.M.P. van der Aalst and C. Stahl, Modeling Business Processes - A Petri Net-Oriented Approach, Cambridge, MA: MIT press, 2011.

[36] K. Jensen, L.M. Kristensen, and L. Wells, “Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems,” International Journal on Software Tools for Technology Transfer (STTT), vol.3, no.4, pp. 213-254, Springer, 2007.

[37] CPN Tools, http://cpntools.org/.

[38] D.Fahland and C.Gierds, “Analyzing and Completing Middleware Designs for Enterprise Integration Using Coloured Petri Nets,” In Proceedings of Conference on Advanced Information Systems Engineering (CAiSE 2013), 2013, Springer Berlin Heidelberg, pp.400-416.

[39] L. Abidi, C. Crin, and S. Evangelista, “A Petri-Net Model for the Publish-Subscribe Paradigm and Its Application for the Verification of the BonjourGrid Middleware,” In Proceedings of IEEE Services Computing (SCC), 2011, IEEE, pp.496-503.

[40] “RTI Data Distribution Service”, http://www.rti.com/products/dds/.

[41] RTI. Connext DDS. http://www.rti.com/products/dds/ benchmarks- cpp-linux-scalability.html.

Saeed Shokrollahi is a Ph.D. candidate at Computer Engineering Department of Shahid Beheshti University, Tehran, Iran. He spent his sabbatical leave at “Security, Privacy, and Data Protection Laboratory”, Computer Science Department of University of Milan, Crema, Italy. His current research interests are Information Security and Access Control, Ultra-large-scale Systems, Event-driven Middleware, Software Architecture, Software Testing, and Model Verification Using Colored Petri Nets.

Fereidoon Shams is an associate professor of Computer Engineering Department at Shahid Beheshti University, Tehran, Iran. He received his Ph.D. in Software Engineering from Department of Computer Science, Manchester University, England and his M.S. from Sharif University of Technology, Tehran, Iran. He is the heading of two research groups namely ASER (Automated Software Engineering Research) and ISA (Information Systems Architecture) at Shahid Beheshti University. His current research interests are Software Architecture, Enterprise Architecture, Service Driven Architecture, Agile Methodologies, Ultra-Large-Scale Systems, Ontological Engineering and Security.

Javad Esmaeili is an assistant professor at the Department of Computer Engineering at Shahid Beheshti University, Tehran, Iran. He received his Ph.D. degree in Parallel Processing, Design and Implementation of Functional Languages, from University of Salford, Greater Manchester, England. He is a member of Computer Society of Iran. His current research interests are Distributed and Parallel Processing, System Analysis, Design and Implementation of Programming Languages, E-banking, Security and Access Control.