

Regular Paper

Policy Provisioning and Its Access Control Beyond Administrative and Collaborative Domains

HIDEHITO GOMI^{1,a)}

Received: April 11, 2011, Accepted: September 12, 2011

Abstract: A policy provisioning framework is described that supports management of the lifecycle of personal information and its data-handling policies distributed beyond security domains. A model for creating data-handling policies reflecting the intentions of its system administrator and the privacy preferences of the data owner is explained. Also, algorithms for systematically propagating and integrating data-handling policies from system entities in different administrative domains are presented. This framework enables data-handling policies to be properly deployed and enforced in a way that enhances security and privacy.

Keywords: policy provisioning, identity management, access control, policy management, data-handling policies

1. Introduction

Many applications are being executed in distributed systems and among different multiple organizations with the ongoing development of the Internet. Personal information in such environments is often exchanged beyond the boundaries of security domains. It is generally difficult for both administrators and owners to control data once they have been propagated outside their security domains. Thus, managing identity and policy in distributed environments is one of the most important issues in preserving security and privacy.

There have been several technical projects on identity management including access control and privacy management. Privacy-aware access control [1], [2], [3] has especially aimed at incorporating privacy-related policies into traditional access-control policies. Another emerging concept of *identity governance* [4] has addressed user-centric control of access and introduced a method of tracking data for propagating identity information. Although these research efforts have enabled fine-grained access control for managing identity from security and privacy perspectives, they have not fully addressed how data-handling policies can be created and integrated, which satisfy the different requirements of distinct actors in different security domains from a practical viewpoint. Since these actors are involved with data practices, but generally have different responsibilities, enforcing policies need to be created from administrative and privacy standpoints. Another existing work on secure interoperation in a multi-domain environment has proposed schemes for generating an inter-domain security policy to grant user access from different security domains [5], [6]. However, these approaches lack support of lifecycle management of security policies in each domain and the notion of privacy protection.

This paper proposes a policy provisioning framework that helps to manage the lifecycle of identity information using handling policies that reflect its system administrator's intentions and its data owner's preferences from both administrative and privacy viewpoints. Also described are algorithms that enable data-handling policies or privacy preferences to be created and integrated from multiple actors to control access to identity information. This work focuses on collaboratively building a policy provisioning model and framework for distributed policy and identity management systems, whereas the specific representation of policies, detailed resolutions on policy conflicts, and the encapsulation and transport mechanisms for data and policies are beyond the scope of this work.

It was assumed that each system entity in this work had the distinct responsibility for managing data and policies, but collaboratively exchanged them with one another and handled them in conformity with the policies that they agreed upon to establish a foundation for improving identity governance from the viewpoints of administrators and users in the system.

The rest of this paper is organized as follows. Section 2 presents problems and requirements from the scenario that motivated this work. Section 3 introduces a policy provisioning model. Section 4 describes a policy provisioning framework based on the proposed model. Section 5 presents a case study of the proposed framework. Section 6 discusses several issues related to policy management and Section 7 presents related work. Section 8 concludes the paper with a summary of the key points and an outline of future work.

2. Problem and Requirement Statements

This section describes problems that need to be addressed when developing a framework to support policy provisioning between multiple domains, and this leads to several requirements.

Figure 1 outlines the scenario that was the motivation behind

¹ Yahoo! JAPAN Research, Minato, Tokyo 107-6211, Japan

^{a)} hgomi@yahoo-corp.jp

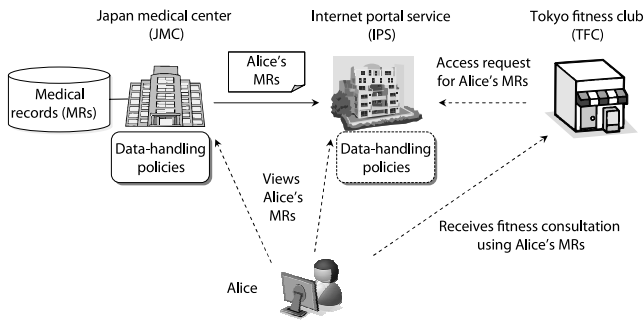


Fig. 1 Motivating scenario.

this work. Alice is a woman who had a medical examination at the Japan medical center (JMC), which manages her medical records (MRs) according to its data-handling policies. Alice can access and view her own MRs at JMC. Since Alice has an active account at the Internet portal service (IPS), which is run by a different private company, but is in a coalition relationship with the JMC, she would like to store and view a subset of her MRs at the IPS propagated from the JMC for the sake of convenience. In addition, Alice would like to use the MRs stored at the IPS to take advantage of a fitness consultation service given by the Tokyo fitness club (TFC), which has a trusted relationship with the IPS. Alice hopes to use her own MRs for her personal benefit although she is also concerned about them being disclosed to unauthorized entities because they contain sensitive and private information. Hence, the JMC, the IPS, and the TFC need to collaboratively manage and handle Alice's MRs in a manner that is secure and that preserves her privacy.

In this case, how does the administrator of the IPS develop its data-handling policies, which are represented by the rounded rectangles pointed to by the dotted lines with arrows in Fig. 1, to manage and handle Alice's MRs if the JMC's data-handling policies are given? To fulfill this scenario according to Alice's wishes, the IPS needs to have data-handling policies including those that grant access requests to Alice's MRs from the TFC as well as obtain her consent for doing so. However, the TFC's use of the MRs is secondary from the viewpoint of the global system because it was originally managed by the JMC and was propagated to the IPS. If the JMC has its own policies on constraining data disclosure, i.e., denying the propagation of MRs to the TFC, and the IPS agrees upon the policies when the MRs are received from the JMC, then the IPS must conform to the agreed-upon policies to retain and handle them. In addition, the JMC and the IPS may have common policies that they must comply with if they are in the same company group and the IPS depends on the JMC's formulating policies. That is, the IPS's data-handling policies should be developed from the perspectives of the organization or system it belongs to and the relationships with entities it interacts with.

The above observation leads to a set of technical issues and requirements that need to be addressed by the model and framework that is proposed in this paper.

(1) **Policy integration and incorporation.** Data-handling policies need to be developed by reflecting on the requirements and intentions arising from different actors who are asso-

ciated with creating and using the data. This may create some conflict with their individual requirements or intentions. Therefore, any conflicts need to be avoided to generate consistent data-handling policies for each system entity from the global perspective of the overall system.

(2) **Lifecycle management of policies.** Data need to be kept under control at an entity accepting propagation from the entity that originally managed it in an environment where personal data are propagated in system entities distributed over network. Since data propagation may successively occur from entity to entity according to its associated policies and user wishes, the system needs to support lifecycle management of the policies to persistently control data wherever they are located in the system.

In light of the above requirements, a model and a framework for policy provisioning were designed according to the four underlying concepts:

- **Persistently sticky policies.** This concept enables personal data to be kept under control even after they have been propagated beyond distinct domains in a distributed environment. Although previous research efforts [7], [8] introduced the "sticky policy" concept, they only supported a single step for data propagation, not multiple steps from a broader perspective.
- **Hierarchical policies.** Policies to be enforced with this concept are incorporated into a hierarchical organization or system.
- **A chain of policies.** This concept represents a flow of policies propagated in distinct domains.
- **Globally identifiable policies.** This concept enables policies to be tracked in a distributed environment to keep them and their associated data under control.

These concepts are introduced in the proposed model, which will be described in Section 3. Then, the proposed framework for provisioning policies between system entities based on the model will be described in Section 4 by specifying policy-related operations common to system entities such as policy exchange between them and procedures for handling policies within an individual entity to make the overall system secure.

3. Model

This section explains the model for policy provisioning.

A **data user (DU)** is an individual whose personal data are related to him or her. A DU delegates secure management and convenient utilization of his or her personal data to other entities specifying privacy preferences on how the data are to be handled by them. A DU can demonstrate his or her wishes by means of *consent* to questions from other entities as one of the representations of privacy preferences.

A **data controller (DC)** is an entity that maintains a DU's personal data on his or her behalf in compliance with its own privacy or security policies reflecting both their privacy preferences. A DC can securely provide a DU's personal data to another entity in a different administrative domain on the basis of the agreement with the entity on how the data are to be used and handled. A DC is liable for securely managing and propagating a DU's personal

data.

A **data processor (DP)** is an entity that processes a DU's personal data obtained from a DC in conformity with the agreement reached with the latter on how the data are to be handled. A DP is placed in a distinct administrative domain from that of a DC. A DP is liable for handling personal data originally managed by a DC. This liability is different from that for a DC since a DP does not need to maintain or determine the purposes for which the data are processed, and because this liability depends on the agreement on data processing between a DC and a DP.

The DC and DP are not actual entities; they are simply roles in the model. Therefore, a single domain can play both roles. That is, when domain d_1 acting as a DP receives personal data from domain d_2 acting as a DC with an agreed-upon policy allowing d_1 to store and further propagate the received data to the other domain, d_3 , d_1 can act as a DC for d_3 . In other words, the relationship between DC and DP is relative and one-way specific to the pair of two entities for the particular types of personal data in this model. In this example, d_1 can be a DC for d_3 , but cannot be a DC for d_2 , because d_2 was originally a DC for d_1 .

It is assumed that these entities are trusted and can be expected to comply with the agreement. The purpose of this work was to establish an agreement on data handling between trusted entities and create and deploy policies to be appropriately enforced, rather than to detect their misbehaviors.

3.1 Policy Binding to Data

Data and their handling policies in this model are very closely associated. When a DC receives an access request to data, the DC determines whether to grant or deny the access enforcing the handling policies associated with the data. When a DP attempts to process data, the DP also uses the handling policies associated with the data to make an authorization decision on data processing.

If a DC needs to provide a DP with personal data, the DC encapsulates the data and their associated policies and transfers both to the DP. The DP complies with the policy agreed upon and received from the DC prior to having received the data. Namely, agreed-upon policies migrate with the data to govern the data practices of a DC that receives both the data and policies. The agreed-upon policies correspond to an agreement between a DC and a DP when data are transferred and these are the legal grounds for appropriately restricting data processing by a DP.

The encapsulation and transport mechanisms for data and policies are beyond the scope of this model. Instead, it focuses on the design of the framework for hierarchically developing policies among distinct entities, which will be described in the next section.

3.2 Policy Hierarchy

Figure 2 outlines a hierarchical policy model that encompasses the defined entities and exchanged policies.

A *super-domain* is a super-organization or system such as an industrial department or a governmental body to which *domains* belong. A domain is an administrative organization or system independent of others that acts as a DC or a DP entity. A super-

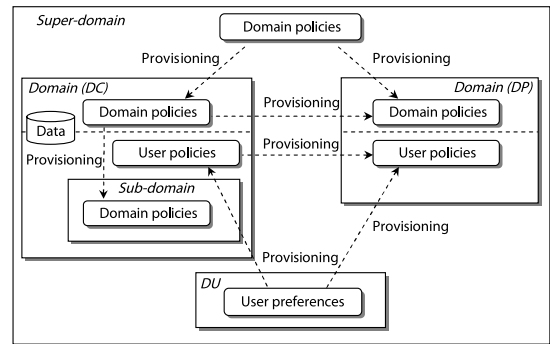


Fig. 2 Hierarchical policy model.

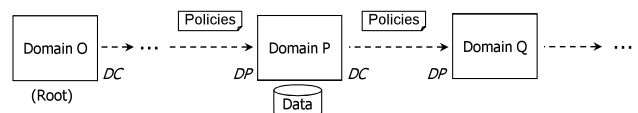


Fig. 3 Policy provisioning chain.

domain and a domain have a relative association. The *sub-domain* belongs to its upper class domain. The relationships between the super-domain and domain, and domain and sub-domain are hierarchical. These relationships generally hold true without limiting the representation in Fig. 2. A domain acting as a DC provides a DU with a service that manages the DU's personal data. A domain acting as a DP provides a DU with a service that uses a DU's personal data. It is assumed that a domain has only one super-domain.

A super-domain has *super-domain policies* that are meta-level and general constraining the activities of all domains that belong to the super-domain, by reflecting its laws or regulations with which the domains need to comply.

The *domain policies* are organizational domain-specific that inherit the super-domain policies in the super-domain, and are not specific to DUs. When a DC propagates personal data to a DP, the DC and the DP agree on a set of policies on how the data are to be used and handled prior to being propagated. The agreed-upon policies migrate with the data from the DC to the DP.

The *user policies* are specified by a domain for a DU, reflecting *user preferences*, which are a DU's privacy-related preferences for handling the DU's personal data. As a result of user preferences being incorporated into user policies, a DU needs to follow the user policies specified by domains to enjoy their services. More detailed descriptions on how these policies are created and provided will be given in the sections that follow.

3.3 Policy Provisioning Chain

A DU's data possibly propagate from domain to domain. Here, the data-handling policies associated with the data also propagate from a domain acting as a DC to a domain acting as a DP. The flow of the policies constitutes a chain of domains as seen in **Fig. 3**.

When domain P manages data, it obtains super-domain policies from its super-domain, and additionally agreed-upon policies if the data have originally been propagated by another domain acting as a DC to incorporate them into its domain policies. P, acting

as a DC, propagates the data and their handling policies to domain Q, acting as a DP, after P and Q have agreed upon the policies. As a result of data being propagated and policies being agreed on, Q becomes responsible for handling the data. In this way, if data propagate from domain to domain, their associated handling policies change to those reflecting local domain policies and they propagate together with data to enforce the behavior of another domain that receives the data. The first entity that provides policies in a provisioning chain is called the *root domain*, which is denoted by domain O in Fig. 3.

There are two types of relationships between adjacent domains in the policy provisioning chain. The first is a hierarchical relationship. Since the upper domain manages the lower domain in a policy hierarchy in this case, the lower domain inherits the policies from those of the upper domain. The second is a propagation relationship in different domains that are placed on the same level in the policy hierarchy. Here, policies are propagated from a DC to a DP after they reach agreement on how data are to be handled.

3.4 Policy Components

Let DI denote the set of domain identifiers that uniquely identify domains in the system and PI denote the set of policy identifiers that uniquely identify policies in a domain.

Definition 1 (Global policy identifier (GPI)). A GPI is a pair $\langle d_id, p_id \rangle$, where $d_id \in DI$ is the identifier of the domain that creates and issues a policy and $p_id \in PI(d_id)$ is the identifier of the corresponding policy in the domain.

A GPI is used to uniquely identify the corresponding policy in the proposed distributed system, where GPI represents the set of GPIs.

Next, the following definition is given as a component of a policy.

Definition 2 (Target). A target is a four-tuple $\langle domains, subjects, data, action \rangle$, where $domains \subseteq \{d | d \in DI\}$ represent the set of identifiers of domains in the system, $subjects \subseteq \{s | s \in (U \cup DI)\}$ are the subjects that consist of the union set of DUs U and domains DI , $data$ represent the target group of objects or resources in the domains that is accessed by the subjects, and $action \in A$ is a specific activity that invokes a function call on the data.

This model specifies a policy as the following definition.

Definition 3 (Policy). A policy is a four-tuple $\langle gpi, par_gpi, target, constraints \rangle$, where $gpi \in GPI$ is the GPI of this policy, $par_gpi \in GPI$ is the GPI of the parent policy on which basis this policy is created, and $target \in T$ is the target of the former policy.

3.5 Policy Classification and Examples

There are several classes of policies. There are two types of domain policies, i.e., *common policies* and *governance policies*. Common policies control the actions of DUs on data in corresponding domains while governance policies specify the actions and constraints of corresponding domains, not DUs.

Example (Common domain policies). A common domain policy statement in a natural language is “Any users are allowed, by the domain to which they belong, to propagate their personal attribute data to other users or domains if they show their explicit

consent to the propagation.” This statement can be specified in the form described in Section 3.4:

$$(P0.a) : \langle \langle d_0, p_{0.a} \rangle, _, \langle *, \{u_0\}, data, propagate \rangle, \\ \langle d_0 \in DI, p_{0.a} \in PI(d_0), u_0 \in U(d_0), \\ propagate \in A, att(u_0, data), consent(u_0) \rangle \rangle,$$

where “*” means that any sub-domains in domain d_0 are included in the target of this policy and axioms $att(u_0, data)$ and $consent(u_0)$ denote that $data$ represent the set of u_0 ’s personal attributes and that u_0 consents to the execution of the action ($propagate$). The “_” stands for the empty global policy identifier indicating that this policy is the root.

A domain in another common domain policy that is a digital content owner can state that no subscribers have any right to propagate the content in the following form:

$$(P0.b) : \langle \langle d_0, p_{0.b} \rangle, _, \langle \{d_0\}, \{u_0\}, content, \neg propagate \rangle, \\ \langle d_0 \in DI, p_{0.b} \in PI(d_0), u_0 \in U_s \subseteq U, \\ propagate \in A, own(d_0, content) \rangle \rangle,$$

where operator \neg specifies a logical negation, the set of DUs, U_s , denotes the subscribers of the content in the domain, and axiom $own(d_0, content)$ indicates that domain d_0 owns the content.

Governance policies are directives for managing the behavior of sub-domains from an administration point of view.

Example (Governance domain policies). Information on personal attributes such as name and address can ultimately be managed by the DU that owns them, assuming the concept is acceptable from the standpoint of local laws.

$$(P0.c) : \langle \langle d_0, p_{0.c} \rangle, _, \langle \{d_1\}, \{d_1\}, buying_history, retain \rangle, \\ \langle d_0, d_1 \in DI, p_{0.c} \in PI(d_0), d_1 < d_0, retain \in A, \\ retentionPeriod(3years) \rangle \rangle,$$

where *buying_history* denotes a DU’s buying history collected by domain d_1 as a result of providing its commercial service, operator “<” between domains indicates the hierarchical relation between them, and *retentionPeriod*(·) specifies the period during which d_1 may retain the history data so that domain d_0 restricts d_1 ’s actions in retaining DU’s data.

Another example

$$(P0.d) : \langle \langle d_0, p_{0.d} \rangle, _, \langle \{d_1\}, \{d_0\}, policies, publish \rangle, \\ \langle d_0, d_1 \in DI, p_{0.d} \in PI(d_0), d_1 < d_0, \\ publish \in A, create(d_0, policies) \rangle \rangle$$

specifies that domain d_1 grants d_0 ’s publishing message containing the policies created by d_1 . The d_0 can administer its d_1 to accept its policies by d_1 ’s deploying and enforcing the above (meta-level) policy (P0.d) distributed by d_0 .

The representation of domain policies can be changed into one specific to a domain to which the policies are propagated. The following policy example is created from domain policy (P0.a) as a baseline policy.

Example (Instantiated domain policies). A policy instantiating policy (P0.a) is

(P1.a) : $\langle\langle d_1, p_{1.a} \rangle, \langle d_0, p_{0.a} \rangle, \langle \{d_1\}, \{u_1\}, data, propagate \rangle,$
 $\langle d_0, d_1 \in DI, p_{0.a} \in PI(d_0), p_{1.a} \in PI(d_1),$
 $u_1 \in U(d_1), propagate \in A,$
 $att(u_1, data), consent(u_1) \rangle\rangle.$

In policy (P1.a), the administrator of this domain (d_1) limits the subjects of the target to DUs in domain d_1 whereas the target subjects of original policy (P0.a) are DUs in domain d_0 .

User policies in a domain are policies for a particular DU restricting a specific type of the DU's personal data within a certain context. The following example is a user policy created based on the policy (P1.a) by using the same approach as that in creating (P1.a) from (P0.a).

Example (User policies). A domain policies instantiating policy (P1.a) for DU Alice in the domain is

(P1.b) : $\langle\langle d_1, p_{1.b} \rangle, \langle d_1, p_{1.a} \rangle, \langle \{d_1\}, \{alice\}, data, propagate \rangle,$
 $\langle d_0, d_1 \in DI, p_{0.a} \in PI(d_0), p_{1.a}, p_{1.b} \in PI(d_1), alice,$
 $bob \in U(d_1), propagate \in A,$
 $att(alice, data), consent(alice),$
 $recipient(data, bob) \rangle\rangle.$

Policy (P1.b) is specific to Alice by converting a general DU u_1 in (P1.a) into Alice. In addition, an axiom $recipient(\cdot)$ is added to restrict the recipient of Alice's data only to DU Bob.

4. Policy Provisioning Framework

This section describes detailed procedures on managing the lifecycle of policies as well as on data to be managed using these policies.

4.1 Architecture

The system architecture of each domain for the proposed provisioning framework is shown in **Fig. 4**. Both DC and DP systems have common functions such as *filter*, *access control*, *policy management*, and *user interaction*. The filter function monitors requests from a DC or a DU whose access needs to be controlled. The access control function determines whether requests are to be granted or denied, the policy management function consists of sub-functions, which will be described later, and the user interaction function interacts with DUs with a DC or DP to perform policy-related actions.

The *mapper*, *creation*, *update*, *revocation*, and *incorporation* are common to DC and DP in the policy management function. The mapper function changes a target contained in a policy representation into a domain specific one and converts the name iden-

tifiers of DUs. Since each domain manages DUs, policies, and objects independently of other domains as a basic unit of administration in the distributed system, their identifiers may differ from those of other domains. In such cases, the mapping function resolves the name differences between domains. The creation function creates a new policy. The update and revocation functions update and revoke an existing policy. The incorporation function integrates a policy with another that has been provided by other domains. The *publisher* function in a DC publicizes and sends policies to DPs in other domains in a *push* manner in which the DC initializes the propagation of its policies or in a *pull* manner in which it returns with them in response to DPs' requests. In contrast, the *subscriber* of a DP subscribes and receives a DC's policies in the opposite manner to a DC.

A DC manages *policies*, *data*, *published policy information*, and *agreed upon policies*. The published policy information a DB manages is a set of the identifier of a propagated policy and its recipient domains. This information is used to send notifications to the domains when the policies are updated or revoked. The agreed upon policies are user policies that DC and DP agree upon how to handle DUs' data. A DP also manages its policies whereas it may not have data depending on the constraints of the policies. Both a DC and a DP manage policies in each policy DB so that they are retrieved using a key such as a policy identifier and an action value.

4.2 Policy Operations

This section describes policy operations that include creation, update, revocation, publication, subscription, and incorporation using the functions outlined in Fig. 4.

4.2.1 Policy Creation

Policies are created by the administrator of the domain. Since a root domain corresponds to an administrative organization such as an industrial company or governmental body, its domain policies are created from the administrative viewpoints of security or privacy. The brief algorithm for creating a policy is listed in **Algorithm 1**.

Algorithm 1 createPolicy(domains, subjects, data, action, constraints)

Require: d : the identifier of the domain within which this function operates.

- 1: $p \leftarrow \text{new_policy}()$
- 2: $p.gpi = \text{new_gpi}(\text{generate_id}(), d)$
- 3: $p.par_gpi = \text{null}$
- 4: $p.target = \text{new_target}(\text{domains}, \text{subjects}, \text{data}, \text{action}, \text{constraints})$
- 5: $P.\text{putPolicy}(p)$
- 6: **return** p

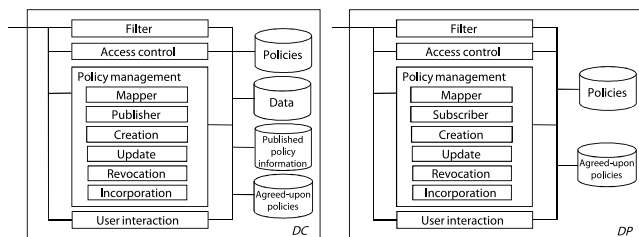


Fig. 4 System architecture.

A new policy is created in this procedure and its GPI and target are set if arguments such as domains, data, actions, and constraints are given. The parent global-policy identifier is set to **null** indicating that there is no parent information because this domain is the root of the policy.

Creation Request. If the administrator of a DC domain would like a DP domain to deploy the created policy, he or she propagates a message encapsulating it to the DP domain. The procedure for creating and sending policies is listed in **Algorithm 2**.

Algorithm 2 sendCreatedPolicies(*pols*, *d*)

Require: *pols*: the created policies; *d*: the domain to which *p* is propagated.
PP: published policy information DB.

```

1: pols' ← copy(pols)
2: for all i such that  $p_i \in \text{pols}$  do
3:    $p_i \leftarrow \text{map}(p_i, d)$ 
4:    $\text{gpi} = \text{new\_gpi}(d, p_i.\text{gpi}, p.\text{id})$ 
5:    $\text{PP.putDomains}(\text{gpi}, d)$ 
6: end for
7:  $\text{msg} \leftarrow \text{createMessage}(\text{"creation"}, \text{pols'})$ 
8: send(msg, d)

```

In step 1, the local identifier of a DU is converted into his or her federated identifier if it is used to identify him or her between the domain sending and receiving the policy. Function `new_gpi(·)` in step 4 creates a new GPI and this is then stored in the published policy information DB *PP* (step 5). A message containing policy *p'* with the message indication tag “creation” is produced and then the message is propagated to *d* (steps 7–8).

Operation for Received Creation Request. When a DC domain receives the creation request, the following procedure is performed.

Algorithm 3 receiveCreatedPolicies(*pols*, *d_s*)

Require: *pols*: the created policies received; *d*: this domain; *d_s*: the domain propagating the policies.

```

1: for all i such that  $p_i \in \text{pols}$  do
2:    $p \leftarrow \text{map}(p_i, d_s)$ 
3:    $p.\text{par\_gpi} = p_i.\text{gpi}$ 
4:    $p.\text{gpi} = \text{new\_gpi}(d, \text{generate\_id}())$ 
5:    $P.\text{putPolicy}(p)$ 
6: end for

```

When the DP domain receives the created policies, it recreates a policy by resolving the name identifiers of subjects for each policy (step 2), sets the parent GPI as domain *d_s*, sets the local GPI as the domain operating this algorithm (steps 4–5), and it is then registered in the policy DB (step 5).

4.2.2 Policy Update

The administrator of the domain updates an existing policy; the policy created as a result of the update is checked whether it is consistent with the super-domain policies on which the original policy is based, which is listed in **Algorithm 4**.

Algorithm 4 updatePolicy(*p*)

Require: *p*: the updated policy; *P*: the policy DB.

```

1: pols ←  $P.\text{getDomainPolicies}()$ 
2:  $\text{result} \leftarrow \text{checkSimilarity}(\text{pols}, p)$ 
3: if  $\text{result} = \text{true}$ , then
4:   sendUpdatePolicy(p)
5: end if

```

In step 1, existing policy *p* is updated. In step 2, the latest domain policies in the domain’s policy DB are retrieved by calling `P.getDomainPolicies()` and the similarities between the domain policies and the updated policy are checked using `checkSimilarity(·)`, which will be explained with **Algorithm 5** (step 3). The types of relationship between two policies are listed in **Fig. 5** [9].



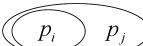


Policy similarity types	Authorized requests
p_i Converges p_j	
p_i Diverges p_j	
p_i Restricts p_j	
p_i Extends p_j	
p_i Shuffles p_j	

Fig. 5 Types of similarities in policies.

Algorithm 5 checkSimilarity(*pols*, *p*)

```

1: spols ←  $\text{pols.getPolicy}(p.\text{target.action})$ 
2: if spols = null, then
3:   pols.add(p)
4:   return true
5: end if
6: for all i such that  $p_i \in \text{spols}$  do
7:   if  $p_i.\text{diverges}(p)$  or  $p_i.\text{restricts}(p)$ , then
8:     return false
9:   else if  $p_i.\text{extends}(p)$ , then
10:     $\text{pols.del}(p_i)$ 
11:     $\text{pols.add}(p)$ 
12:   else if  $p_i.\text{shuffles}(p)$ , then
13:     $p' \leftarrow \text{createMeetPolicy}(p_i, p)$ 
14:     $\text{pols.del}(p_i)$ 
15:     $\text{pols.add}(p')$ 
16:   end if
17: end for
18: return true

```

There are five types of similarities in policies, “converges,” “diverges,” “restricts,” “extends,” and “shuffles.” Note that the similarities between two policies are viewed with respect to which of their conditions hold and that the area covered by a circle representing a policy corresponds to the scope with which it grants execution. Of these, the relationship between an existing domain policy and a new policy to be checked corresponds to “diverges” or “restricts” (step 7); a **false** value is returned (step 8), since the new policy has a different constraint from that of the existing one. If the relationship corresponds to “extends” (step 9), the existing policy is deleted and the new policy is registered (steps 10–11). Otherwise, if the two policies are in a “shuffles” relationship (step 12), a policy for the union of the existing and new policies is newly created by calling function `createMeetPolicy(·)` (step 13); the existing one is deleted (step 14), and the new intersection policy is added to the list of domain policies (step 15). Finally, this function returns **true** with a set of registered domain policies (step 18). Note that the above approach to integrating policies strengthens policy constraints except for “converges,” “diverges,” and “restricts” cases in which the existing policy encompasses the new one.

Update Request. If the administrator updates a policy that has been propagated to a domain, he or she propagates the updated policy to the domain to reflect the update to the policy that has been deployed. The procedure for updating and sending an updated policy to domains is listed in **Algorithm 6**.

Algorithm 6 sendUpdatedPolicies(*pols*)

Require: *pols*: the updated policies; *PP*: published policy information DB.

```

1: pols' ← copy(pols)
2: for all i such that  $p_i \in \text{pols}'$  do
3:   domains ← PP.getDomains( $p_i$ )
4:   for all j such that  $d_j \in \text{domains}$  do
5:      $p \leftarrow \text{map}(p_i, d_j)$ 
6:      $\text{msg} \leftarrow \text{createMessage}(\text{"update"}, \{p\})$ 
7:     send( $\text{msg}, d_j$ )
8:   end for
9: end for

```

In step 1, the information on domains to which policy p has been propagated is obtained by searching the published policy data. Then, for each domain stored in the domains, an updated policy is propagated after the corresponding message is created with its indication tag “update” (steps 2–6).

Operation for Received Update Request. When a DP domain receives the update request in Algorithm 6, it uses the following procedure in Algorithm 7.

Algorithm 7 receiveUpdatedPolicies(*pols*, *d_s*)

Require: *pols*: the updated policies, *d_s*: the domain propagating the policies.

```

1: for all i such that  $p_i \in \text{pols}$  do
2:    $p \leftarrow P.\text{getPolicy}(p_i.\text{gpi.p.id})$ 
3:    $p \leftarrow \text{map}(p, d_s)$ 
4:   updatePolicy( $p$ )
5: end for

```

The DP domain searches a policy corresponding to each updated policy it receives with its policy identifier in the policy DB (step 1). Then, it resolves the name identifier in the policy and recreates a new policy by calling function updatePolicy(p) in Algorithm 4. This means that policy-update operations are recursively performed beyond distinct domains if an update policy complies with its corresponding policy propagated by the DP’s super-domain.

4.2.3 Policy Revocation

When the administrator of the domain revokes an existing policy, the parent domain is checked to restore a parent domain policy if the policy to be revoked has been created by restricting the parent policy. This procedure is listed in Algorithm 8.

Algorithm 8 revokePolicy(p)

Require: p : the policy to be revoked.

```

1: if  $p.\text{par.gpi} = \text{null}$ , then
2:   sendRevocations( $\{p\}$ )
3:   P.delPolicy( $p$ )
4: else
5:    $p' \leftarrow p.\text{par.gpi.p.id}$ 
6:   P.delPolicy( $p$ )
7:   P.putPolicy( $p'$ )
8:   sendUpdatePolicies( $\{p'\}$ )
9: end if

```

If the policy to be revoked has been created locally (step 1), a revocation message is sent by calling sendRevocations(\cdot), which will be explained with Algorithm 9, and the policy is revoked (step 3). If the policy to be revoked is based on a parent policy (step 4), a corresponding update message is sent to domains to which the original policy has been propagated so that

they can deploy the parent policy instead of the policy that has been deployed.

Revocation Request. If the administrator revokes a policy that has been propagated to a domain, he or she sends a revocation message to the domain to successively revoke the policy that has been deployed. The procedure for revoking and sending a revocation message to corresponding domains is listed in Algorithm 9.

Algorithm 9 sendRevocations(*pols*)

Require: *pols*: the policies to be revoked; *PP*: published policy information DB.

```

1: for all i such that  $p_i \in \text{pols}$  do
2:   domains ← PP.getDomains( $p_i$ )
3:   for all j such that  $d_j \in \text{domains}$  do
4:      $\text{msg} \leftarrow \text{createMessage}(\text{"revocation"}, \{p_i\})$ 
5:     send( $\text{msg}, d_j$ )
6:     PP.delDomain( $p_i, d_j$ )
7:   end for
8: end for

```

In steps 3–6, a revocation message is created and sent to each domain d_j regarding policy p , which has been propagated. After that, each item of domain information and policy is delegated from the published policy DB (step 6).

Operation for Received Revocation Request. When a DC domain receives the revocation request, it uses the following procedure listed in Algorithm 10.

Algorithm 10 receiveRevocations(*pols*)

Require: *pols*: the policies to be revoked.

```

1: for all i such that  $p_i \in \text{pols}$  do
2:    $p \leftarrow P.\text{getPolicy}(p_i.\text{gpi.p.id})$ 
3:   revokePolicy( $p$ )
4: end for

```

In the same way as the procedure performed in Algorithm 7, policies to be revoked are sought by their policy identifier in the policy DB and revoked. In this procedure, the operations related to revocation are successively performed in the domains that have propagated and deployed the original policies.

4.2.4 Policy Subscription

The policy subscription function in a DP retrieves a DC’s policies. The following procedure for a DP creates and sends a policy subscription request to a DC, which is listed in Algorithm 11.

Algorithm 11 sendSubscriptionRequest(*d_c*, *pols*)

Require: *d_c*: a DC domain; *pols*: the policies of a DP sending this request.

```

1:  $\text{pols}^{(dp)} \leftarrow \emptyset$ 
2: for all i such that  $p_i \in \text{pols}$  do
3:    $p \leftarrow \text{map}(p_i, d_c)$ 
4:    $\text{pols}^{(dp)}.add(p)$ 
5: end for
6:  $\text{msg} \leftarrow \text{createMessage}(\text{"subscription"}, \text{pols}^{(dp)})$ 
7: send( $\text{msg}, d_c$ )

```

The policies in the arguments of the above function specify the DP’s policies restricted by the target encapsulated therein. The information on elements in the target affects what kinds of policies the DP would like to retrieve from a DC to which the request has been transmitted. If the policies, *pols*, are **null**, the DC hopes to obtain the DC’s latest policies related to the DP that

have been propagated to the DP. If *pols* is not **null**, on the other hand, there are some variations in the requested policies. For example, they are domain policies that do not constrain any specific subjects or user policies that specify how a DU's data are to be handled if these elements are contained in their target. The DC domain sends a subscription request with the indication tag “subscription” attaching the requested policy information after name identifier resolution (steps 1–6).

When the DP receives a response to the above subscription request from a DC or a publication message initiated by a DC, the following procedure is performed by the DP to subscribe to policies from the DC, which is listed in **Algorithm 12**.

Algorithm 12 subscribePolicies(*d_s*)

Require: *pols*: the published policies; *d_s*: super-domain.
1: **for all** *i* such that $p_i \in pols$ **do**
2: $p \leftarrow \text{map}(p_i, d_s)$
3: $pols.add(p)$
4: **end for**
5: **return** *pols*

The received policies are not added to the policy DB since the DP has already received them if they are domain policies. If they are user policies, they are stored in the agreed upon policy DB as the agreement between the DC and the DP.

4.2.5 Policy Publication

A DC domain publishes the latest policies managed in the policy DB to sub-domains at every fixed period in a *push* manner or when the domain receives policy subscription requests from sub-domains in a *pull* manner. The policies published to a sub-domain are limited to those whose target encompasses the sub-domain.

Publication Message. A DC's publication message corresponds to the propagation of the DC's latest policies that have been propagated to DP domains or in response to DP domains' policy subscription requests, which was explained in Section 4.2.4. Each publication message has the indication tag “publication.” The procedure for a DC to create and send publication messages is listed in **Algorithm 13**.

Algorithm 13 publishPolicies(*req*, *d*)

Require: *req*: a DP's subscription request; *d*: the identifier of the DP to which the published policies are sent.
1: **if** *req* \neq **null**, **then** {publishing in a pull manner}
2: $pols^{(dp)} \leftarrow req.getPolicies()$
3: **for all** *j* such that $p_j \in pols^{(dp)}$ **do**
4: $p' \leftarrow \text{map}(p_j, d)$
5: $pols \leftarrow P.getPolicies(p'.target)$
6: **if** $\neg(\exists p \in pols); p.converges(p')$ or $p.extends(p')$, **then**
7: **return false**
8: **end if**
9: **end for**
10: **else** {publishing in a push manner}
11: $pols^{(dp)} \leftarrow P.getPolicies(d)$
12: **end if**
13: $msg \leftarrow \text{createMessage}(\text{“publication”}, pols^{(dp)})$
14: $\text{send}(msg, d)$
15: **return true**

When the DC receives a DP's subscription request, *req* (step 1), it obtains the DP's policies that are requirements for handling the

data specified therein (step 2). Each policy from the DP is converted to a domain specific one (step 3), policies related to the DP policy are retrieved by searching the policy DB with a key to its target information (steps 4–5), and whether the DP policy conflicts with existing DC policies is checked (steps 6–8). Although this procedure is a policy negotiation between a DC and a DP, if there is conflict between their policies, the DC refuses to provide the DP with the data corresponding to the requested DP policies. If *req* is **null**, the DC obtains the latest policies related to a DP to initiate a message being sent that contains them (steps 10–11). Finally, the DC creates a message containing the policies with the indication tag “publication” and propagates them to the DP (steps 13–14).

4.2.6 Policy Incorporation

Policies in a domain need to be created based on its existing domain policies to avoid conflicts between them. The procedure in a domain for incorporating policies is listed in **Algorithm 14**. This procedure is used when a DC or DP creates a new domain policy complying with the existing domain policies or when a DP receives the agreed upon policies after requesting a subscription message from the DC.

Algorithm 14 incorporatePolicy(*p*)

1: $pols^{(d)} \leftarrow \text{getPolicies}()$
2: $result \leftarrow \text{checkSimilarity}(pols^{(d)}, p)$
3: **if** *result* = **true**, **then**
4: **return** $pols^{(d)}$
5: **else**
6: **return null**
7: **end if**

The DC retrieves its domain policies (step 1) and checks the similarities between the existing policies and the policy to be incorporated (step 2). If there are no conflicts, the policy is incorporated and the updated policy set is returned (steps 3–4). This procedure avoids policies from being incorporated that have conflicts.

5. Case Study

This section presents a case study using the proposed framework.

A domain's user policies are generally not sufficient to enforce them when the domain attempts to make a decision to grant a data access request because it is difficult for a domain's administrators to statically specify any constraints dependent on data types and DUs, or because it is impossible for a DU to specify his or her user preferences unless conditions are presented. If the policies include a DU's decision or consent, the domain especially needs to interact with the DU and obtain it at runtime. Since this DU's decision or consent indicates one of the DU's user preferences, they need to be incorporated into a domain's policies.

Figure 6 outlines a series of interactions between a DP, a DC, and a DU for the DP to retrieve the DU's personal attribute data managed by the DC. The procedure includes control by a DC to access managed personal data and dynamically incorporate the user preferences of the DU.

The DP sends a request for attaching an agreement to its DC

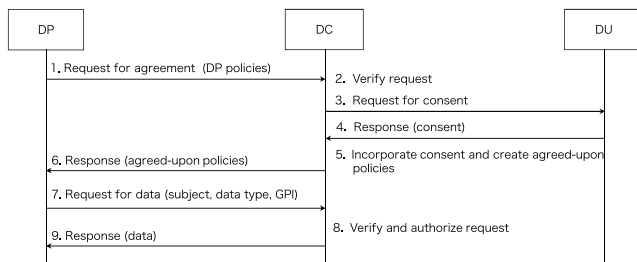


Fig. 6 DC's policy and procedure for data provisioning.

policies specifying how and for what purpose the DC intends to use a particular DU's personal attributes (step 1, see Section 4.2.4). The DC verifies the request message and checks whether the policies sent from the DP conflict with the DC's policies (step 2, see Section 4.2.5). The DC requests consent from the DU that owns the target data to consent by describing how the data will be used by the DP because the DC's policies have a constraint on the DU's consent to the action of data propagation (step 3). The DU shows the consent to the DC, which is one of the user preferences (step 4). Therefore, the consent is incorporated into the DC policies and agreed-upon policies are created using the updated DC policies and DP policies (step 5, see Section 4.2.6). The policies are in agreement including an identifier that can be used to identify the authenticity of the data request arriving from the DP. After the DC responds with these agreed-upon policies (see Section 4.2.5), the DP stores these policies as an agreement with the DC on data-handling (step 6, see Section 4.2.4). The DP then makes a new request for the DU's personal data with a GPI of the agreed-upon policies (step 7). The DC confirms whether it manages the agreed-upon policies identified by the GPI contained in the received request from the DP. The DC verifies the agreed-upon policies (step 8) and then propagates the requested data to the DP (step 9).

In the above example, a DC incorporates a DU's consent into its user policies at runtime when a decision on access control to propagate data is needed. In the same way, a DP also incorporates a DU's consent to process the DU's data at runtime when the DP attempts to do so.

6. Discussion

This section discusses several topics and issues related to the proposed model and framework.

6.1 Validation of Proposed Framework

The proposed framework integrates the data-handling policies of a DC and a DP when they exchange corresponding personal data. When a domain creates new domain policies, the proposed framework incorporates super-domain policies into the new ones. In these operations, a parent domain's policies are always reflected in the new policies while conflicts between policies are detected and avoided by checking for similarities. Hence, requirement (1) presented in Section 2 is satisfied. In addition, the proposed framework can keep track of data-handling policies wherever their corresponding data propagate in different entities in a distributed environment by assigning the GPIs to the data based on the proposed model and by conveying the information

along with the data as specified in the policy operations in Section 4. Therefore, requirement (2) presented in Section 2 is also satisfied.

6.2 Data Retention

Policies in the proposed model that reflect local laws or regulations of social organizations or computer systems are appropriately provided beyond administrative domains in distributed environments since the model has a hierarchical structure for creating and propagating policies and supports a policy propagation chain in accordance with data propagation. By means of this approach, the administrator of a root domain can retain the management of data enforcing their provisioned policies that reflect his or her intentions regarding data governance even after the data have been propagated in the distributed system. In addition, participating parties in the system can clarify their liabilities concerning data practices and improve their accountability for activities in handling data.

6.3 Policy Conflicts

The proposed model facilitates the avoidance of policy conflicts between entities detecting similarities between policies when new policies are created or integrated. Since DP entities need to accept a DC's controlling policies or strengthen their constraints, there is no room for policy negotiations in policies between them. This is appropriate from the governance and compliance viewpoints of a root domain's administrator. However, this approach may reduce the expressiveness of policies when flexible representations such as exceptional action rules are needed. Although this is a problem involving a trade-off, finding an appropriate balance between administrative restrictive descriptions and rich representations of policies is an open issue that bears further investigation.

6.4 Directive and Recommendation Policies

Governance policies state that personal information can be managed by a DU as its owner from a user-centric perspective. However, it is difficult for a DU to take appropriate action in all environments. For example, privacy laws and privacy guidelines such as those of the OECD [10] dictate that enterprises should take into account the consent given by people to use their data for specified purposes. However, people may possibly act inappropriately if they do not understand what their consent involves, which unfortunately does not match the intentions of legislators of privacy-related guidelines. In such cases, administrators can specify complementary directive policies stating that enterprises should provide sufficient explanations to people about how their data will be used or that enterprises should suggest possible actions for them to take.

6.5 Practicality

Although this work dealt with policy provisioning by introducing a general description of data-handling policies to describe the general framework and algorithms, further more specific representations of data-handling policies are needed to deploy them in real systems because the mapping and resolution of name iden-

tifiers depend on what policy representations are deployed. Although policy operations as well as their propagation are evidenced by each algorithm, an overall implementation and feasibility study of the proposed framework remain to be done in future work.

7. Related Work

This section highlights research efforts in the area of access control, privacy management, and policy-specification languages related to the work presented in this paper.

The idea of controlling access to data even after they have been disseminated has been considered especially by the digital rights management (DRM) community [11]. Their work has focused on protecting digital content from unauthorized copying and distribution by disseminating packages containing the content data and access control policies. Prior work [7], [8] toward managing privacy has introduced the concept of “sticky policies,” in which handling policies are directly associated with personal information. In their approaches, users retain control over their personal information even after it has been disclosed by enforcing privacy policies, which reflect their preferences about how it is to be used next at its recipient site. The work here inherits the basic idea of tight bundling of data and policy described above, regardless of whether the type of policy is security or privacy related.

There have been numerous research efforts related to extensions of traditional mechanisms for access control to protect privacy [1], [2], [12]. Ardagna et al. [2] proposed a privacy-aware system to control access that enforced access control policies together with privacy policies such as release and data-handling policies that regulated the use of personal information in secondary applications. They focused on the introduction of data-handling policy language and the integration of traditional access control and data-handling policies created from two actors, i.e., a service provider that managed personal information and a user who originally had the information. However, their work did not describe how data-handling policies were created and deployed in a system that consisted of entities that had distinct responsibilities or roles and that supported multiple chains to disseminate data among those entities. The work described here instead focuses on policy management in which a data managing provider collaboratively establishes data enforcing policies.

Other relevant work on privacy and identity management has been on identity governance, which is an emerging concept to provide fine-grained conditional disclosure of identity information and enforce corresponding data-handling policies. Liberty Alliance specifies fundamental privacy constraints on such governance as the use, display, retention, storage, and propagation of identity information [4]. Although this work allows access control that enhances privacy by defining new types of expressive privacy policies, it does not fully take into consideration the integration or composition of policies among distinct actors located in different administrative domains. The proposed framework addresses a method of transmitting and incorporating data-handling policies associated with shared identity information between actors. Previous work [13] has addressed the notion of provisioning policies in a distributed environment. However, it has not sup-

ported persistent management policies. In contrast, the present work provides a comprehensive framework for keeping data and their associated policies under control from the broad perspective of distributed systems.

Relevant work has been done in the field of policy management such as policy integration and conflict resolution. Mazzoleni et al. [9] proposed policy integration algorithms for XACML [14]. They believed that XACML had not been built to manage security for systems in which enterprises were dynamically constructed with the collaboration of multiple independent subjects. The approach proposed here is relevant in that entities located in different security domains collaboratively share data and their policies. Belokosztolszki and Moody [15] introduced meta-policies for distributed role-based access control (RBAC) [16]. Warner et al. [17] proposed a coalition-based access control (CBAC) model in which access control policies were dynamically agreed upon between entities in a coalition relationship. Although their work was relevant in that they considered a hierarchical structure for managing policy in distributed systems, they did not deal with how to integrate and deploy the policies of entities in a hierarchical structure and distributed environment. Bettini et al. [18] formalized a rule-based policy framework that controlled access by user requests for action by evaluating rules associated with provisions and obligations, which were pre-conditions to be satisfied before and post-conditions to be satisfied after the action had been performed. Their framework provided a mechanism for reasoning about the policy rules in the presence of provisions and obligations in a single administrative domain to derive an appropriate set of these. In contrast, the framework presented here is composed of a set of policies using different actors with different responsibilities in distinct domains and it enforces the composed policies that reflect a system administrator and a user in distributed identity management systems.

Another important contribution has focused on secure interoperation in a multi-domain environment where user access from different domains is allowed by employing their own security policies [5], [6]. Shafiq et al. [5] proposed a policy integration framework for merging the heterogeneous RBAC policies of distinct domains into a global access control policy by resolving conflicts that may have arisen among them. Although their research shared this work's goal of creating and integrating policies that satisfied requirements arising from distinct entities in different domains, their approach lacked support for the lifecycle management of access control policies in each domain in which personal information is propagated. In addition, their approach lacked the notion of protecting a user's privacy because it did not deal with administrative policies or user preferences in handling identity information. In contrast, the present work proposes an interoperation framework for providing data-handling policies to trusted systems to persistently control identity information in a way that preserves privacy. Promruen et al. [6] proposed an RBAC framework using generalized temporal role based access control (GTRBAC) [19], which generates an inter-domain policy with temporal constraints to allow secure access from an external partner. They presented algorithms in the framework for generating a minimal disjoint set of merged policies by determin-

ing the relations between them such as containment, equivalence, overlapping, and disjunction, which are relevant to the proposed schemes for updating, publishing, and incorporating policies in this work. The main limitation to their solution with respect to policy integration was that it was specific to roles because its operation involved restructuring a role hierarchy to establish an inter-domain access policy. In contrast, the proposed solution provides a scheme for integrating policies from distinct entities in a hierarchical relationship to control identity information regardless of access control models that an individual entity uses therein.

There has been a great deal of work on description languages and constraints on privacy policies. P3P [20] and its complement APPEL [21] provided the means for expressing comprehensive user preferences. XACML [14] specified an access control language to describe access control constraints and provides privacy extensions to support privacy related constraints. EPAL [22] provided a privacy policy language for governing data practices in enterprise systems. The Liberty Identity Governance Framework (IGF) [4] specified privacy constraints. Although the framework proposed here assumes such an expressive privacy policy and access control language as basic building blocks, it focuses on managing the lifecycles within which security and privacy policies are enforced irrespective of their schema or the format they are represented in.

8. Conclusion and Future Work

This paper described a policy provisioning model in which distinct entities in distributed environments collaboratively create and propagate data-handling policies. Algorithms for creating and integrating policies enable data-handling policies to be deployed and enforced to securely and privately control access to personal data in an enhanced manner. The proposed framework helps to manage the lifecycle of personal data and their data-handling policies to reflect the intentions of the system administrator and their owners. Future work includes the incorporation of policy-conflict resolutions and feasible studies on the proposed framework.

Reference

- [1] Byun, J.-W., Bertino, E. and Li, N.: Purpose Based Access Control of Complex Data for Privacy Protection, *Proc. 10th ACM Symposium on Access Control Models and Technologies (SACMAT'05)*, pp.102–110 (2005).
- [2] Ardagna, C.A., Cremonini, M., De Capitani di Vimercati, S. and Samarati, P.: A Privacy-Aware Access Control System, *Journal of Computer Security*, Vol.16, No.4, pp.369–397 (2008).
- [3] Ni, Q., Bertino, E., Lobo, C., Brodie, C., Karat, C.-M., Karat, J. and Trombetta, A.: Privacy-Aware Role-Based Access Control, *ACM Trans. on Information and System Security*, Vol.13, No.3, pp.24:1–24:31 (2010).
- [4] Liberty Alliance Project: Liberty IGF Privacy Constraints Specification (2008), available from <http://www.projectliberty.org/specs>.
- [5] Shafiq, B., Joshi, J., Bertino, E. and Ghafoor, A.: Secure Interoperation in a Multidomain Environment Employing RBAC Policies, *IEEE Trans. Knowledge and Data Engineering*, Vol.17, No.11, pp.1557–1577 (2005).
- [6] Piromruen, S. and Joshi, J.: An RBAC Framework for Time Constrained Secure Interoperation in Multi-domain Environments, *Proc. 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'05)*, pp.36–48 (2005).
- [7] Karjoth, G., Schunter, M. and Waidner, M.: Platform for Enterprise Privacy Practices: Privacy-Enabled Management of Customer Data, *Proc. 2nd International Conference on Privacy Enhancing Technologies (PET'02)*, pp.69–84 (2002).
- [8] Casassa Mont, M., Pearson, S. and Bramhall, P.: Towards Accountable Management of Identity Privacy: Sticky Policies and Enforceable Tracing Services, *Proc. 14th International Workshop on Database and Expert Systems Applications (DEXA'03)*, pp.377–382 (2003).
- [9] Mazzoleni, P., Crispo, B., Sivasubramanian, S. and Bertino, E.: XACML Policy Integration Algorithms, *ACM Trans. on Information and System Security*, Vol.11, No.1, pp.1–29 (2008).
- [10] OECD: OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data (2004), available from http://www.oecd.org/document/18/0,2340,en_2649_201185_1815186_1_1_1_1,00.html.
- [11] Schneck, P.: Persistent Access Control to Prevent Piracy of Digital Information, *Proc. IEEE*, Vol.87, No.7, pp.1239–1250 (1999).
- [12] Casassa Mont, M. and Thyne, R.: Privacy Policy Enforcement in Enterprises with Identity Management Solutions, *Journal of Computing Security*, Vol.16, No.2, pp.133–163 (2008).
- [13] Gomi, H.: Policy Provisioning for Distributed Identity Management Systems, *Proc. 2nd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management (IDMAN'10)*, pp.130–144 (2010).
- [14] OASIS: eXtensible Access Control Markup Language (XACML) (2005).
- [15] Belokosztolszki, A. and Moody, K.: Meta-Policies for Distributed Role-Based Access Control Systems, *Proc. 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, pp.3–18 (2002).
- [16] Sandhu, R., Coyne, E., Feinstein, H. and Youman, C.: Role-based Access Control Models, *IEEE Computer*, Vol.29, No.2, pp.38–47 (1996).
- [17] Warner, J., Atluri, V. and Mukkamala, R.: A Credential-Based Approach for Facilitating Automatic Resource Sharing Among Ad-Hoc Dynamic Coalitions, *Proc. 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec'05)*, pp.252–266 (2005).
- [18] Bettini, C., Jajodia, S., Sean Wang, X. and Wijesekera, D.: Provisions and Obligations in Policy Management and Security Applications, *Proc. 28th International Conference on Very Large Data Bases (VLDB'02)*, pp.502–513 (2002).
- [19] Joshi, J., Bertino, E., Latif, U. and Ghafoor, A.: Generalized Temporal Role Based Access Control Model, *IEEE Trans. on Knowledge and Data Engineering*, Vol.17, No.1, pp.4–23 (2005).
- [20] W3C: The Platform for Privacy Preferences 1.0 (P3P1.0) Specification (2002), available from <http://www.w3.org/TR/P3P/>.
- [21] W3C: A P3P Preference Exchange Language 1.0 (APPEL1.0) (2002), available from <http://www.w3.org/TR/P3P-preferences/>.
- [22] IBM: Enterprise Privacy Authorization Language (EPAL 1.2) (2003), available from <http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/>.



Hidehito Gomi received his B. Eng. degree from the Department of Applied Mathematics and Physics and the M. Eng. degree from the Division of Applied Systems Science, Faculty of Engineering, Kyoto University, Kyoto, Japan, in 1994 and 1996, respectively. From 1996 to 2007, he was a researcher in the laboratories of NEC Corporation. From 2001 to 2003, he was a visiting researcher in the Computer Science Department of Stanford University, CA, USA. In 2007, he joined Yahoo! JAPAN Research, where he is continuing his research on security architecture, identity management, and trust management. He is a member of the IEEE-CS, ACM, IEICE, and IPSJ.