Management of Conflicting Obligations in Self-Protecting Policy-Based Systems

Rema Ananthanarayanan
IBM India Research Lab
New Delhi, India
arema@in.ibm.com

Mukesh Mohania
IBM India Research Lab
New Delhi, India
mkmukesh@in.ibm.com

Ajay Gupta
IBM India Research Lab
New Delhi, India
ajaygupta@in.ibm.com

Abstract

Policy-based management of business systems is increasingly becoming the norm for autonomic computing since these systems can adapt to the changing needs and increasing complexity of the underlying organizations or enterprises. One of the vital characteristics of these systems is self-protection, i.e., the ability to secure information and resources, by anticipating, detecting, identifying and protecting against any form of unauthorized access and permitting all authorized accesses based on the users’ roles and pre-established policies. In this paper, we focus on one aspect of self-protecting autonomous systems, which is, how to automatically enforce privacy policies related to data handling, for compliance and auditing purposes. The automatic management of privacy sensitive information based on enterprise policies that are driven by a combination of user preferences, internal objectives and external regulations is a key aspect to any enterprise to prevent misuse of this information. These policies extend beyond simple authorization rules, and also mandate obligations to be enforced under certain conditions. One issue in the automatic enforcement of obligations is the presence of conflicts among different obligations which mandate different actions on the same resource, based on different conditions in which the resource is accessed. In this paper we propose algorithms for detecting and resolving conflicts among obligations in both static and runtime environments. We then briefly describe our prototype obligation management system with the conflict resolution module that achieves the automated enforcement of obligations for data-handling based on privacy policies.

1 Introduction and Motivation

Policies form the basis of any system of autonomic computing, where well-defined policies enable systems to manage themselves and respond to changes in the environment. Policies are one aspect of information which influences the behaviour of objects within the system [13]. Alternately, a policy statement may be viewed as comprising one or more rules that encode or specify system behaviour. Each rule specifies what action is allowed or denied, or what action is to be effected in different situations. In the context of autonomic computing, the rules may also be viewed as specifications that show how autonomous agents interact with each other.

The key characteristics of autonomic computing are self-healing, self-configuring, self-optimising and self-protecting [1]. Our focus in this work is on one facet of self-protecting systems, where systems safeguard the information held by them by providing the right information to the right users based on the users’ roles and pre-established policies. The self-protecting characteristic is very essential for businesses to effectively enforce security and privacy policies. In the increasingly complex legal landscape of privacy laws, customers’ rights to privacy protection vie with business interests to provide better targeted services to customers based on information that they have on customers. To ensure protection of privacy information and to safeguard how this information is used, companies define privacy policies that comprise various privacy rules; each of these rules specify how various pieces of information need to be handled. For scalable and effective implementation of these rules, it is essential to have in place a system for automatic enforcement of the various rules. However, privacy rules, in addition to specifying authorizations, also mandate obligations that need to be effected on certain kinds of data access. Examples of these obligations include tasks such as notifying the customer, when information pertaining to the customer is accessed for any purpose or retention-related tasks such as deleting records at a specific time or archiving documents after some use. Hence systems that enforce these privacy rules should also be able to enforce obligations associated with these rules.

One of the key differences between authorization rules in general and the rules in privacy policies is that the former are defined by the system administrators, based on rights and authorities of various users; rules in privacy policies are framed in response to customer preferences and external regulations, apart from internal administrative require-
ments. Customers may state their preferences regarding how much of their personal information may be accessed, by whom, and for what purpose. Customers may also request that they be notified when their personal information is accessed, or that the information be deleted after some number of accesses. This brings in the need for associating obligations with some of the privacy rules. These obligations are defined based on many aspects of the user request, such as the user role, the kind of resource being accessed, the intent and so on, and hence may sometimes specify conflicting tasks with respect to the same resource. This creates the need to handle conflicting obligations in privacy rules.

Figure 1 shows the role of an obligation management system in an enterprise with a policy-based management system. Business goals and objectives drive the framing of policies in different areas such as administration, safety, security and finance. These policies, framed in a human language, are rewritten in structured format (say, using XML) for the purposes of implementation and machine readability. Each of the policies may comprise one or more rules, and zero or more of the rules may take effect in different situations. For instance, a specific request for some data by a user may be authorized under some conditions, to be followed by the execution of zero or more obligations. An obligation management system would then perform the task of checking each specific user request for an applicable rule. If the rule mandates an obligation, then the obligation management system would also ensure that the obligation is executed and the necessary information logged or audit trails maintained as necessary. The box shown as request interceptor in figure 1 could be part of the obligation management system (as we have designed in our prototype) or could be an additional component that provides input to the obligation management system. In this paper, we describe a conflict management system that is part of an automatic obligation management system, to enforce obligations related to a privacy policy. Our main contributions in this work are to:

- Highlight the importance of automated obligation management associated with data-handling in policy-based systems, for autonomic computing
- Describe a conflict management system for managing conflicting obligations related to a privacy policy. The system comprises conflict detection and conflict resolution for the static and runtime cases. We describe algorithms for each of these cases.
- Describe our prototype automated obligation enforcement system and associated issues, and the conflict resolution system.

The rest of this paper is organised as follows. Section 2 presents the background to work on privacy policies and automation and also presents the definitions and notations that we use in the rest of the paper. In section 3, we discuss obligations and conflicts with respect to privacy policies. In section 4, we discuss conflict detection and conflict resolution among conflicting obligations in privacy policies. In section 5, we present an overview of our obligation management system for automatic enforcement of obligations and the conflict management scheme implemented. In section 6 we give our concluding remarks.
2 Background

2.1 Related work

The main motivation behind policy-based management systems has been to support dynamic adaptability of behavior by changing policy without recoding or stopping the system [6]. [11] specifies 2 types of policies of interest in a distributed system - authorization policies and obligation policies. Authorization policies are used to specify what services or resources a subject (a management agent, user or role) can access. Obligation policies specify what activities a subject must or must not do to a set of target objects, and define the duties of the policy subject. [12] introduces the notion of adding obligations to common authorization mechanisms, associating obligations with permissions. Obligations are introduced as a means of ensuring system integrity, since some actions always need to be followed by others to ensure system integrity. Here, a permission is viewed as an authorization to do some action, requiring also an obligation, where an obligation is written as <obligation> :: <required> by <deadline> or else <sanction>. Association of obligations with authorizations add a degree of determinism to the system, in terms of what will happen when something is authorized.

Policies provide the framework for the building of autonomic systems. Many languages have been defined for policy specification. PONDER [7] is a language used for specifying management and security policies. It is declarative and object-oriented and can be used to specify both security and management policies. The language defines domains for grouping objects, roles, and relationships to define interactions between roles and management structure.

P3P is a W3 standard used by websites to encode their data collection and data use practices in a machine readable XML format [2].

XACML [3] or extensible Access Control Language is an XML-based language for access control, standardised by the Organisation for the Advancement of Structured Information Standards (OASIS). XACML describes both the access control policy language and a request/response language to express queries about whether a particular access should be allowed, and describes answers to those queries.

In the domain of privacy policies, P3P and XACML have been widely used to specify the privacy policies of enterprises. As stated earlier, these policies go beyond simple ‘Allow’ or ‘Deny’ rules of traditional authorization rules, and also optionally specify conditions to be met prior to granting permission for a request, and obligations to be executed, when a request is granted. Policies are framed to cover a number of requirements, ranging from routine day-to-day operations and non-routine system breakdown or other emergency tasks; further, the definition attempts to cover all categories of users in the system, administrative, managerial and non-managerial. Hence it is not uncommon for policy conflicts to occur in the system because of the conflicting requirements that arise in different scenarios. To cite a common example, one system policy may state that the system may not be shutdown when users are logged in and working, while another policy may state that in the case of certain emergencies, the system administrator may shut down the system without adequate advance warning. Conflict resolution in general business policies is well-studied [11][8] [9]. [5] formalizes a notion of obligation and investigates mechanisms for monitoring obligations and compensating for unfulfilled obligations. [4] defines a rule-based policy framework that includes provisions and obligations. Provisions are what need to be met before the task is authorized and obligations are what need to be performed after the task is authorized. These works describe the need for obligations in general, and how the system may compensate for unfulfilled obligations. In our current work, we describe one of the issues in the automated enforcement of obligations, viz., the detection and resolution of conflicts amongst conflicting obligations. In the rest of this paper, we develop the notion of obligations related to privacy policies, a representation for such obligations, and present algorithms for conflict management.

2.2 Definitions

In this section we list some of the definitions and notations that we will use to define the proposed conflict management. We have been guided by a policy definition similar to XACML standard. Each privacy rule defines whether a data access is allowed or denied, based on the kind of data being accessed, the user role, the intent and intended action. In addition, each rule may also mandate an obligation on the system.

Definition 1 A privacy policy comprises a set of privacy rules.

We denote a privacy policy by Π. A rule in a privacy policy specifies whether a specific request is to be allowed or denied, in the context of some conditions in the environment evaluating to true; the rule may optionally also mandate one or more obligations. Formally, we define a rule as under:

Definition 2 A rule is a four tuple, comprising an event, a condition, a permission and optionally, one or more obligations.

Each of these components is now defined below. An event relates to those components of a user request that are used to determine whether the request is allowed or denied. We define an event as follows:
Definition 3 An event is a 4-tuple, comprising the user category, the data category, the action and the purpose associated with a user request.

An event $E$ is represented as $E = \langle u, d, a, p \rangle$ where $u$ is the user category to which the user (who is requesting the data) belongs, $d$ is a data category to which the data item belongs, $a$ is the action that the user wishes to perform on the resource, and $p$ is the purpose for which the resource is being accessed. The following are some of the other terms that we will use in the rest of the paper.

User category Each user of the system is assigned to one or more user categories, based on the user roles. The user categories could be hierarchical.

Data category Each data item that is to be privacy-protected is assigned to a specific data category. The data categories could be hierarchical.

Action Users may access the data items to perform one or more of the specified set of actions. A representative set of actions could be {'View', 'Modify', 'Delete'}.

Purpose Users may access the data item for a specific purpose, from one of the set of purposes specified. A representative set of purposes could be {'Marketing', 'Research', 'LegalPurposes', 'Administrative'}.

Data item A data item is an entity that is to be privacy protected, i.e., whose access is controlled by one or more rules in the privacy policy. Each data item belongs to some data category. Examples of data items include emails, facsimile documents, customer records and scanned images (of x-rays, for instance, when the domain is the health industry).

The privacy rules come into effect when a user makes a request on the system.

Definition 4 A user request is a 4-tuple comprising the user id, the data item being requested, the intended action and purpose.

Note that each user request maps internally to an event, where the user is mapped to the user category and the data item is mapped the data category. Some more terms that we use are:

Condition This is a clause associated with some rules, which needs to be true, for the rule to ‘Allow’ access. Examples of conditions include If the user is above 18 years of age, If the user has consented, ...

Permission This is a binary predicate that states whether the user request is allowed or denied. It can take one of the two values, ‘Allow’ and ‘Deny’.

Figure 2. Object model of the rules in a policy

Obligation An obligation is a task optionally associated with some rules, that specifies what action needs to be mandatorily performed after the user request is fulfilled.

Let $O$ represent the set of obligations in the system. We subsequently discuss obligations in more detail. Figure 2 is an object model representation of the key objects of interest, and their relationships, in the context of a privacy policy as we have defined above. Note that in theory a large number of events are possible, but in practice the system is interested only in events which are to be specifically permitted or specifically denied. All other events are not usually catalogued in the policy, but assumed to take on a default permission of ‘Allow’ or ‘Deny’. Hence in practice the number of events that figure in the policy are much smaller than the total number of events based on the possible combinations of user groups, data items and other entities that define an event. Further, as we can see, a user request is only a specific instance of an event. In the subsequent section, we expand on the definition of obligations given here, and also define conflicts among obligations.

3 Obligations and Conflicts

3.1 Obligations

As stated in [12], obligations are introduced to preserve the integrity of the system and to ensure that all the commitments with respect to various business policies are fulfilled. Access and use of privacy information in enterprises is governed by various rules, both externally mandated by various government agencies, and internally framed in the interests of establishing credibility and customer trust. Customers may have indicated that they wish to be intimated whenever personal information is accessed for marketing purposes.
Alternately, as part of the privacy policy, enterprises may promise that some information would be deleted immediately after use. Obligations are defined to fulfill these kinds of requirements. The nature of the tasks also enables the classification of obligations in the privacy or data-handling domain into the following types:

- **Notification related**: These obligations require the system to inform or notify specific people when specific data items are accessed. When multiple notification-related obligations are defined on a resource, these can all be fulfilled, and hence no conflicting obligations are likely to arise.

- **Retention-related**: These kinds of obligations basically specify actions related to life-cycle of the data item accessed. More specifically, viewing the data item as going through the stages of creation, use, retention and disposition, retention-related obligations have bearing on the retention schedule and the disposition schedule of the items. Since different obligations may specify different requirements on the retention of the data item, conflicts are likely to arise in this case. Figure 3 shows a simplified view of the life cycle of a data item or information resource, in the system. This figure defines four states, in addition to a ‘Start’ state and an ‘End’ state. Retention-related obligations in the privacy and data-handling domain generally involve a transition from one state to another in the life cycle definition. Transitions from one state to another may be effected based on user tasks such as reading or updating documents and temporal events such as after a certain time period, or after $m$ accesses by users. The above figure is only one possible state diagram. The actual state diagram would be defined by the record-keeping and data-handling rules of the enterprise. Further, some of the states in the figure, such as the ‘Active’ state, could themselves be viewed as comprising additional states. These are described in more detail subsequently.

We now define an obligation more rigourously, in the context of privacy protection of data.

An obligation comprises a task, a predicate and a set of attributes. The task component defines the action to be taken, in essence identifying the obligation as retention-related or notification-related type. The predicate associates the task with an object or data item. The set of attributes are optionally used to qualify the task. For instance, some obligations could be associated with a temporal component, such as stating before when the task has to be carried out, or after how long. In these cases, the set of attributes are used to capture these requirements. Using the object notation, the individual components of task, predicate and attributes of an obligation $o$ may be referred to by $o\text{.task}$, $o\text{.predicate}$ and $o\text{.Attributes}$ respectively, where $Attributes$ represents the set of attribute for the obligation $o$. We shall use this notation subsequently for defining conflicts between obligations. The policy may specify the rules and associated obligations in any structured format, one common means being to use a mark-up language such as XML. Few examples of obligations are given below, followed by the XML representation of the first and second obligation. As we can see, the obligations as defined in this context are different from the obligation rules specified in [6]. Further, unlike [5] we are not looking at how to penalise or compensate for unfulfilled obligations, but to ensure that the obligations mandated are fulfilled. In the next section we look at conflicts arising among various obligations.

**Examples of some privacy-related obligations**

1. *Notify individual when personal identifiable information is accessed by sales team for marketing purposes.*

2. *Delete the medical records from the repository, when the administrator has transferred the information to an alternate hospital.*

3. *Archive purchase information after it has been accessed 3 times for marketing or research.*

**3.2 Conflicts**

A business policy may define many rules that arise out of diverse business requirements in the enterprise, leading occasionally to rules that conflict with each other. In the most common case of authorization rules, these conflicts may arise [11] because

- Different sets of conditions may result in an allow and deny to same role, for the same request

- Conflicting rule definitions may result in a specific access request always resolving to ‘allow’ or always resolving to ‘deny’.

---

**Figure 3. Simplified view of a data item lifecycle**

![Figure 3](image-url)
4 Conflict detection and resolution

4.1 Conflict detection

Conflict detection is the process of identifying that a conflict exists in the system. Conflict detection may be done at different stages:

- Detection of static conflicts: This may be done at policy definition time, or after the policy is published, but before deployment. This involves detection of situations which could potentially cause conflicts, if allowed to occur. By definition, static conflicts may or may not manifest in the runtime system.

- Detection of runtime conflicts: This involves identifying a conflict when it arises at runtime.

4.1.1 Detection of static conflicts

Let \( O \) be the set of obligations in the system. Let \( D \) be the set of data items that are to be privacy-protected.

A mapping \( M(d) \rightarrow \{ O \} \) can be defined that maps each data item \( d \) to the set of obligations that apply on that \( d \). A sufficient condition for a conflict-free system is for \( M \) to be a one-to-one mapping; in this case, each data item maps to at most one obligation, and no conflict is possible. If \( M \) is a many-to-one mapping, where one data item maps to more than one obligation, then it is possible that the multiple obligations that apply to the same data item may conflict with each other, based on the tasks that are executed on this item.

The qualifying attributes associated with obligations specify temporal components, such as when the task is to be performed, before which time or after how long and so on. When no qualifying attribute is present, then the default interpretation is that the task is to be executed immediately. For example, a request for data that has the associated obligation to delete the relevant documents would imply that once the request is granted, the documents are to be deleted immediately. However, an obligation that states ‘Delete the document after 6 months’ would imply that the data request is satisfied immediately, but the obligation component is executed after 6 months. When the obligation is executed immediately, the change in life-cycle state of the resource, if any, is also effected immediately. In this case, no conflict is possible, since the entire set of tasks, for instance, satisfying the user request and fulfilling the obligation/s can be viewed as one atomic operation. Multiple obligations would not apply on the same object. However, when the obligations have a temporal component that specify when the obligations are to be effected, or for how long, or how many times, then multiple obligations get defined on the same instance of the data item, giving rise to conflicting requirements. In static
conflict detection, our aim is to identify, for each data item, the set of obligations that could apply jointly at any point in time.

Our algorithm for static conflict detection may now be described as follows. Given a policy (the set of rules) and a set of resources or data items, the algorithm first identifies, for each data category, the set of obligations that apply to that category, based on the rules in the policy. Subsequently, for each data category and associated set of obligations, the algorithm identifies whether any of the obligations potentially conflict with each other; if yes, then these obligations are identified. Two or more obligations conflict with each other if one or more of them has attributes that define temporal components regarding when the task is to be carried out. If none of the obligations associated with a data category have a temporal component, then no conflict is possible. The algorithm is formally defined as follows. (For a rule $r$ in the policy, we use the notation $r.$obligation to refer to the set of obligations mandated by $r$.)

**Algorithm 1 DetectStaticConflicts()**

1. Let $n$ be the number of data categories in the system that are to be privacy-protected. Let $O_i$ represent, for each $i$, $1 \leq i \leq n$, the set of obligations that apply on that category. At the start, $O_i = \emptyset$, $\forall i$.

2. For each rule $r$ in $\Pi$, and for each data category $i$ to which $r.$applies, if $r.$obligation $\neq \emptyset$, then $O_i = O_i \cup r.$obligation. Note that $r.$obligation could represent more than one obligation. At the end of this step, for each data category $i$, $1 \leq i \leq n$, we have the set $\{O_i\}$. I.e., we have the mapping $M$, where $M(d_i) \rightarrow \{O_i\}$, $\forall i$.

3. For each $O_i$, $1 \leq i \leq n$, let $O_{C_i}$ be the set of obligations that conflict with each other. Compute $O_{C_i}$ as follows.

   (a) Initialise: $O_{C_i} = \emptyset$.

   (b) By definition, each set $O_i$, $1 \leq i \leq n$, may be written as $\{O_i\} = \{O_{R_i}\} \cup \{O_{N_i}\}$ where $\{O_{N_i}\}$ and $\{O_{R_i}\}$ represent the set of notification-related obligations and retention-related obligations that apply to $d_i$. Again, by definition, $O_{N_i}$ does not contain obligations that conflict with each other.

   (c) If $|O_{R_i}| \leq 1$, then $O_{C_i} = \emptyset$. Else for each pair of obligations $o_i$ and $o_j$ in $O_{R_i}$, where $o_i \neq o_j$, if $o_i.$attributes $= o_j.$attributes $= \emptyset$, then there is no conflict between $o_i$ and $o_j$. Else, $O_{C_i} = O_{C_i} \cup o_i \cup o_j$.

4. We have now defined a mapping $M$, $M(d) \rightarrow \{O_C\}$, where each data item $d$ is mapped to the set of obligations that could potentially cause conflicts.

**end Algorithm**

At the end of the algorithm, for each data item, we have a list of obligations that could potentially create conflicts when mandated.

### 4.1.2 Runtime detection of conflicts

In the runtime detection, we track user requests as they enter the system, and check to see if they necessitate any obligations. If yes, then we look for conflicts as they arise. We now define a live obligation as follows.

**Definition 8** For a data item $d$, a live obligation is an obligation which has been mandated on the data item, but which has not yet been executed.

For runtime detection, we associate each data item with its set of live obligations. When a new request for data is submitted, if this is allowed, and if this mandates an obligation in the item, then it is added to the set of live obligations for that item. Whenever a new obligation is added to the set of live obligations, then a set of steps similar to step (3) in our algorithm DetectStaticConflicts is performed on the set of live obligations for that data resource, to determine if the new obligation conflicts with any of the existing ones in the list. Since we are talking of the runtime case, the values of the various attributes associated with the obligations are known, and it is straightforward to determine if any of these conflict. The algorithm operates as follows. At startup time, no obligations apply and every resource is associated with an empty set of live obligations. As user requests enter the system, based on the rules applicable, different obligations may be mandated on each data item. These obligations are added to the list of live obligations for that resource. As and when these obligations are executed, they are removed from the live list. Whenever a new obligation is added, conflict detection is performed for the current set of live obligations. The full set of steps for runtime detection is listed under:

**Algorithm 2 DetectConflictsInRuntimeSystem()**

1. For each resource $d$, let $L_d$ be the set of live obligations associated with that resource. At startup time, $L_d = \emptyset$.

2. Whenever a new request for data comes in, if that request is allowed, and if it mandates an obligation $o$ on resource $d$, $L_d = L_d \cup o$.

3. Perform conflict detection on the set of obligations in $L_d$, based on step (3) of the algorithm DetectConflictsInStaticSystem.

4. Any set of obligations $O_{C_d}$ that conflict with each other are identified as runtime conflicts, for $d$.

5. When an obligation is executed, remove that obligation from the live list associated with the related data items.
4.2 Resolution of conflicts

In the above sections, we have described steps for the detection of conflicts in the system, both static and runtime. Here we discuss schemes for resolving conflicting obligations. Conflict resolution is the process of determining which one of the conflicting obligations to enforce on a data item, when more than one obligation is mandated, all of which cannot be executed together. Conflict resolution can again be done at two stages.

1. Resolution of static conflicts: Multiple obligations that could apply on a data item are resolved, so that only one obligation applies and a conflict does not occur.
2. Runtime resolution: When a conflict occurs at runtime, execute steps to resolve the conflict.

The advantage of static resolution is that it involves no performance overheads. The rules are uniformly applicable, independent of the order in which the user requests occur. The disadvantages include overheads in detecting and resolving conflicts that may never occur. Runtime resolution applies only to obligations that actually arise in the system, which may be much less in number than the number of potential conflicts. The disadvantage, in addition to performance hit, is that it is not possible to predict in advance how different data items may get effected, since the actual resolution steps sometimes depend on the order in which the user requests arise.

4.2.1 Resolution of static conflicts

Static resolution of conflicts may be done at two levels.

1. Policy level This involves identifying the conflicting obligations and redefining or reframing those rules that result in conflicting obligations. Since this implies changes at the policy level, this may be used as a means of providing feedback to the policy maker.
2. Obligation management level This involves selecting or determining which one of the conflicting obligations to apply to a data item, when more than one obligation is defined on the item, and they all cannot be executed together.

In the case of conflicting business rules, some of the more common methods for resolving conflicts include giving higher priority to the more specific version of the rule than the more general rule, giving higher priority to users who are higher in the authority chain and so on [9],[11]. Multiple rules may fire when certain events occur; [10] handles this through a declarative specification of meta-rules. Metarules are defined to express relationships between various rules, which allow the user to reason statically about the rule behaviour. Figure 4 represents the goal of static conflict resolution. Conflicts arise because more than one obligation gets defined on a data item, and it is not possible to execute all the obligations. Hence conflict resolution at static time aims to determine which one to apply, among the many obligations that currently apply. As shown in the picture, the aim is to convert the many-to-one map from obligations to data items, to a one-to-one map, where each obligation applies to at most one data item. One way to select the most relevant obligation is to assign ranks to the various obligations, and select the one with the highest or lowest rank. The ranking may be done using various rules and heuristics. Since obligations in privacy data handling are related to user requests, we describe schemes for ranking obligations based on (a) the task associated with the obligations and (b) the user requests that give rise to these obligations. The ranking may be done by the system administrator or the entity responsible for enforcing the privacy policies. The following are some of the rules that may be used for ranking obligations. We start with the following definitions.

Definition 9 The rank of an obligation is a value assigned to an obligation that determines the priority of the obligation with respect to another obligation.

Definition 10 With respect to the states in the life cycle of a data resource, a state $s_i$ is said to strictly precede a state $s_j$ if $s_i$ always occurs before $s_j$ in the resource lifetime. A state $s_j$ is said to strictly follow a state $s_i$ if $s_j$ always occurs after $s_i$ in the resource lifecycle.

Based on the above, some schemes for assigning ranks to obligations are given below.

1. Ranking obligations based on the tasks associated with the obligations

- Given 2 tasks (related to the obligations) defining changes in state from $s_i$ to $s_j$ and $s_i$ to $s_k$ respectively, the task effecting a state change from $s_i$ to $s_k$ may be given a higher rank, if $k$ always follows $j$ in the lifecycle.
- Disposition-related or deletion-related tasks may be assigned higher ranks than non-disposition related tasks.
For 2 obligations with the same task, the task to be effected in the shorter time-frame may be assigned a higher rank than the task to be effected in the longer time frame.

2. **Ranking of user requests/events** Here, the user requests that fire the rules that mandate obligations are ranked. This ranking may again be achieved based on a number of parameters, some of which are the various components that comprise a user request. Different user categories may be assigned different priorities; if the categorization is hierarchical, categories at a lower level may be given more priority or a higher rank. Similarly, ranks may be assigned to the different purposes and actions associated with the user request. Further, weights may be assigned to the different components such as the user category, the action and purpose, and the final rank may be decided on the basis of the ranks assigned to each of the components combined with the weights for these components.

3. **Ranking of user requests and obligations** This comprises a combination of the steps above, where all the obligations associated with each resource are ranked based on a combination of weights to the obligation itself, and the requests mandating the obligation.

Based on the above rules for assigning ranks to the various obligations, we now define our algorithm for conflict resolution in the static case.

**Algorithm 3 ConflictResolutionInStaticSystem()**

1. For each data category \( d \), determine the set of obligations that apply to that category, \( \{ O_d \} \).

2. Determine if any of these obligations cause conflicts, by applying algorithm DetectStaticConflicts. If the set \( O_{C_4} \) output by algorithm DetectStaticConflicts is empty, then stop here. Else, proceed with step (3).

3. Rank the obligations in \( \{ O_{C_4} \} \) using one or more of the criteria defined in the above paragraphs.

4. Of the multiple obligations that apply on the data category, select the one with the highest rank.

end Algorithm

4.2.2 Resolution of conflicts in the runtime system

In the case of runtime resolution, only the obligations that actually arise have to be checked for conflicts. The system does not have to plan for all potential obligations. When none of the access rules or privacy rules are violated, for every additional request, a different view of the resource may be created and used to serve that request. In this sense, resolution of conflicts at runtime provides the additional flexibility that the many-one mapping of obligations to data items need not be converted to a one-to-one mapping. It may be sufficient to create a view of a one-to-one mapping at runtime, provided no rules are violated.

**Definition 11** A view of a resource for a specific user is a snapshot of the resource, associated with a set of obligations that apply to that user’s request.

For instance, when user \( A \) accesses a resource for some purpose, \( o_1 \) is defined that requires that the resource is to be archived after 3 months. When user \( B \) accesses the resource for some other purpose, \( o_2 \) applies, which requires that the resource be deleted after 6 months. Now it is possible to
define a view for user A, that defines the resource as ‘to be archived’ after 3 months. Further, user B could be oblivious of this view.

**Definition 12** An obligation \( o_1 \) is said to override another obligation \( o_2 \) if \( o_1 \) is of higher rank than \( o_2 \), and both cannot be executed together.

For instance, if \( o_1 \) specifies that a data item is to be deleted immediately on access, and if \( o_2 \) specifies that the data item is to be deleted after six accesses, then we may decide, based on the context that \( o_1 \) overrides \( o_2 \). In the resolution of conflicts in runtime systems, user requests are tracked as they arise and obligations, if any, are associated with the respective data items. When a new obligation is defined on a data item which already has one or more live obligations associated with it, one of the following applies:

- The new obligation may override the current set of live obligations, in which case it replaces the current set of obligations associated with that resource.

- The new obligation may not override the existing set of live obligations, but could exist along with them; in this case, a different view of the resource is created for each of the live obligations.

The algorithm may now be defined as follows.

**Algorithm 4 ResolveConflictsAtRuntime()**

For each resource \( d \) in the system:

1. Maintain the set of live obligations \( L_d \) and the set of views, \( V_d \). At start time, \( L_d = \emptyset \) and \( V_d = \emptyset \).

2. When a data request on \( d \) is allowed, if this also has an associated obligation \( o \), then \( L_d = L_d \cup o \).

3. If any of the existing obligations override the new obligation, then remove the new obligation from \( L_d \).

4. If the new obligation overrides any of the existing obligations, then remove the obligations that are overridden from \( L_d \).

5. If neither (3) nor (4) above is true, then create a new view \( v \) for the relevant data item, with this obligation \( o \). Update the set of views associated with this data item as \( V_d = V_d \cup v \).

**end Algorithm**

**Note:** When performing conflict detection and resolution in the static case, the algorithms have been defined with respect to the data categories. This is because the privacy rules are defined with respect to the data categories, and we are looking at potential conflicts. However, when performing conflict detection and resolution in the runtime case, the algorithms are defined on the individual data items, since we are now considering specific instances of obligations that apply on specific data items.

5 **Prototype of an obligation enforcement system**

5.1 **Architecture of the system**

We have implemented a prototype system for automated enforcement of obligations related to a privacy policy. The prototype was built to check the feasibility of an automated obligation management system for a defined privacy policy, and to understand the practical issues in designing and building the same. This system has been built using IBM DB2 Content Manager (CM) as the data repository, and using IBM Records Manager (RM) to enforce the obligations. CM functions as a repository of data in multiple media formats, and allows users to query for the data using various APIs. RM is a records management engine that can be integrated with different document or content management tools or applications, for providing the necessary records-keeping function in an organization. It allows for hierarchical classification of resources in different groups (or fileplan components, in RM terminology) and allows administrators to specify what actions should be taken for all resources of a specific fileplan component, in terms of life-cycle transitions.

Figure 5 shows the main components of our solution - a policy parser, a request interceptor an event handler and an obligation enforcement system. We assume the policy is specified in an XACML format or slight variations thereof.

The policy parser parses the input policy file and extracts the various user categories, data categories, actions, purposes, rules and obligations. For the purposes of the prototype, we have assumed that all conditional clauses associated with the rules evaluate to true. For each rule, the policy translator also constructs the event defining that rule, (as the 4-tuple, <user category, data category, action, purpose>) and the specific authorization. This is communicated to the event handler. The file plan component is designed to map the different data categories to different file plan compo-
ments. Multiple obligations may map to the same resource, and hence conflict resolution was required to resolve these. The scheme used for conflict resolution is discussed subsequently. The privacy engine maps different users to different user categories, and different data items to different data categories. These are the tasks performed at setup time, or before the system is deployed.

CM allows registered users to log in and query the system for resources. Since privacy rules dictate additional actions and intents to be satisfied before a request is allowed, the request interceptor provides an additional interface where users also specify why they require the information and what action they intend to perform. The request interceptor checks for authorization with CM, and if allowed, executes the query on CM and gets the result set. For each data item in the result set, it constructs a 4-tuple of <user category, data category, action, purpose> which is communicated to the event handler.

The event handler checks if this event is allowed or denied, based on the setup time information that it has received from the policy parser. If allowed, it also checks if there is an obligation mandated by the relevant rule. If yes, then the event handler informs RM (which is the obligation enforcement component in our prototype) of the obligation, by communicating <data category, data item identification> to RM. RM marks the data item in the corresponding file plan component, based on the data category to which it belongs. Subsequently, the event handler informs the request interceptor on the ‘Allow’ or ‘Deny’ ruling, as applicable. The request interceptor then displays the permitted results to the user.

The request interceptor and the event handler have also been designed to log the various actions, at the different stages, including the user queries, registering of obligations (when they are mandated) and execution of the obligations. The above architecture has been extended to also handle various kinds of obligations and to support different obligation enforcement systems other than RM.

Some of the issues arising from the above implementation are:

1. Handling conflicts among obligations (which is discussed at length in this work;)
2. Monitoring obligation execution over extended periods, since some obligations could be defined to take effect much later in time than the time of data access.
3. A means to verify the action and purpose specified by the user; in our prototype we assume that what is stated by the user is correct.

We now discuss a specific case of how we have handled item (1) above through the conflict management scheme that we have implemented in the prototype.

5.2 Prototype of the conflict management system

The various obligations define retention-related tasks in the various data items being accessed. These include tasks where the data accessed is to be deleted immediately or after a certain time period, or where the document is to be retained in current state for a certain time period or archived after a certain time period. In the prototype, we have implemented a scheme for the detection and resolution of conflicts amongst obligations in the static system. We now describe the algorithm that we have implemented.

The enforcement of obligations is achieved through Records Manager (RM). Each document or data item in RM is associated with a file plan component (defined earlier.) Each data category in the data repository is associated with a file plan component. The conflict resolution is performed as under:

1. For each data category, initially identify the set of obligations that could apply on that data category. Identify the retention-related obligations from this set.
2. Group together all obligations requiring the same task; rank the tasks in order of importance, from highest to lowest.
3. Within each group, sort the obligations in order of increasing time component of the temporal attribute, if any.
4. Within each group, if all obligations have a temporal component, then rank them in terms of shorter time to execution, to longer time to execution. If there is an obligation without a temporal component, it is assigned the highest rank (shortest time to execution.)
5. If all the obligations associated with a data category belong to one group only, then the obligation with the largest rank in this group is selected for execution, and this data category is associated with this obligation.
6. If the obligations associated with a data category belong to different groups, then, determine a combined rank for each, based on the rank within the group and the rank of the group.
7. Select the obligation with the largest rank, and associate that with the data category.

Our logic in adopting the least time to execute obligation is that, when two users state preferences as ‘archive after 6 months’ and ‘archive after 12 months’ respectively, then selecting the initial one to implement does not affect the system integrity. However, depending on the application domain, for instance, when the data records are being retained for audit purposes or for compliance with other legal regulations, then it might be more relevant to adopt the
‘longer-time-to-execute’ obligation. In this case, it might also be more relevant to perform the ranking itself using the reverse criteria from the one specified earlier. Our algorithms specify rules for ranking obligations and resolving conflicts; the application semantics and the context would decide the actual scheme in which the ranking is done.

6 Conclusions

In this paper, we have highlighted the need for conflict resolution amidst conflicting objectives related to data-handling policies, for autonomous systems. Rules related to privacy policies may have obligations associated with them and an autonomic policy-based management system should be able to automate the implementation of privacy rules and related obligations also. One of the issues in automated obligation enforcement is the handling of conflicts that may be present in the privacy rules. We have defined algorithms for detection of conflicting obligations, both statically and at runtime. We have then introduced the notion of ranking of obligations, and suggested several schemes for ranking the obligations associated with privacy rules, based on different criteria such as the tasks associated with the obligations and the user requests. We have then discussed algorithms for the resolution of conflicts in the static and runtime cases, based on this notion of ranking the obligations. We have also described a prototype automated obligation enforcement system and the conflict resolution that we have implemented in the prototype system.

Obligations associated with privacy policies have been studied earlier, in the context of penalising for unfulfilled obligations or compensating for obligations that are not fulfilled. Conflict resolution has also been studied in the case of business policies for authorising various activities. We believe our work is one of the first to discuss the role of obligation management in autonomous systems and to present algorithms for conflict management of obligations related to privacy protection of data items, for self-protecting systems.

One of our observations is that, when there are conflicting obligations, we are tacitly modifying the effect of some of the rules through our conflict resolution mechanism. This is driven purely by technical considerations for building an automated system for obligation enforcement and conflict management. However, the policy maker may have other considerations that may override some of the schemes for ranking of obligations, in which case our conflict resolution system is suggested as a support system to detect and resolve conflicting objectives in autonomous systems.

References

03.ibm.com/autonomic/pdfs/AC Concepts.pdf (last viewed on 24-Jan-2005).