

Methods for Conflict Resolution in Policy-Based Management Systems

Nicole Dunlop §, Jadwiga Indulska †‡, Kerry Raymond ‡

§ Harrow School of Computer Science, University of Westminster, London, UK

† School of Information Technology and Electrical Engineering,
The University of Queensland, Australia

‡ CRC for Enterprise Distributed Systems Technology (DSTC), Australia

E-mail: dunlop@wmin.ac.uk, jaga@itee.uq.edu.au, kerry@dstc.edu.au

Abstract

While developments in distributed object computing environments, such as the Common Object Request Broker Architecture (CORBA) [17] and the Telecommunication Intelligent Network Architecture (TINA) [16], have enabled interoperability between domains in large open distributed systems, managing the resources within such systems has become an increasingly complex task. This challenge has been considered for several years within the distributed systems management research community and policy-based management has recently emerged as a promising solution.

Large evolving enterprises present a significant challenge for policy-based management partly due to the requirement to support both mutual transparency and individual autonomy between domains [2], but also because the fluidity and complexity of interactions occurring within such environments requires an ability to cope with the coexistence of multiple, potentially inconsistent policies. This paper discusses the need of providing both dynamic (run-time) and static (compile-time) conflict detection and resolution for policies in such systems and builds on our earlier conflict detection work [7, 8] to introduce the methods for conflict resolution in large open distributed systems.

1 Introduction and Motivation

In large evolving enterprises it is often necessary to manage a substantial set of diverse objects, across various organisational boundaries, over a potentially extensive lifetime. Furthermore, users need to be able to adapt to changing circumstances and organisational responsibilities and hence, to make use of new services as they are introduced. When managing these large sets of organisational objects across diverse boundaries conflicting requirements often emerge, which subsequently materialise into policy-based conflicts.

We believe there is limited value in developing policy-based management models that do not provide ensuing support for both policy-based conflict detection and resolution. Our previous work on policy-based conflict detection [8] determined that there exists two broad classes of conflicts that need to be

understood and independently managed; which are *static* and *dynamic* policy-based conflict. It is vital to make the distinction between these two classes of conflicts as the process of conflict detection and resolution is frequently computationally intensive, time-consuming and hence, expensive and is most *preferably* done statically, at compile-time. Not all conflicts, however, can be detected and resolved statically at compile-time and we discovered that detection of conflicts both statically and at run-time relies on a knowledge of the temporal characteristics of the policy (since policy conflicts will only occur when the objectives of two or more active policies cannot be *simultaneously* met). By combining the deontic and temporal properties of policy, we were then able to specify a set of conflict profiles for both static and dynamic conflicts. A series of algorithms for conflict detection were then designed and these were implemented within a prototype that demonstrated that it is possible to detect conflict at run-time in a way that is computationally-efficient [5].

Following the detection of policy-based conflicts, it is necessary to consider the critical issue of dynamic conflict *resolution*. While some work regarding conflict resolution in policy-based management has been attempted [1, 2, 11, 15], our research has revealed that these approaches have not recognised a need for conflict management at run-time and furthermore, some assume that *one* conflict resolution method is sufficient to resolve *all* conflict types, which we have found to be an incorrect assumption.

This paper begins by providing the background to our previous work on conflict detection in Section 2, followed by a discussion of the methods we have developed for conflict resolution in Section 3. Section 4 is a discussion of the prototype system built and Section 5 concludes the paper and discusses future work.

2 Background

This section provides the background to our current work on conflict resolution and commences with an overview of our policy model [6], in Section 2.1, followed by an overview of our previously developed methods for conflict detection [5, 6, 7, 8], in Section 2.2.

2.1 Overview of Policy Model Concepts

Central to our policy model are the concepts of *Enterprise Domain*, *Policy Space*, *Role* and *Policy Authority* which are represented in the simplified model of Figure 1. Enterprise domains are enterprise-level classifications of all the entities (users, roles, artefacts) within the organisation and are a convenient means by which we can *enumerate* objects under common authority or ownership. As such, it is usually a hierarchical division of departments, faculties, teams etc.

The purpose of identifying policy spaces is to combine all of the entities and actions about which we *are interested in writing and applying policy*. A policy space is generally a declarative statement or description of the entities and actions about which policy will be written. Policy spaces are defined as distinct from enterprise domains because the application of policy does not need to follow organisational boundaries. That is, where an enterprise domain might be *Computer Science Department*, a policy space might be *University Parents*, where *University Parents* would be a selection of entities belonging to many domains within the organisation. Furthermore, roles within our policy model may be either static or dynamic, where membership to static roles is an enumeration of selected entities within the organisation, whereas membership to dynamic roles is evaluated at run-time according to some predefined predicate (for example, *Cheapest Supplier*).

Policy authorities in our model are assigned to both enterprise domains and policy spaces. An enterprise domain authority is the assigned owner of resources within the enterprise domain, whereas a policy space authority is one which may write policy regarding the entities and actions contained within the policy space. Identifying policy authorities is essential to the process of conflict resolution, as will be shown in Section 3.

2.1.1 Definition of Policy

The purpose of *policy* is to define or constrain the current or future behaviour of objects to ensure that their actions are aligned with the objectives of the enterprise. While deontic logic expresses many normative notions that could be useful in policy specification, for example, *rights*, *claims*, *liberties*, *privileges*, *power*, *immunity*, *prima facie*, *conditional and defeasible obligations*, *contrary to duty* etc; we choose to focus on the set of modal operators of *obligation*, *permission* and *prohibition*, as defined in standard deontic logic [10] and widely accepted in the literature [4, 15, 20]. These policy modes can be described as follows:

- *Permission* is “the action of permitting an allowance; liberty or a licence granted to do something” [21].
- *Prohibition* is “an unambiguous statement or rule, regulating *disallowed* behaviour in a system” [19].
- *Obligation* is “an agreement, enforceable by law, whereby a person or persons become bound to a particular action or performance of some duty by a contract containing such an agreement” [19].

2.1.2 Policy Notation

We believe that an understanding of the temporal nature of policy is pivotal to both the specification and subsequent classification of conflicts in policy-based management systems. Therefore, we have defined a number of novel policy types (based on the deontic concepts of *obligation*, *permission* and *prohibition*) and the temporal characteristics of the policy type. A complete formal specification of these policy types appears in our earlier work [7].

Within the policy class, the attribute *Policy Name* is used to uniquely identify each policy and the *Policy Description* will be a textual interpretation of the purpose of the policy. The *Policy Type* attribute refers to the deontic notions of *obligation* ($O+$, $O-$), *permission* (P) and *prohibition* (F). The *Temporal Classifier* denotes one of the policy types belonging to either *obligation*, *permission* or *prohibition*, that is, a temporal classifier might indicate that an obligation holds always, at all times, as opposed to an obligation that recurs at a stated time or event. For example, $[P_{always}]$, denotes a continuous permission that holds always, at all times. The related attribute *Commence* is used to identify an instance of the type Occurrence (event or time), at which this policy will commence. Similarly, *Finish* is used to identify an instance of the type Occurrence (event or time), at which this policy will be dismissed and finally, *Recur* is used to identify an instance of the type Occurrence (event or time), at which this policy will recur.

2.2 Overview of Conflict Detection

Policy conflict occurs when the objectives of two or more policies cannot be simultaneously met. In this section we describe the goals of conflict detection and briefly discuss a classification of conflicts and the need for both dynamic and static conflict analysis. A more thorough treatment of this topic appears in our previous work [5, 8].

2.2.1 Goals of Conflict Detection

The fundamental purpose of conflict detection is to analyse policy specifications in order to provide a *profile* of the types of conflict occurring within a system. The goals of conflict detection are characterised as follows:

- to *identify* actual conflict that has occurred and could be resolved statically, at compile-time;
- to *predict* that a potential conflict, may, occur in the future (and more specifically, exactly what circumstances will expose that conflict); and
- to *communicate* the actual or potential conflict to a resolution process.

2.2.2 Classification of Conflicts

We believe conflicts can be classified into four broad categories as defined below. Note that each category of conflict may present itself either statically or dynamically. The first of these categories is the *Internal Policy Conflict* which occurs when the

$$\begin{aligned}
& (\forall p2.F_{time} : time, p2.C_{event}, p1.R_{event} : event | \\
& p1 \xrightarrow{\text{conflict}} p2 \text{ at } \forall p1.R_{event} \text{ between } [p2.C_{event}, p2.F_{time}] | \\
& \quad \text{where trigger } [p2.C_{event}] \quad (8)
\end{aligned}$$

$$\begin{aligned}
& (\forall p2.C_{event}, p2.F_{event}, p1.R_{event} : event | \\
& p1 \xrightarrow{\text{conflict}} p2 \text{ at } \forall p1.R_{event} \text{ between } [p2.C_{event}, p2.F_{event}] | \\
& \quad \text{where trigger } [p2.C_{event}] \quad (9)
\end{aligned}$$

Figure 2. Conflict Database Entry.

Accordingly, from (2), we understand that if policy $p1$ has a *recur* attribute that is of the occurrence type, time, and policy $p2$ has both *commence* and *finish* attributes of the occurrence type, time; then we will need to further assess for **actual** conflict (coinciding with every recurrence of $p1.R_{time}$ within the period referenced by $[p2.C_{time}, p2.F_{time}]$).

If, however, policy $p1$ has a *recur* attribute that is of the occurrence type, event, and policy $p2$ has both *commence* and *finish* attributes of the occurrence type, event; then commensurate with (9), we will need to manage the **potential** for conflict (occurring at every recurrence of $p1.R_{event}$ within the period referenced by $[p2.C_{event}, p2.F_{event}]$). It is clear that policies of these particular occurrence types will need to be managed at run-time, as the realisation of conflict is dependent on the incidence and pattern of events occurring at run-time. Thus, the types of the temporal attributes, *commence*, *finish* and *recur*, inevitably dictate the type of conflict that will need to be managed (ie. actual or potential).

Static and run-time conflict detection relies on a knowledge of the temporal characteristics of the policies in the specification. Through examining each combination of the policy types, we have been able to specify when each of the policy types would cause a conflict (actual and/or potential) [5]. The complete specification of conflicts occurring has been recorded in a conflict database [7] and it was found that the majority of conflicts could not be determined *statically* at specification time, but rather, referred to potential conflicts that could only be determined *dynamically* at run-time. That is, of the 4,238 types of conflicts identified, 3,850 (or 90.84%) were dynamic conflicts, and a mere 388 (or 9.16%) were static conflicts.

3 Conflict Resolution

Once the conflict detection process has produced a specification of all the actual and potential conflict occurring within a system, it becomes necessary to apply methods for conflict resolution. This section begins by describing the goals of conflict resolution in Section 3.1, followed by *when* it is appropriate to resolve conflicts in Section 3.2. Approaches for monitoring conflicts at run-time and for conflict resolution are discussed in Section 3.3 and Section 3.4, respectively.

3.1 Goals of Conflict Resolution

The fundamental purpose of conflict resolution is to determine *when* it is appropriate to resolve conflict and *how* the conflict will be resolved. The goals of conflict resolution can be characterised as follows:

- to *communicate* with the conflict detection process in order to obtain specifications of the actual and/or potential conflict occurring within a system.
- to *decide when* it is appropriate to resolve the conflict;
- to *monitor* identified potential conflict that may occur, requiring resolution;
- to *decide how to resolve* the actual or potential conflict in an appropriate manner.

3.2 When To Resolve Conflict

The process of conflict resolution can be approached either *optimistically*, *pessimistically*, through a combination of both approaches in a *balanced* manner or each conflict can be approached *individually* according to the likelihood of conflict occurring and the cost of conflict resolution.

3.2.1 Pessimistic Conflict Resolution

The pessimistic approach to conflict resolution assumes that *all* conflict states (both actual and potential) *will* result in conflict at some time and therefore must be resolved immediately at compile-time. Since, the pessimistic approach is essentially a preventative measure, it involves much initial checking to ensure that non-compliance does **not** occur. In order to illustrate this method, consider the role definition of the Assistant Human Resources Manager, *assistant hr manager*, represented in (10):

$$R_{assistant_hr_manager} \longrightarrow [P_{ahm1}, P_{ahm2}] \quad (10)$$

It can be seen that the role specification of *assistant hr manager* contains both of the policies specified in Figure 3, including (P_{ahm1}), which states that the Assistant Human Resources Manager is not permitted to access office premises after hours; and the policy (P_{ahm2}), which states that the Assistant Human Resources Manager is obliged to be on office premises when new staff arrive. Through the use of both the conflict database and the methods for conflict detection overviewed in Section 2.2, it would be ascertained that these policy types (that is, F_{within} and O_{recur_always}), exhibit the **potential** for conflict at every recurrence of ($P_{ahm2}.R_{event}$) within the period referenced by $[P_{ahm1}.C_{time}, P_{ahm1}.F_{time}]$, that is whenever the event “New Staff Arrival” occurs within the period [10.00pm - 6:00am].

Furthermore, policies of these particular occurrence types should be managed at run-time, since the realisation of conflict is dependent on the incidence and pattern of events occurring at run-time. However, the pessimistic approach requires that

```

Policy "P_ahm1" {
  PolicyName: "Assistant HR Manager - Prohibited Office Access."
  PolicyType: "F"
  Subject: StaticRole "Assistant HR Manager"
  Action: Action "Access Office Premises"
  Target: Organisation "Brisbane CBD Office"
  TemporalClassifier: "F_within"
  Commence: Time "22:00:00"
  Finish: Time "6:00:00" }

```

```

Policy "P_ahm2" {
  PolicyName: "Assistant HR Manager - Orientation."
  PolicyType: "O+"
  Subject: StaticRole "Assistant HR Manager"
  Action: Action "Access Office Premises"
  Target: Organisation "Brisbane CBD Offices"
  TemporalClassifier: "O_recur_always"
  Recur: ExternalEvent "New Staff Arrival" }

```

Figure 3. Policy (P.ahm1) and (P.ahm2).

all conflicts (both actual and potential) be resolved at compile-time. Nevertheless, it is apparent that resolving this potential conflict at compile-time would probably be quite wasteful, since it is highly unlikely that new staff will arrive at the office after hours.

While the pessimistic approach often involves greater cost at compile-time this can be said to be an advantage of the approach since incurring the cost of resolving conflict at compile-time is generally preferable to run-time conflict resolution. However, the number of conflicts occurring in large, open distributed environments (both actual and potential) can be very large and resolving all such conflicts, would become a costly exercise because much of the effort expended in resolving **all** identified conflicts, a-priori, would prove to be unnecessary. It is often more practical to tolerate the existence of potential conflicts and manage the occurrence of conflict if it materialises. Furthermore, the pessimistic approach to conflict resolution does not allow for the evolving and essentially dynamic nature of large open distributed systems. Nevertheless, pessimistic conflict resolution is useful in situations where conflict cannot be tolerated, for example, high-security arrangements or where the cost of resolving conflict at run-time cannot be tolerated.

3.2.2 Optimistic Conflict Resolution

Optimistic conflict resolution does not involve any preventative measures but relies instead on detecting non-compliance and correcting it. As a result, the optimistic approach often requires greater effort and cost at run-time to detect all such non-compliance. Conflict must therefore occur in a run-time environment before a resolution is initiated. We considered each class of conflicts individually, that is, *prohibition / permission* - (F / P); *obligation / prohibition* - (O+ / F) and *obliged / not obliged* - (O+ / O-) as detailed in [6], however, we will focus on one such example here, which will be the *prohibition / permission* - (F / P) conflict class, described below.

Prohibition / Permission - (F / P)

In order to describe how the optimistic approach behaves with respect to the (F / P) conflict class, consider the role definition specified in (11):

$$R_{systems_administrator} \longrightarrow [P_{sa1}, P_{sa2}] \quad (11)$$

The role of *systems_administrator* contains both of the policies (P_{sa1}) and (P_{sa2}), specified in Figure 4. The policy (P_{sa1}) asserts that the Systems Administrator is permitted to archive the organisation's file servers as required; and the policy (P_{sa2}), states that due to reduced network performance, the Systems Administrator is not permitted to archive the organisation's file servers during business hours. Through examination of the conflict database it will be determined that these particular policy types (i.e. P_{always} and F_{within}), will exhibit actual conflict between [$P_{sa2}.C_{time}$, $P_{sa2}.F_{time}$], that is, between the hours of [9.00am - 5.00pm].

```

Policy "P_sa1" {
  PolicyName: "Systems Administrator - Reboot File Servers."
  PolicyType: "P"
  Subject: StaticRole "Systems Administrator"
  Action: Action "Archive"
  Target: Artefact "File_Server"
  TemporalClassifier: "P_always" }

```

```

Policy "P_sa2" {
  PolicyName: "Systems Administrator - Archive File Server."
  PolicyType: "F"
  Subject: StaticRole "Systems Administrator"
  Action: Action "Archive"
  Target: Organisation "File_Server"
  TemporalClassifier: "F_within"
  Commence: Time "9:00am"
  Finish: Time "5:00pm" }

```

Figure 4. Policy (P.sa1) and (P.sa2).

In this example, the optimistic approach would monitor the actions occurring at run-time to see if the permitted action of *archive file_servers* takes place between the prohibited business hours before a state of conflict would be declared and the subsequent conflict resolution is performed. This is reasonable since the systems administrator may never perform an archive operation during business hours, therefore the resolution of such conflict would be unnecessary.

If, however, the situation were such that a permission policy (P_{p1}), was *fully* subsumed by a prohibition policy (P_{p2}), as illustrated in Figure 5, it becomes more difficult to see the value of the optimistic approach since there is **no** scope in this particular set of circumstances for the permitted action to take place without producing a conflict state. Nevertheless, the advantage of the optimistic approach to conflict resolution, is that *no* conflict is resolved that does not *specifically* warrant it. The same cannot be said for the pessimistic approach, and to a lesser extent the balanced approach following.

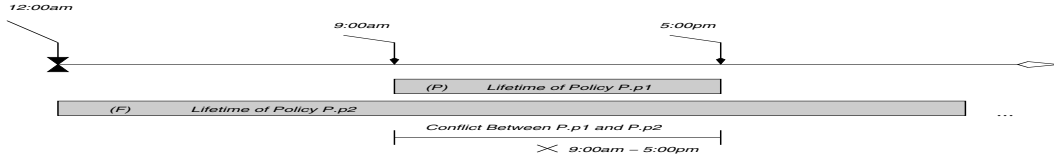


Figure 5. Permission Fully Subsumed by Prohibition.

3.2.3 Balanced Conflict Resolution

The balanced approach to conflict resolution assumes that the likelihood of *actual* conflict occurring is quite high, and thus, resolves it statically (at compile-time); thereby reducing some the cost of identifying, monitoring and resolving such conflict in a run-time environment. Whereas identified *potential* conflict is not resolved at compile-time, but instead is monitored and resolved in the run-time environment *if required*. In order to illustrate such an approach, consider the role definition of *sales manager*, as specified in (12):

$$R_{sales_manager} \rightarrow [P_{sm1}, P_{sm2}] \quad (12)$$

The role *sales_manager* contains the both the policy (P_{sm1}) and (P_{sm2}) specified in Figure 6. The policy (P_{sm1}) asserts that the Sales Manager is permitted to drive a company-assigned vehicle during business hours and the policy (P_{sm2}), states that the Sales Manager is not permitted to drive a company-assigned vehicle during weekend hours. Through examination of the conflict database [7], it will be determined that these particular policy types (i.e. P_{within} and F_{within}), that have commence and finish occurrence attributes of type *time*; will exhibit *actual* internal policy conflict between (Saturday, 8:00am - Saturday 6:00pm) and again during (Sunday, 8:00am - Sunday 6:00pm). Following the balanced approach to conflict resolution, this identified *actual* conflict would be resolved immediately at compile-time.

However, if the previous example of potential conflict in Section 3.2.1 was used, such conflict between (P_{ahm1}) and (P_{ahm2}) would not be resolved unless (at run-time) the unlikely situation occurred where a new staff member arrived after hours. We believe that the balanced approach is appropriate for large open distributed environments, due to the fact that the number of both actual and potential conflicts can be very large. However, the algorithms to support the balanced approach are more complex than those required for the pessimistic approach (due to the requirement to monitor potential conflicts at run-time).

3.2.4 Individual Conflict Resolution

It can be concluded from the previous discussion regarding the *optimistic*, *balanced* and *pessimistic* approaches to conflict resolution, that computational cost at compile-time is substantially higher in the pessimistic approach as compared to the balanced and optimistic approaches, since *all* conflict is removed prior to run-time in the pessimistic approach. The balanced approach

```

Policy "Psm1" {
  PolicyName: "Sales Manager - Company Vehicle Business Hours."
  PolicyType: "P"
  Subject: StaticRole "Sales Manager"
  Action: Action "Drive_Vehicle"
  Target: Artefact "432-AXY"
  TemporalClassifier: "Pwithin"
  Commence: Time "8:00"
  Finish: Time "18:00" }

Policy "Psm2" {
  PolicyName: "Sales Manager Company Vehicle Weekend."
  PolicyType: "F"
  Subject: StaticRole "Sales Manager"
  Action: Action "Drive_Vehicle"
  Target: Organisation "432-AXY"
  TemporalClassifier: "Fwithin"
  Commence: Time "Saturday 0:00am"
  Finish: Time "Monday 0:00am" }

```

Figure 6. Policy (P.sm1) and (P.sm2).

distinguishes between actual conflict and the potential for conflict and resolves only the actual conflict prior to run-time, as a result, a comparatively medium-level computational cost is incurred at compile-time. Whereas, the optimistic approach resolves *no* conflict until such conflict physically presents itself in the run-time environment. Hence, the very low computational cost at compile-time for the optimistic approach.

The *individual conflict resolution* process attempts to assess each detected conflict separately in order to determine whether the conflict should be resolved immediately at compile-time or deferred until run-time. In order to assess each conflict *individually*, it is necessary to consider both the *likelihood* of conflict occurring (statistical analysis), and *cost* of resolving or not resolving the conflict (consequence analysis) as discussed below.

Likelihood of Conflict Occurring (Statistical Analysis)

In order to assess the likelihood of conflict occurring it is necessary to consider each conflict type individually, that is, *internal policy conflict*, *external policy conflict*, *role conflict* and *policy space conflict*. Furthermore, it is also interesting to consider each conflict class separately that is, (P/F), (O+/F) and (O+/ O-) [6]. However, for the purposes of this paper, we will consider internal policy conflict within the (O+/O-) conflict class only.

Internal Policy Conflict (O+ / O-)

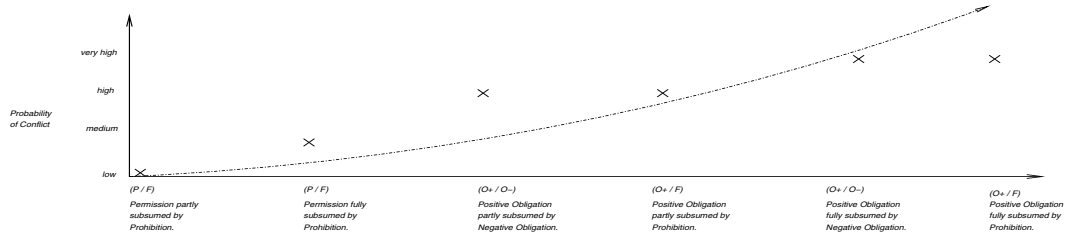


Figure 7. Probability of Conflict Within Conflict Classes.

For the (O+/O-) conflict class we first consider the situation where a positive obligation has been **fully** subsumed by a negative obligation. It can be seen from Figure 7, that the likelihood of conflict occurring in such situations is comparatively *very high* and hence, under the individual conflict resolution method, it would be determined that this type of conflict should be resolved immediately at compile-time. However, in the case where a positive obligation has been **partly** subsumed by a negative obligation; again according to Figure 7, such conflict has a comparatively *high* probability of occurring. Therefore, further examination regarding how much scope exists for the obligation to be performed, will be required.

Consider for example, Figure 8, where it can be seen that even though the obligation cannot be fulfilled between (9:00am - 10:00am), it can be fulfilled between (12:00am - 9:00am) or during (10:00am - 6:00pm). In this case, since there is extensive scope for fulfilling the obligation outside of the conflict area (9:00am - 10:00am), it could be concluded that resolution of this particular conflict in these specific circumstances, could be delayed until run-time. Whereas in the case of Figure 9, it can be seen that conflict arises between (9:00am - 5:00pm) and that while there *is* scope for the obligation to be fulfilled between (5:00pm - 6:00pm), it could be sensibly concluded that for this particular obligation; the (5:00pm - 6:00pm) window of opportunity (for the obligation to be fulfilled) is too small and/or inappropriate in the circumstances. Therefore it could be decided that this conflict should be resolved a compile-time in order to avoid this seemingly inevitable conflict.

Cost of Conflict Resolution (Consequence Analysis)

Another factor that needs to be considered relates to the cost of performing the resolution as well as the cost of delaying resolution until run-time. In some circumstances, it is noted that the emergence of *uncertain* conflict states at run-time is clearly undesirable (for example in high security arrangements). Nevertheless, the cost of individually assessing every actual and potential conflict within a system to determine when resolution is appropriate will mean that the implementation complexity of such a system will be very high compared to the *pessimistic*, *optimistic* and *balanced* approaches to resolution, due to the need to progress through a series of involved decisions and analysis regarding each and every identified conflict within the system.

3.3 Monitoring Conflict at Run-Time

The requirement to monitor conflict at run-time occurs when either the *balanced*, *optimistic* or *individual* approaches to conflict resolution are chosen, because in all of these approaches, it is possible to defer the resolution of potential conflict until run-time.

3.3.1 Run-Time Phases of Potential Conflict

In order to monitor conflict at run-time, a series of run-time phases that a potential conflict will progress through before it becomes actual have been identified. These phases are *inactive*, *active* and *pending*. In order to describe the potential conflict phases, consider the role definition specified in (13) following:

$$R_{farm_hand} \longrightarrow [P_{fh1}, P_{fh2}, P_{fh3}] \quad (13)$$

In this case the role specification R_{farm_hand} contains the policies (P_{fh1}), (P_{fh2}) and (P_{fh3}) where policy (P_{fh1}) states that users assigned to the role Farm Hand are not permitted to source water for irrigation from the *central* dam when water levels are unacceptable (that is, Central Dam Level < 10,000 ML); similarly the policy (P_{fh2}) states that when the *central* dam water levels are unacceptable, then Farm Hands will be obliged to source their irrigation water from the supporting *auxiliary* dam, until water levels in the *central* dam rise again to acceptable levels (that is, Central Dam Level \geq 10,000 ML). Finally, policy (P_{fh3}), states that only when the *auxiliary* dam water level has depleted (that is, Auxiliary Dam Level < 1,000 ML), will Farm Hands be permitted to return to sourcing water from the *central* dam.

Through the process of conflict detection we know that the **potential** for conflict arises at every recurrence of (P_{fh3} .*Revent*) within the period referenced by [P_{fh1} .*Cevent*, P_{fh1} .*Fevent*]. That is, the potential conflict will arise whenever the event auxiliary dam level is depleted (i.e. Auxiliary Dam Level < 1,000 ML) occurs between the events, Central Dam Level Unacceptable (i.e. Central Dam Level < 10,000 ML), and Central Dam Level Acceptable (i.e. Central Dam Level \geq 10,000 ML). Therefore, in this example, the **phases** that the potential conflict between policies (P_{fh1}) and (P_{fh3}) will progress through at run-time in order to become a realised, actual conflict, as illustrated in Figure 11 are described as follows - if the water levels in the *central* dam are acceptable (that is, Central Dam Level \geq 10,000 ML), which occurs at Time 1, see Figure 11, then the potential conflict between the policies (P_{fh1}) and (P_{fh3}) is said to be

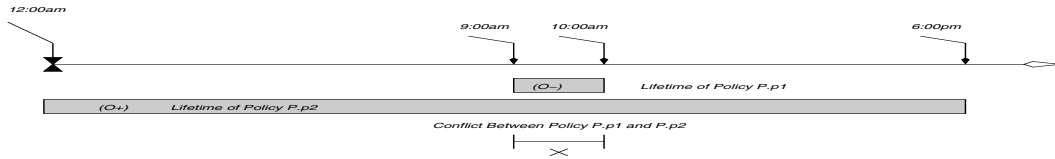


Figure 8. Adequate Scope for Obligation to Occur.

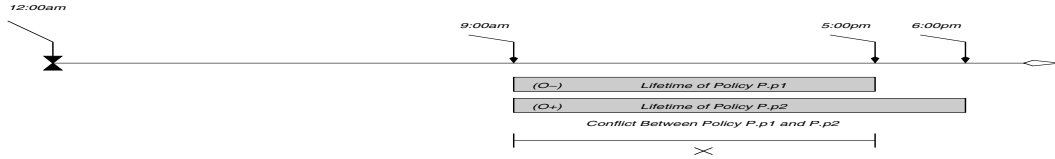


Figure 9. Inadequate Scope for Obligation to Occur.

in an *inactive* state. If however, the event (Central Dam Level $< 10,000$ ML) occurs (as at Time 2), then this potential conflict will progress from the *inactive* state to the *pending* state. Therefore, if the event (Auxiliary Dam Level $< 1,000$) occurs before the *central* dam has reached acceptable levels again (i.e. Central Dam Level $\geq 10,000$ ML), then the *pending* conflict will progress to its final state of *active*, realised conflict (as at Time 3).

If a potential conflict progresses through to the *active* conflict state, such conflict requires run-time conflict resolution. It is therefore necessary to monitor potential conflict at run-time as it progresses through the phases of *inactive*, *pending* and *active*

3.3.2 Indexed Event Databases

In order to aid the process of conflict monitoring at run-time a series of indexed databases were created, including the *active events index* and the *run-time conflict index*. The *active events index* contains a series of *event* and *status* tuples which represent a list of run-time events and their current status as either *active* or *inactive*.

The *run-time conflict index*, as represented in (14), contains the set of attributes: *policy_references*, *initiate_event*, *activate_event*, *dismiss_event*, *status*; which reflect an indexed list of current, potential run-time conflicts. The attribute *policy_references* indicates which policies will be in conflict, the *initiate_event* refers to the event that will force a conflict from the status of “inactive” to “pending”. The *activate_event* forces a conflict from a state of “pending” to a state of “active” conflict and finally the *dismiss_event* dissolves the conflict back to the state of “inactive”. At all times, the attribute *status* is either *active*, *pending* or *inactive*.

$$\text{run-time_conflict_index} < \text{policy_references, initiate_event, activate_event, dismiss_event, status} > \quad (14)$$

In order to explain the usage of the indexed events databases we will consider the *balanced* approach to conflict resolution

where actual conflict is resolved immediately at compile-time and potential conflict is deferred until run-time. In this case, once the conflict profile from the conflict database has been retrieved for a set of policies, the potential conflicts between such policies will be included in the *run-time conflict index*, describing which events will initiate, activate and dismiss this conflict. The advantage of using such databases, is that when events are occurring at run-time the index can be checked to determine which conflicts will be affected by the event. Therefore the need to examine every combination of active policies and events at run-time is eliminated.

3.4 How to Resolve Conflict

The process of resolving conflict fundamentally involves either retracting and rewriting the conflicting policies or determining which policy will take precedence in the event that an actual conflict materialises. Retracting and rewriting policy involves removing the conflicting policies and attempting to rewrite them so they no longer produce a conflict state. However, the process of retracting and rewriting policies is not always feasible since such action will often interfere with the autonomy of the policy spaces that are responsible for specifying the behaviour of the entities and actions, with which they are associated.

3.4.1 Establishing Precedence

A more practical method of resolving conflict is therefore to identify which policy involved in a conflict situation will take precedence. In order to address this issue, a discussion regarding the methods that can be used to establish precedence in conflict situations and how the established precedence is declared will follow.

Considering again the balanced approach to conflict resolution establishing policy precedence will be necessary for every identified actual conflict and only for the potential conflict that materialises at run-time. As discussed in Linington *et al* [14], a number of principles that commonly form the foundation of legal frameworks in many countries (originally identified in

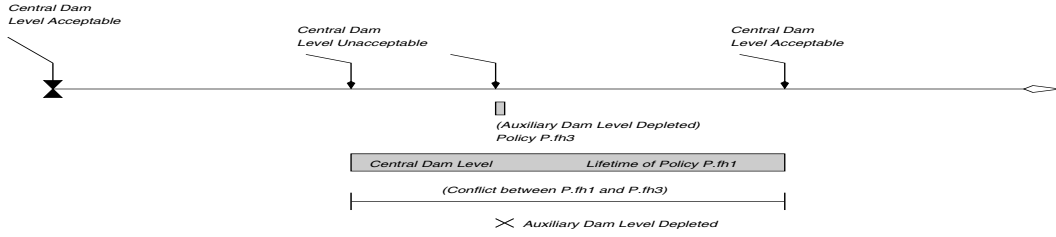


Figure 10. Conflict Between Policies (P.fh1) and (P.fh3).

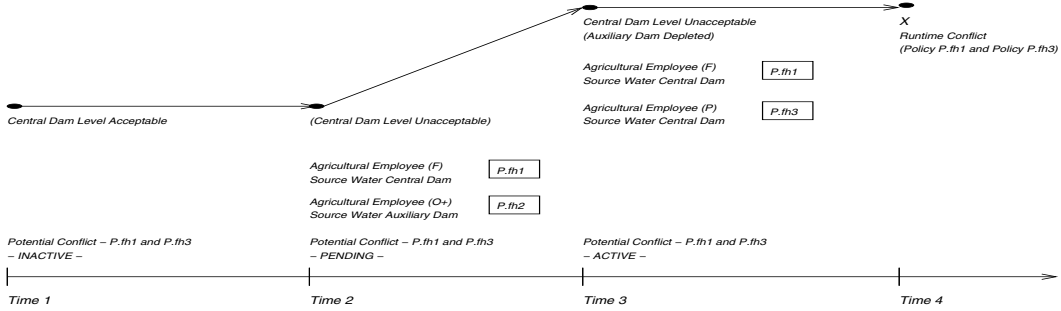


Figure 11. Run-Time Phases of Conflict Between Policies (P.fh1) and (P.fh3).

den Haan [9]), include the following: *lex specialis legi generali derogat* - the specific overrides the general; *lex posterior legi anteriori derogat* - new law overrides old law; and *lex superior legi inferiori derogat* - the higher authority overrides the lower authority. Such principles are interesting as they can be applied to the problem of establishing precedence in policy-based conflict situations. In addition to these principles, we will also discuss the principles of assigning *explicit weights or priorities* to policies; and *negative/positive* policy precedence. Each of these principles will be described in the following sections.

Specific Overrides General

The principle of the *specific overrides the general* has most recently been utilised by Larrondo-Petrie *et al* [13]; where conflicts between authorisation policies in object-oriented databases are resolved through calculating the *distance* between a policy and the set of managed objects it references. In order to illustrate this process, consider the role definitions of *academic_staff* and *professor* represented in (15) and (16) respectively, which have been assigned the policies of (P_{as1}) and (P_{p1}), as specified in Figure 12 following:

$$R_{academic_staff} \rightarrow [P_{as1}] \quad (15)$$

$$R_{professor} \rightarrow [P_{p1}] \quad (16)$$

If it is considered that all entities assigned to the role of *professor* are also assigned to the general role of *academic_staff*, then external policy conflict will occur between the policies of (P_{as1}) and (P_{p1}), for every individual assigned to the role of *professor*. Therefore, in order to resolve the conflict between these policies using the principle of the *specific overrides the general*; if an entity ($e1$) were assigned to the role of *professor* and

```

Policy "Pas1" {
  PolicyName: "Academic Staff - Funding Consulting Projects."
  PolicyType: "F"
  Source: StaticRole "Academic Staff"
  Action: Action "Assign Funding"
  Destination: Artefact "External Consulting Project"
  TemporalClassifier: "Falways" }

Policy "Pp1" {
  PolicyName: "Professor - Funding Consulting Projects."
  PolicyType: "P"
  Source: StaticRole "Professor"
  Action: Action "Assign Funding"
  Destination: Artefact "External Consulting Project"
  TemporalClassifier: "Palways" }

```

Figure 12. Policy (P.as1) and (P.p1).

hence, also to the role of *academic staff*, then the distance between the *academic_staff* policy (P_{as1}) and the entity ($e1$) would be greater than the distance between the *professor's* policy (P_{p1}) and the entity ($e1$). Accordingly, the principle of *specific overrides general* would determine that the policy applying to the more specific class of *professors*, (P_{p1}), will take precedence over the policy applying to the more general class of *academic staff*, that is policy (P_{as1}).

However, while the *specific overrides the general* resolution method is entirely appropriate in this case, it is not difficult to discover situations where such a resolution method yields quite inappropriate results. Consider for example, that the role definition *academic_staff* was altered to include a policy that prohibits *academic staff* from travelling to international conferences between the events Travel Budget Depleted, Travel Budget Replete and the role definition of *professor* is altered

to include a policy that permits professors to travel to international conferences at all times.

In this case conflict will result for every entity assigned to the role of *academic_staff* and *professor* when the departmental budget reaches depleted levels. According to the resolution method *specific overrides general*, the conference travel policy associated with *professors*, will override the more general policy associated with *academic staff*. This will result in the undesirable situation of *professors* being able to continue international conference travel when the departmental budget has reached depletion.

Furthermore, it is also possible to identify situations where the *specific overrides general* resolution method cannot determine which policy will take precedence in the event of conflict. Such a situation arises where neither policy in the conflict situation can be deemed to be more specific than the other.

New Law Overrides Old Law

The principle of *new law overrides old law* can also be employed as a conflict resolution method and involves associating all policies with a creation date. Therefore, when conflict between a set of policies arises, the policy with the latest creation date will take precedence over the the policy with the earlier creation date.

Similar to the *specific overrides general* resolution method, the *new law overrides old law* principle resolves conflict suitably in some cases and inappropriately in others. For example, if the preceding role definitions of *academic_staff* and *lecturer* are considered, as defined in (15) and (16) respectively are considered, then it is known that policies (P_{as1}) and (P_{p1}), as specified in Figure 12, would be in a state of external policy conflict for all entities assigned to the role *professor*, at all times. Consequently, in order to resolve the conflict between these policies using the principle of *new law overrides old law*, the creation dates of both policies would be examined. If it were the case that the policy associated with *professors*, (P_{p1}) had a creation date that was earlier than the creation date of the policy associated with *academic_staff*, (P_{as1}), then the policy associated with general *academic_staff* would override the policy associated with *professors*. Thus, resolution of this particular conflict would be deemed unsatisfactory, as *professors* should in fact, be able to assign funding to external consulting projects.

Higher Authority Overrides Lower Authority

For the purpose of establishing precedence between conflicting policies, the conflict resolution method based on the principle of *higher authority overrides lower authority* is arguably the most instinctive of the three principles identified in Section 3.2. Nevertheless, like the preceding principles of *specific overrides general* and *new law overrides old law*, unfortunately, cases where the principle of *higher authority overrides lower authority* produces inappropriate results, can also be established. Considering again, the previous role definitions of *academic_staff* and *lecturer*, as defined in (15) and (16), if it were the case that the University's Vice-Chancellor issued the policy (P_{as1}) and the Head of Department (a lower authority) issued policy (P_{p1});

then according to the principle of *higher authority overrides lower authority*, the policy related to general *academic_staff*, (P_{as1}) issued by the Vice-Chancellor, would override the policy assigned to *professors*, (P_{p1}) by the Head of Department. Again, such resolution would be undesirable as it is preferred that *professors* be able to assign funding to external consulting projects.

Furthermore, it is also understood that in cases where conflicting policies have been issued by the **same** authority, or where authorities are not in a hierarchical relationship, the principle of *higher authority overrides lower authority* is of no use in resolving the conflict situation.

Assigning Explicit Weights or Priorities to Policies

The principle of *priority assignment*, primarily involves the assignment of individual weighting or priorities to each specified policy. These priority values are intended to be used to define an order of precedence between conflicting policies. While this approach is quite effective when considered within the scope of a single policy space, this approach would be exceptionally difficult to implement within a large open distributed environment and in order to illustrate why such difficulty arises, consider the policy space definitions of the *faculty of engineering* and the *school of computer science*, as represented in (17) and (18) respectively. It can be seen that policy space, *faculty of engineering* has been assigned the policy (P_{hod1}) and policy space, *school of computer science* has been assigned the policy ($P_{scholarship1}$), as specified in Figure 13 following.

$$PS_{faculty_of_engineering} \longrightarrow [(E_{cs_hod}, E_{cs_postgraduates}, \dots E_z), (A_{approve_scholarship}, \dots A_z), (P_{hod1}, \dots P_z)] \quad (17)$$

$$PS_{school_of_computer_science} \longrightarrow [(E_{cs_hod}, E_{cs_postgraduates}, \dots E_z), (A_{approve_scholarship}, \dots A_z), (P_{scholarship1}, \dots P_z)] \quad (18)$$

```
Policy "Phod1" {
  PolicyName: "Head of Department - Scholarship Approval."
  PolicyType: "F"
  Source: StaticRole "CS Head of Department"
  Action: Action "Approve Scholarship"
  Destination: DynamicRole "CS Postgraduates"
  TemporalClassifier: "Falways" }
```

```
Policy "Pscholarship1" {
  PolicyName: "CS Head of Department - Scholarship Approval."
  PolicyType: "P"
  Source: StaticRole "CS Head of Department"
  Action: Action "Approve Scholarship"
  Destination: DynamicRole "CS Postgraduates"
  TemporalClassifier: "Palways" }
```

Figure 13. Policy (P.hod1) and (P.scholarship1).

Furthermore, the policies (P_{hod1}) and ($P_{scholarship1}$) will be in a state of conflict with each other at all times. Therefore, according to the principle of *priority assignment*, the policy assigned the greater priority or weight would take precedence in this conflict situation. The difficulty in using this principle for large open distributed systems then arises because the authority of the policy space *faculty of engineering*, may assign the policy (P_{hod1}) a greater priority or weight than the authority of policy space *department of computer science*, would assign the policy ($P_{scholarship1}$). Thus, ensuring that (P_{hod1}) always overrides policy (P_{hod1}), which may not be the resolution that either policy space authority intends. Since policy spaces operate largely autonomously, the assignment of individual policy priorities could easily result in arbitrary priorities being assigned to policies and inappropriate precedence orders being determined. Due to these limitations the principle of *priority assignment* would therefore, only be viable for determining a precedence order between conflicting policies within a **single** policy space.

Interestingly, however, the principle of *priority assignment* has been repeatedly identified in the literature as being an adequate method for resolving conflict between policies, including, amongst others, the Multi-Policy Security approach by Bidan *et al* [2] and the Temporal Role-Based Access Control (TRBAC) framework by Bertino *et al* [1].

Negative/Positive Policy Precedence

The assumption of *negative policy precedence* requires that in conflict states, the negative policy will always take precedence over the positive policy, similarly the principle of *positive policy precedence*, ensures that where a conflict situation arises, the positive policy will always take precedence over the negative policy. Consider for example, the previous policies (P_{as1}) where *academic staff are not permitted to assign funding to external consulting projects*, and policy (P_{p1}) which states that *professors are permitted to assign funding to external consulting projects*, as defined in Figure 12. If the conflict between these policies were to be resolved using the *negative policies take precedence* method, then in this case all *academic staff*, including the *professors*, would not be able to assign funding to external consulting projects, which is not an appropriate outcome. However, like previously discussed precedence establishment methods, this method can yield appropriate results in other circumstances.

The *negative/positive policy precedence* principle has been applied for the purposes of conflict resolution between policies within the Unified Framework for Multiple Access Control Policies by Jajodia *et al* [11].

3.4.2 Establishing Precedence Summary

In view of the fact that the methods for establishing precedence in conflict situations discussed above have been shown to produce valid results in some circumstances and invalid results in others, it is the conclusion of this paper that it is *inappropriate* to suggest that *one* precedence establishment method is adequate to resolve all conflicts within large open distributed environments and that *several* precedence establishment methods

would need to be employed.

We have further found that it is useful to make an initial determination regarding whether the conflict is contained within a single policy space, or whether the conflict spans multiple policy spaces. This is important since if the conflict is wholly contained within a single policy space, the policy space authority (as defined in Section 2.1), becomes a suitable entity to determine which precedence establishment method is appropriate for the conflict in question. Consequently, the policy space authority will be at liberty to impose any method of precedence establishment it deems appropriate for the actions and entities within its policy space and may chose a single method for precedence establishment and apply that to all of the conflicts within its policy space, or it may chose to determine precedence on an individual conflict basis.

If however, the policy space authority is constrained by obligation to another policy space, that is, the conflict spans more than one policy space then a determination at the organisational-level as to how to establish precedence in order to resolve the conflict will become appropriate. That is, an organisation may determine that conflict which spans multiple policy spaces may be resolved by allowing the policy space with the most senior policy space authority to override the policies of the conflicting policy spaces (alternatively, another of the precedence establishment methods described previously may be chosen). If no authority between policy spaces can be determined (i.e. neither policy space is senior to the other) then an appeal to the *owner* of the resources, or the *enterprise domain authority* will be made. Similarly, the *enterprise domain authority* is at liberty to impose any method of precedence establishment it deems appropriate for the actions and entities within its enterprise domain.

In any case, regardless of the method chosen at either the policy space level or the enterprise domain level, once precedence has been established between conflicting policies, then this determination must be declared between the policies, so if the conflict situation arises again, the existing resolution can be referred to and the conflict automatically resolved. It will not be necessary to go through the process of determining an appropriate resolution again.

4 Prototype

The work undertaken on this thesis has incorporated the development of a prototype to demonstrate how dynamic conflict is detected, monitored and resolved [5]. The Java-based implementation (of approximately 160,000 lines of code) uses the Object Management Group's Meta-Object Facility (MOF) [18] to manage meta-information, and in this case, to store the policy-based information model. Instances of the policy model are parsed and stored using an implementation of the Human-Usable Textual Notation (HUTN) [22].

By simulating the application of policies over a sequence of run-time events, the prototype has demonstrated that:

- the *algorithms* developed for managing conflict detection and resolution *behave correctly*.

- the *algorithms* for detecting and resolving conflict are computationally feasible.
- both the *policy model and the policy language* developed, are *sufficient* to address the challenges of conflict detection and resolution.

Furthermore, the implementation of a prototype system provided valuable experience in experimenting with different approaches for managing consistency in policy-based management systems, which proved to be very useful in refining the final algorithms for conflict detection and resolution. This implementation has demonstrated that, through using the methods established in this paper for the specification and management of policy-based conflict detection and resolution, that policy-based management systems are technically-feasible in large open distributed environments.

5 Conclusion and Further Work

The primary goal of our work has been to ensure that the *consistency* of policy-based management systems is maintained despite the evolution of organisational policies, roles etc. The background to our current work on conflict resolution was presented which included a brief description of both our policy model and the methods employed for conflict detection. Our current work regarding conflict resolution was the focus of this paper and our methods for both conflict detection and resolution have been tested through the development of a prototype implementation.

We discovered while developing methods for conflict resolution that *several* techniques are necessary in order to deal with the varied range of policy-based conflicts and the suitability of each technique is dependent on the type of conflict it is attempting to resolve. We have found that because conflict occurs not only within the scope of a single policy space, but also across organisational boundaries, that it becomes imperative to understand the nature of the relationships that exist between policy spaces and enterprise domains (*for example*, senior/dependent, co-operating, independent) [6]. Furthermore, we have found that representing authorities within our policy model has been key to resolving conflict between multiple domains.

A number of areas of possible future work have been identified at the conclusion of this paper, including the examination of a broader range of deontic normative notions (e.g. *interdictions, liberties, rights, power, prima facie obligations*), for the specification of policy types. Given that further policy types were defined it would then be necessary to examine the subsequent *conflict types* that result. Furthermore, the area of *policy refinement* need further work as policies are considered to exist at many different levels of abstraction and the transformation process from high-level policy to low-level implementable has remained a largely unresolved problem.

6 Acknowledgements

The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology

(DSTC) through the Australian Federal Government's CRC Programme (Department of Industry, Science and Resources).

References

- [1] E. Bertino, C. Bettini, E. Ferrari, P. Samarati, "An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning", ACM Transactions on Database Systems, Volume 23, Number 3, September 1998.
- [2] C. Bidan, V. Issarny, "Dealing with Multi-Policy Security in Distributed Systems", Proceedings of the Fifth European Symposium on Research in Computer Security (ESORICS '98), Volume 1485, Lecture Notes in Computer Science (LNCS), Belgium, September 1998.
- [3] M. Chechik, D. O. Păun, "Events in Property Patterns", Theoretical and Practical Aspects of SPIN Model Checking, Lecture Notes in Computer Science, Volume 1680, Springer-Verlag, September, 1999.
- [4] F. Chen, R. S. Sandhu, "Constraints for Role-Based Access Control", Proceedings of the First ACM/NIST Role-Based Access Control Workshop (RBAC '95), Maryland, USA, December, 1995.
- [5] N. Dunlop, "Dynamic Policy-Based Management in Open Distributed Environments", Ph.D. Thesis, University of Queensland, Brisbane, Australia, September, 2002.
- [6] N. Dunlop, J. Indulska, K. A. Raymond, "Dynamic Policy Model for Large Evolving Enterprises", Proceedings of the Fifth International Conference on Enterprise Distributed Object Computing (EDOC 2001), Seattle, Washington, USA, September, 2001.
- [7] N. Dunlop, J. Indulska, K. A. Raymond, "A Formal Specification of Conflicts in Dynamic Policy-Based Management Systems", DSTC Technical Report, CRC for Enterprise Distributed Systems, University of Queensland, August, 2001.
- [8] N. Dunlop, J. Indulska, K. A. Raymond, "Dynamic Conflict Detection for Large Evolving Enterprises", Proceedings of the Sixth International Conference on Enterprise Distributed Object Computing (EDOC 2002), Lausanne, Switzerland, September, 2002.
- [9] N. den Haan, "Investigations into the Application of Deontic Logic", Executable Modal and Temporal Logics, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93), August 1993, LNCS Volume 897, Springer-Verlag.
- [10] R. Hilpinen (editor), "Deontic Logic : Systematic Readings", D. Reidel Publishing Company, Dordrecht, Holland, 1971.
- [11] S. Jajodia, P. Samarati, V. S. Subrahmanian, "A Logical Language for Expressing Authorizations", Proceedings of the IEEE Symposium on Security and Privacy, Oakland, California, USA, May 1997, IEEE Press.
- [12] L. Lamport, "The Temporal Logic of Actions", Proceedings of the ACM Transactions on Programming Languages and Systems (ACM TOPLAS), Volume 16, Issue 3, ACM Press, May 1994.
- [13] M. M. Larrondo-Petrie, E. Gudes, H. Song, E. B. Fernandez, Security Policies in Object-Oriented Databases, IFIP Database Security, III: Status and Prospects, North-Holland, 1990.
- [14] P. F. Linington, Z. Milosevic, K. A. Raymond, "Policies in Communities: Extending the ODP Enterprise Viewpoint", Proceedings of the Second International Enterprise Distributed Object Computing Workshop (EDOC '98), La Jolla, California, USA, November 1998.
- [15] E. C. Lupu, M. S. Sloman, "Conflicts in Policy-Based Distributed Systems Management", IEEE Transactions on Software Engineering - Special Issue on Inconsistency Management, 1999.
- [16] H. Mulder, "(TINA) - Business Model and Reference Points - Version 4.0", Telecommunications Intelligent Network Architecture Consortium, 2002.
- [17] Object Management Group (OMG), "The Common Object Request Broker Architecture and Specification, CORBA/IIOP, 2.6.1 Specification. Specification [formal/02-05-15]", April, 2000.
- [18] Object Management Group (OMG), "Complete MOF 1.3 Specification (MOF 1.3) formal/00-04-03", April, 2000.
- [19] Oxford University Press, "Oxford English Dictionary", January, 2000.
- [20] R. S. Sandhu, "Role-Based Access Control", Advances in Computing, Volume 46, Academic Press, 1998.
- [21] C. Schwarz, G. Davidson, A. Seaton, V. Tebbit, W. and R. Chambers Limited and Cambridge University Press, "Chambers Cambridge English Dictionary", Edinburgh, UK, 1998.
- [22] J. Steel, K. A. Raymond, "Generating Human-Usable Textual Notations for Information Models", Proceedings of the Fifth International Conference on Enterprise Distributed Object Computing (EDOC 2001), Seattle, Washington, USA, September, 2001.