Policies in a Resource Manager of Workflow Systems: Modeling, Enforcement and Management

Yan-Nong Huang, Ming-Chien Shan
Software Technology Laboratory
HPL-98-156
September, 1998

E-mail: [ynhuang,shan]@hpl.hp.com

workflow, resource, policy, interval-based, query rewriting

This paper proposes a new method to handle policies in Resource Management of Workflow Systems. Three types of policies are studied including qualification, requirement and substitution policies. The first two types of policies map an activity specification into constraints on resources that are qualified to carry out the activity. The third type of policy intends to suggest alternatives in cases where requested resources are not available. An SQL-like language is used to specify policies. Policy enforcement is realized through a query rewriting based on relevant policies. A novel approach is investigated for effective management of large policy bases, which consists of relational representation of policies and efficient retrieval of relevant policies for a given resource query.

Internal Accession Date Only

Policies in a Resource Manager of Workflow Systems: Modeling, Enforcement and Management

Yan-Nong Huang and Ming-Chien Shan
Hewlett-Packard Laboratories
1501 Page Mill Road, 1U-4A
Palo Alto, California 94304
Email: ynhuang@hpl.hp.com, shan@hpl.hp.com

Abstract This paper proposes a new method to handle policies in Resource Management of Workflow Systems. Three types of policies are studied including qualification, requirement and substitution policies. The first two types of policies map an activity specification into constraints on resources that are qualified to carry out the activity. The third type of policies intends to suggest alternatives in cases where requested resources are not available. A SQL-like language is used to specify policies. Policy enforcement is realized through a query rewriting based on relevant policies. A novel approach is investigated for effective management of large policy bases, which consists of relational representation of policies and efficient retrieval of relevant policies for a given resource query.

Keywords Workflow, Resource, Policy, Interval-Based, Query Rewriting.

1 Introduction

An information system is composed of a database system and one or many applications manipulating the database. The database system is a common data repository shared among multiple applications. Besides data, people sometimes move components which seemly belong to applications into the database, so that multiple applications can share common components. In other words, the common components become part of the database's semantics. Active databases are an example of such kind, where dynamic characteristics of data are pushed down to the database, so the data can always behave the same way no matter what applications are.

We are interested in Workflow Management Systems (WFMS) [5] [11], and particularly, in Resource Management (RM) [6] of WFMS. A WFMS consists of coordinating executions of multiple activities,

instructing *who* (resource) do what (activity) and *when*. The "when" part is taken care of by the workflow engine which orders the executions of activities based on a process definition. The "who" part is handled by the resource manager that aims at finding suitable resources at the run-time for the accomplishment of an activity as the engine steps through the process definition.

Resources of different kinds (human and material, for example) constitute the information system of our interest, their management consists of resource modeling and effective allocation upon users' requests. Since resource allocation needs to follow certain general guidelines (authority, security, for example) - no matter who or what application issues requests: so those general guidelines are better considered as part of the resources' semantics. That is the reason why we are interested in resource policy management in RM. Resource policies are general guidelines every individual resource allocation must observe. They differ from process specific policies which are only applied to a particular process. The policy manager is a module within the resource manager, responsible for efficiently managing a (potentially large) set of policies and enforcing them in resource allocation.

We propose to enforce policies by *query rewriting*. A resource query is sent to the policy manager where relevant policies are first retrieved, then either additional selection criteria are appended to the initial query (in the case of requirement policies) or a new query is returned (in the case of substitution policies). Therefore, the policy manager can be seen as both a *regulator* and a *facilitator* where a resource query is either "polished" or given alternatives in a controlled way before submitted for actual resource retrieval. By doing so, returned resources can always be guaranteed to fully comply with the resource usage guidelines.

1.1 Design Goals for the Policy Management

Technical issues involved in this study can be presented as the following set of design goals.

1. A simple policy model allowing users to express relationships between an activity and a resource that can be used to carry out the activity;

2. A Policy Language (PL) allowing users to *define* policies; PL must be easy to use and *as close as possible* to SQL;

3. Resource query *enhancement/rewriting*: a major functionality of the policy manager is to enforce policies by enhancing/rewriting the initial resource query;

4. *Efficient* management of policy base: retrieving relevant policies applicable to a given resource query may become time-consuming when dealing with a large policy base, strategies are to be devised to achieve good performance.

1.2 An Overview on Related Work

Policy, as a broad term, has been used in system management of different kinds (see for example,[1], [9], [4], [7]). Roughly speaking, all policies in a system constitute a set of constraints (which can rigorously be expressed as a Boolean expression), upon which "legal" actions or "consistent" states are defined. General enough in the conceptual terms, we believe though policies need to be dealt with on a case by case basis; because systems of different contexts may have significantly different discourse domains.

Policy management has been considered in the workflow management. For example, [1] and [3] discussed policies for resource allocation. Our work is different from [1] or [3], in that:

1. They basically deal with process specific policies whereas our focus is on the general resource policies.

2. Our policy model is more general. In our model, a policy is composed of an activity and a resource, activities and resources are organized into two classification hierarchies. So a policy involving a more general activity and/or a more general resource is applicable to any specific activity or resource.

3. Our implementation is more scalable than theirs. Policies are managed in a relational database, efficient accesses to a large set of policies are guaranteed by an effective indexing on the policy tables.

4. Our policy model is finer in that given an activity, the user not only can specify the resource type that can be used to carry out the activity but also properties qualified resources should hold. In the meantime, our policy language is simpler and close to SQL.

Rule management has been a research issue in other information management fields like active databases or deductive databases. Rules in these fields usually involve conditions and actions to be fired once the conditions are met. Actions normally generate new states of the database, which can eventually trigger other actions. A similar scenario occurs with RM when requested resources are not available, RM could suggest, through *substitution policies* (C.f., 3.3), alternatives as replacement. If none of the alternatives were available, another round of replacements would be necessary. In other words, resource substitution could eventually take place recursively. Note that RM usually substitutes user-requested resources by *compromising* some of the initial requirements. It is clear that one does not want any compromise to continue "indefinitely". Therefore, we choose *not* to substitute the requested resources more than once before notifying success or failure of the query.

The challenge we are facing, however, is related to the management of a potentially large set of policies.

So far little attention has been paid in the literature to managing effectively *large* volume of policies, despite the increasing prominence of the problem as the management task becomes more and more

sophisticated. Representing policies in a *computerized* way so that relevant policies can be retrieved efficiently for any given situation is our major concern.

1.3 Paper Organization

The rest of the paper is organized in the following way. In Section 2, we give a brief overview of our resource manager, with the purpose of showing how the present research is positioned in its context. In Section 3, three types of policies for resource management are presented and the policy language is illustrated with examples. Query rewriting for policy enforcement is discussed in Section 4. Issues on managing effectively large policy bases are elaborated in Section 5. An analytical evaluation is given in Section 6. Some conclusive remarks are drawn in Section 7. The syntax of the resource query language and the policy language is given in Appendix.

2 Context

In this section, we briefly discuss the context of the present research.

2.1 Architecture

Two main components exist in our Resource Manager (Figure 1). One is the resource manager per se, responsible for modeling and managing resources; the other is the policy manager allowing the user to manage policies. Three interfaces are offered, each obviously requiring a different set of access privileges. The policy language interface allows one to insert new policies and consult existing ones. With the resource definition language interface, users can manipulate both meta and instance resource data. Finally, the resource query language interface allows the user to express resource requests.

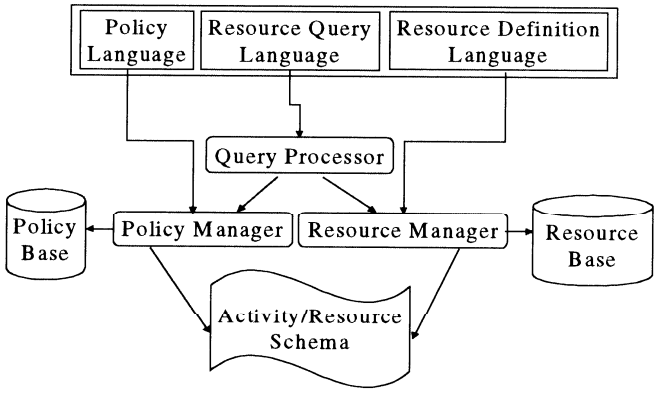

Figure 1: Architecture

Upon receiving a resource query, the query processor dispatches the query to the policy manager for policy enforcement. The policy manager first rewrites the initial query based on *qualification policies* and generates a list of new queries. Each of the new queries is then rewritten, based on *requirement policies*, into an enhanced query. The enhanced new queries are finally sent to the resource manager for resource retrieval.

In the cases where none of the requested resources is available, the *initial* query is re-sent to the policy manager which, based on *substitution policies*, generates alternatives in the form of queries. Each of the alternative queries is treated as a new query, therefore has to go through both *qualification* and *requirement* policy based rewritings. Then, the policy manager once again sends a list of resource queries to the resource manager.

If no relevant resources are found against the rewritten alternative queries, notify the user of the failure. Bear in mind that substitution policies should *not* be used *transitively*.

2.2 Resource and Activity Models

A role is intended to denote a set of capabilities, its extension is a set of resources sharing the same capabilities. In this regard, a role can be seen as a resource type. The resource hierarchy shows resources organized into roles whilst the activity hierarchy describes the classification of activity types.

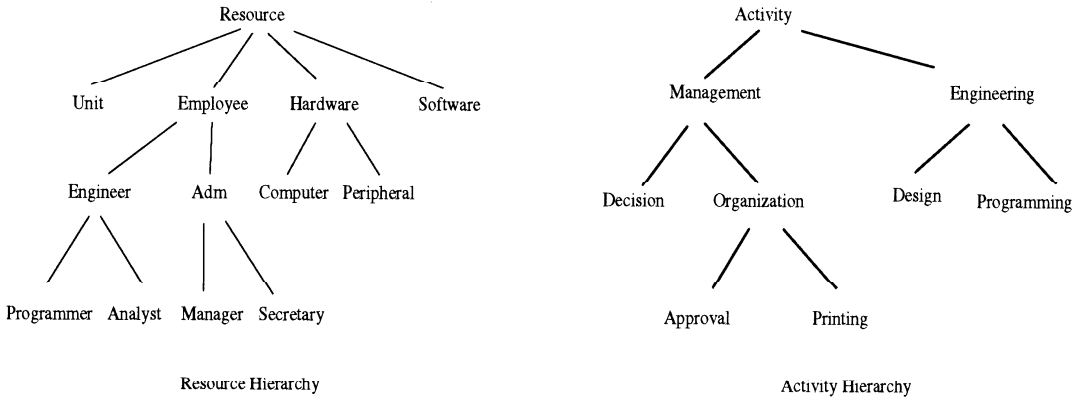

Figure 2: Resource and Activity Classifications

Figure 2 shows an example of resource and activity hierarchies.

A resource type as well as an activity type is described with a set of attributes, and all the attributes of a parent type are inherited by its child types.

In addition to the resource classification, the resource manager holds relationships among different types of resources.

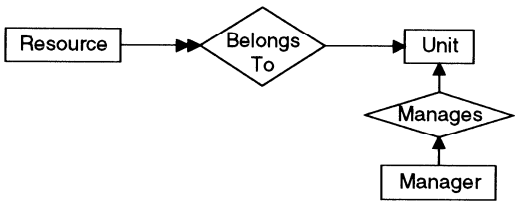

Figure 3: Entity-Relationship Model of Resources

Two possible relationships between resources are exemplified in Figure 3. Note that, like attributes, relationships are inherited from parent resources to child resources.

Views may be created on relationships to facilitate query expressions. For example, ReportsTo(Emp, Mgr) is defined as a join between BelongsTo(Employee, Unit) and Manages(Manager, Unit) on the common attribute Unit.

2.3 Resource Query Language

Users can use the resource query language (RQL) to submit resource requests to the resource manager. The language is composed of SQL *Select* statements augmented with activity specifications.

```
Select   ContactInfo
From     Engineer
Where    Location = 'PA'
For      Programming
With     NumberOfLines = 35000 And Location = 'Mexico'
```

Figure 4: Initial RQL Query

The query in Figure 4 requests ContactInfo of *Engineer* located in 'PA', for activity *Programming* of 35,000 line code and of location 'Mexico'.

Since a resource request is always made upon a known activity, the activity can and should be fully described; namely, each attribute of the activity is to be specified.

3 Policy Model and Language

Three types of policies are considered: qualification policies, requirement policies and substitution policies.

3.1 Qualification Policies

A qualification policy states a type of resources is *qualified* to do a type of activities.

```
Qualify   Programmer
For       Engineering
```

Figure 5: A Qualification Policy

The policy in Figure 5 states the resource type *Programmer* can do the activity type *Engineering*. Since resources and activities are partially ordered, we allow qualification policies to be inherited from parent resources or activities to their children. Consider a general qualification policy "*Qualify* R *for* A", what this policy really means is any sub-type resource of R (including R itself) is qualified to do any sub-type activity of A (including A itself).

All qualification policies in the policy base are *Or-related*, and they obey the *Closed World Assumption* (CWA). Namely, if the policy in Figure 5 is the only policy in the policy base, we may assume no resource types other than *Programmer* can do activity *Engineering*.

3.2 Requirement Policies

A requirement policy states that if a *resource* is chosen to carry out an *activity* with specified characteristics, the *resource* must satisfy certain conditions. Therefore, it expresses a necessary condition for a *resource* type and an *activity* type. All requirement policies in the policy base are *And-related*.

Here are examples of requirement policies:

Require	*Programmer*
Where	Experience > 5
For	*Programming*
With	NumberOfLines > 10000

Require	*Employee*
Where	Language = 'Spanish'
For	*Activity*
With	Location = 'Mexico'

Figure 6: Requirement Policies

The first policy in Figure 6 states that if a *Programmer* is chosen to carry out activity *Programming* of more than 10,000 line code, it is required that the *Programmer* have more than 5 year experience.

Given that both the set of resources and the set of activities are partially ordered, the scope of a requirement policy can stretch over resources and activities which are sub-types of the resource and the activity explicitly mentioned in the policy. For example, the second policy in Figure 6 requires the *Employee* be Spanish speaking if (s)he is engaged in activity *Activity* located in Mexico. Since both *Employee* and *Activity* have sub-types in their respective hierarchies (Figure 2), the policy is actually applicable to any pair of resource and activity as long as the resource is a sub-type of *Employee* (including *Employee* itself) and the activity is a sub-type of *Activity* (including *Activity* itself). This gives a great deal of flexibility to expressing requirement policies.

The syntax of a general requirement policy is as follows:

Require	*R*
Where	<Where>
For	*A*
With	<With>

Figure 7: General Requirement Policy

There, *R* is a resource type and *A* is an activity type. <Where> is a SQL *where* clause which can eventually include nested SQL *select* statements. <With> is a restricted form of SQL *where* clause in which no nested SQL statements are allowed.

Some more complex policy examples follow,

```
Require Manager
Where   ID = (
            Select  Mgr
            From    ReportsTo
            Where   Emp = [Requester]
        )
For     Approval
With    Amount < 1000
```

```
Require Manager
Where   ID = (
            Select      Mgr
            From        ReportsTo
            Where       level = 2
            Start with  Emp = [Requester]
            Connect by  Prior Mgr = Emp
        )
For     Approval
With    Amount > 1000 And Amount < 5000
```

Figure 8: Complex Requirement Policies

Both policies in Figure 8 relate resource *Manager* to activity *Approval*. The first states that if the amount requested for approval is less than $1,000, the authorizer should be the manager of the requester. The second policy (a hierarchical sub-query is used) requires that the authorizer be the manager's manager if the requested amount is greater than $1,000 and less than $5,000. Two points are worth mentioning,

1. Nested SQL statement can be used to construct more complex selection criteria.

2. Attributes of the activity can be referenced in constructing selection criteria. To distinguish an attribute of the activity from that of the resource, the former is enclosed between [and]. In the examples of Figure 8, Requester is an attribute of activity *Approval*.

3.3 Substitution Policies

A substitution policy is composed of three elements: a *substituting resource*, a *substituted resource* and an *activity*; each eventually augmented with descriptions. It states that the substituting resource can replace the substituted resource in the unavailability of the latter, to carry out the activity. Multiple substitution policies are *Or-related*.

Substitute	*Engineer*
Where	Location = 'PA'
By	*Engineer*
Where	Location = 'Cupertino'
For	*Programming*
With	NumberOfLines < 50000

Figure 9: A Substitution Policy

The policy in Figure 9 states that *Engineers* in PA, in their unavailability, can be replaced by *engineers* in Cupertino to carry out activity *Programming* of less than 50,000 line code.

Similar to the requirement policy, the scope of a substitution policy can stretch over resources and activities which are sub-types of the substituted resource and the activity mentioned in the policy. Therefore, the policy in Figure 9 may eventually be applicable to a query looking for a *Programmer* for activity *Programming*.

4 Query Rewriting

Three types of policies imply that a RQL query may go through three different stages of query rewritings under different circumstances. In general, any RQL query is automatically submitted for rewritings based on qualification policies and relevant requirement policies (in the order). In the cases where no available resources are found, the initial query is rewritten based on relevant substitution policies. The three rewritings are discussed below.

4.1 Rewriting Based on Qualification Policies

This query rewriting uses qualification policies to adjust the initial query. The outcome could be a list of queries.

Consider the RQL query in Figure 4, which looks for an *Engineer* for activity *Programming*. Assume the only qualification policy is the one in Figure 5; namely, among the three sub-types *Programmer*, *Analyst* and *Engineer* of *Engineer*, only *Programmer* can carry out the super-type activity *Engineering* of *Programming*. Therefore, the initial RQL query is rewritten as:

Select	ContactInfo
From	*Programmer*
Where	Location = 'PA'
For	*Programming*
With	NumberOfLines = 35000 And Location = 'Mexico'

Figure 10: Rewriting Based on Qualification Policies

where *Engineer* is replaced by *Programmer*.

In general, given a RQL query looking for a resource R for an activity A, R is replaced by each of its sub-types (could be R itself) which, according to the qualification policies, can carry out one of the super-type activities of A (could be A itself too). If none of the sub-types of R can be used to carry out any of the super-type activities of A, the empty set is returned.

Two points are worth mentioning:

1 The qualification policy based rewriting generates a list of resource queries.

2 A resource mentioned in the initial query implies *all* the sub-type resources. In the query of Figure 4, the intention is to retrieve either an engineer or a programmer or an analyst to do the specified job. In contrast, a resource mentioned in each of the rewritten queries *excludes* its *proper* sub-type resources.

4.2 Rewriting Based on Requirement Policies

This query rewriting consists of retrieving *all* requirement policies *applicable* to the RQL query, appending additional selection criteria (*where* clauses of the requirement policies) imposed by each of these requirement policies to the *where* clause of the query. The outcome of this rewriting is an enhanced query.

As discussed in 3.2, a requirement policy involves a resource type and an activity type. A policy is said to be *applicable* or *relevant* to a RQL query, if,

1 the resource in the policy is a super-type of the resource in the query; and,

2 the activity in the policy is a super-type of the activity in the query; and,

3 the activity specification in the query falls within the activity range of the policy.

Note that super-types of a type discussed above include the type itself.

The RQL query in Figure 10 requests resource *Programmer* for activity *Programming*. The first policy in Figure 6 involves *Programmer* and *Programming*, so it might be applicable to the query. To actually apply the policy, one has to check if the activity specification of the query fits into the activity ranges of the policy. The specification of the query includes "NumberOfLines = 35000" which falls within the range "NumberOfLines > 10000" of the policy, so "Experience > 5" will be added as a new criterion to enforce the policy.

The second policy in Figure 6 involves resource *Employee* and activity *Activity*. Since *Employee* is a super-type of *Programmer* and *Activity* is a super-type of *Programming*, this policy could be applied to the query. Because the activity specification of the query matches the activity of the policy, "Language = Spanish" is added as another selection criterion.

Thus the rewritten query is:

Select	ContactInfo
From	*Programmer*
Where	Location = 'PA' And
	Experience > 5 And Language = 'Spanish'
For	*Programming*
With	NumberOfLines = 35000 And Location = 'Mexico'

Figure 11: Rewriting Based on Requirement Policy

The last two selection criteria in the *where* clause in Figure 11 are derived from applying policies on the activity specification.

4.3 Rewriting Based on Substitution Policies

This query rewriting consists of finding all substitution policies *applicable* to the RQL query, then substituting the resource (together with its specification, namely, the *from* and *where* clauses of the query.) based on each of these policies. So, the outcome of this rewriting could be a list of queries.

As discussed in 3.3, a substitution policy involves a substituted resource, a substituting resource and an activity. A substitution policy is said to be *applicable* or *relevant* to a RQL query, if,

1 the substituted resource in the policy has *at least one common sub-type* with the resource in the query[1]; and,

[1] Keep in mind that a resource type mentioned in the initial query implies *all* the sub-type resources.

2 the resource range in the query *intersects with* the resource range in the policy; and,

3 the activity in the policy is a super-type of the activity in the query; and,

4 the activity specification in the query falls within the activity range in the policy.

Note that super-types of a type discussed above include the type itself.

Consider now rewriting the initial query (Figure 4) based on substitution policies. The initial query involves resource *Engineer* and activity *Programming*; the substitution policy in Figure 9 has *Engineer* as the substituted resource and *Programming* as the activity, so it might be applicable to the query.

Like in the case of requirement policies, to actually apply the policy, one has to check if the specifications of the activity and resource in the query fit into the ranges of the activity and the substituted resource in the policy. The specification of the query includes "NumberOfLines = 35000" which falls within the range "NumberOfLines < 50000" of the policy, also the location of requested *Engineer* being 'PA' which matches the substituted resource in the policy. Consequently, the substitution policy in Figure 9 turns out to be relevant to the initial RQL query.

Thus the rewritten query is:

```
Select   ContactInfo
From     Engineer
Where    Location = 'Cupertino'
For      Programming
With     NumberOfLines = 35000 And Location = 'Mexico'
```

Figure 12: Rewriting Based on Substitution Policy

5 Policy Management

Qualification policies basically deal with relationships among resource and activity types, they can therefore be adequately managed in a 3-column table of schema (PID, Resource, Activity).

We are primarily concerned about managing requirement and substitution policies. This consists of representing and efficiently retrieving *relevant* policies for a given query.

Recall that a requirement policy is relevant to a RQL query if the resource in the policy is a super-type of the resource in the query; the activity in the policy is a super-type of the activity in the query; and the activity specification in the query falls within the activity range of the policy.

Recall that a substitution policy is relevant to a RQL query if the substituted resource in the policy has *at least one common sub-type* with the resource in the query; the resource range in the query *intersects with* the resource range in the policy; the activity in the policy is a super-type of the activity in the query; and the activity specification in the query falls within the activity range in the policy.

Managing either of the two types of policies needs to represent ranges (of resources or activities) in such a way that range comparisons can be carried out efficiently. Given the similarities of requirement policies and substitution policies in terms of management, we only deal with requirement policies in the remainder of this section.

5.1 Policy Representation

In a naïve approach, requirement policies are represented in a 4-column table where each column corresponds to a component of a policy. This works fine with string-match, as is the case with activity or resource types; but is not adequate for range comparisons. A better mechanism ought to be investigated for the activity range representation. Since an activity range is a Boolean expression involving activity attributes, the problem therefore becomes how to represent Boolean expressions in a relational data model.

We first normalize a Boolean expression into a disjunctive normal form, then split the requirement policy into several ones, each holding a conjunctive component in the *with* clause. That is, <A, R, $r_1 \vee r_2$, *WhereClause*>[2] is divided into <A, R, r_1, *WhereClause*> and <A, R, r_2, *WhereClause*>. Consequently, we only need to consider representing requirement policies with conjunctive expressions in the *with* clause. Because predicates involved in the *with* clause are of the form: (*attribute* op *value*), where op is among ($\leq$, $<$, $=$, $>$, $\geq$), negative predicates can be represented by positive ones by reversing the inequality for the cases of inequalities, or replacing $\neg$(*attribute* = *value*) by (*attribute* > *value*) $\vee$ (*attribute* < *value*) for the cases of equalities. By grouping together predicates involving the same attribute, one can realize that the *with* clause can be represented as a set of intervals, each corresponding to an attribute of the activity. Finally, since we deal with finite data domains, all open intervals on a finite domain can be represented with closed ones; so only closed intervals are considered. By convention, we use ">" to denote "greater than or equal to" and "<" to "less than or equal to".

The above analysis suggests us to store an activity range as a set of intervals rather than a string. Furthermore, since the number of intervals may vary from one activity range to another, one would have to

---

[2] A requirement policy is denoted as a quadruple <A, R, r, WhereClause> where A is the activity, R is the resource, r is the activity range and WhereClause is the *Where* clause.

allocate a maximum number of columns to represent activity ranges if one wanted to use one table to store all requirement policies. This approach obviously is not optimal because of the potentially low rate of space occupancy.

We propose to use two tables to represent requirement policies. Precisely, a requirement policy is split into two parts which are associated through a unique policy ID (PID). One table holds the correspondences among the activity type, resource type and *Where* clause while the other table manages the relationships between the set of intervals and the PID. The first table has the schema Policies(PID: Number; Activity: String; Resource: String; NumberOfIntervals: Number; WhereClause: String) whereas the second Filter(PID: Number; Attribute: String; LowBound: String; UpperBound: String[3]).

Given the policy in Figure 7, a tuple is inserted into table Policies while a number of tuples are added to table Filter. That is, (pid, A, R, n, <Where>) is added to table Policies, where pid is an automatically generated integer uniquely identifying a policy, A and R are the activity and resource involved in the policy, n is the number of intervals in <With>.

n tuples are inserted to table Filter. In general, if an attribute a is mentioned in <With> and ranged within an interval [lower, upper], tuple (a, pid, lower, upper) is added to table Filter.

For example, to represent the first requirement policy in Figure 6, (100, 'Programming', 'Programmer', 1, 'Experience > 5') is inserted to table Policies and (100, 'NumberOfLines', 10000, Max[4]) is inserted to table Filter, supposing 100 is the automatically generated PID. And, to represent the second requirement policy in Figure 6, (200, 'Activity', 'Employee', 1, 'Language = Spanish') is inserted to table Policies and (200, 'Location', 'Mexico', 'Mexico') is inserted to table Filter, where 200 is the automatically generated PID.

5.2 Relevant Policy Retrieval

Given a RQL query, assume the activity specification (*With* clause) is: $(a_1 = x_1)$ And $(a_2 = x_2)$ And ... And $(a_m = x_m)$. To retrieve relevant policies, one needs to work on table Policies to figure out policies involving activities and resources that are super-types of the activity and resource of the query; in the meantime,

---

[3] Note that attributes can have different data types, and different data types have different ordering schemes. In the implementation, intervals of different data types are stored in different tables. However, for the ease of understanding, we here only consider intervals of the *string* type.

[4] Max denotes the maximum value of the concerned attribute type.

search applicable policies in table Filter against the activity specification of the query. In doing so, two views are created.

```
Create View Relevant_Policies(PID, NumberOfIntervals, WhereClause)
As
        Select  PID, NumberOfIntervals, WhereClause
        From    Policies
        Where   Policies.Activity in Ancestor(A) And
                Policies.Resource in Ancestor(R)
```

Figure 13: View on Policies

In Figure 13, Ancestor(A) denotes ancestors of A in the activity hierarchy whereas Ancestor(R) the set of ancestors of R in the resource hierarchy. If both hierarchies are not too "big" (which is most likely the case in practice), the inclusion check can be implemented as a group of disjunctively related equality comparisons (between Policies.Activity and each member of Ancestor(A); or between Policies.Resource and each member of Ancestor(R)).

Since attributes *Activity* and *Resource* are always mentioned at the same in the query (Figure 13), for better performance, we may create a concatenated index on attributes *Activity* and *Resource*.

The query in Figure 14 intends to count, for each policy, the number of intervals that enclose an attribute value of the activity specification.

```
Create View Relevant_Filter(PID, NumberOfIntervals)
As
 Select  PID, Count(*)
 From    Filter
 Where   (Attribute = a1 And LowerBound < x1 And x1 < UpperBound)
         Or
         (Attribute = a2 And LowerBound < x2 And x2 < UpperBound)
         Or
         ... ...
         Or
         (Attribute = am And LowerBound < xm And xm < UpperBound)
 Group by PID
```

Figure 14: View on Filter

Similarly, since attributes *attribute*, *LowerBound* and *UpperBound* are always mentioned at the same in the query (Figure 14), for better performance, we may create a concatenated index on attributes *attribute*, *LowerBound* and *UpperBound*.

The query in Figure 15 allows one to retrieve additional selection criteria for the RQL query.

```
Select    WhereClause
From      Relevant_Policies, Relevant_Filter
Where     Relevant_Policies.PID = Relevant_ Filter.PID And
          Relevant_Policies.NumberOfIntervals =
                    Relevant_ Filter. NumberOfIntervals
Union
Select    WhereClause
From      Relevant_Policies
Where     Relevant_Policies.NumberOfIntervals = 0
```

Figure 15: Retrieval of Additional Selection Criteria

Since a requirement policy with empty *with* clause has 0 occurrence in table Filter; so as long as the resource of the policy is a parent resource of the requested resource and the activity of the policy is a parent activity of the request activity, the policy becomes relevant. Hence, retrieval of relevant policies has to take into account both "normal" and "less normal" cases, as was done in Figure 15.

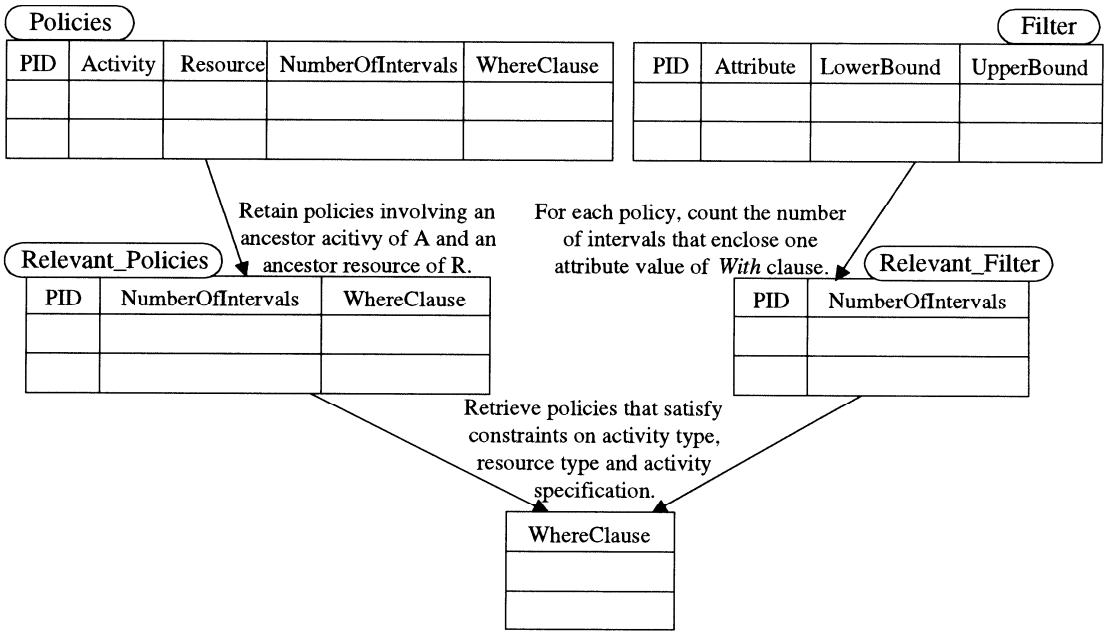

Figure 16: Retrieval of Relevant Policies

Figure 16 summarizes the operation flow for retrieving relevant requirement policies for a given RQL query.

6 Analytical Evaluation

As discussed in 5.2, a concatenated index is created on both tables (Policies and Filter) representing requirement policies, so several alternative execution plans are possible for the query optimizer to process the query intended to retrieve relevant policies. To get a sense of the basis the optimizer makes choices on, and how they are impacted by different policy settings, we evaluation the *selectivity* rates of both views defined in 5.2 (Figure 13 and Figure 14).

Let's first define the parameters.

- |A|: Number of activity types.
- |R|: Number of resource types.
- q: Average number of activity types a resource type is qualified for. This is actually the average number of qualification policies a resource type is involved in.
- c: Average number of different "cases" per pair (resource, activity). This is the average number of requirement policies sharing the same resource and activity types.
- N: Number of requirement policies.
- i: Average number of intervals per activity range.

The number of entries in table Policies is $N = |R| \times q \times c$. If both the activity and resource hierarchies form a complete binary tree, the average number of predecessors of a resource type is $\log|R|$[5] and the average number of predecessors of an activity type is $\log|A|$. So, the selectivity rate on table Policies is:

$$\text{Selectivity}_{\text{Policies}} = \frac{(\log|A|)\times(\log|R|)\times c}{|R|\times q\times c} = \frac{(\log|A|)\times(\log|R|)}{|R|\times q}$$

The number of entries in table Filter is $|R| \times q \times c \times i$. If among all the requirement policies an activity type participates in, the ranges of the activity type are the same for different resource types, and the ranges are pair-wise disjoint; the selectivity rate on table Filter would be:

$$\text{Selectivity}_{\text{Filter}} = \frac{q\times i}{|R|\times q\times c\times i} = \frac{1}{|R|\times c}.$$

---

[5] In a complete binary tree of height n, the average height is:

$$\frac{n\times 2^n + (n-1)\times 2^{(n-1)} + \ldots + 2^1}{2^n + 2^{(n-1)} + \ldots + 2^0} = \frac{(n-1)\times 2^{(n+1)} + 2}{2^{(n+1)} - 2} \approx (n-1).$$

We are interested in the two substantial factors that have impact on the selectivity. The first factor, measured by parameter q, describes how *interactive* it is between the set of activities and the set of resources. The second factor, measured by parameter c, describes how *fragmented* an activity type is in dealing with one particular resource type. Let's now consider the cases where $N = 2^{12}$, $|A| = |R| = 2^6$.

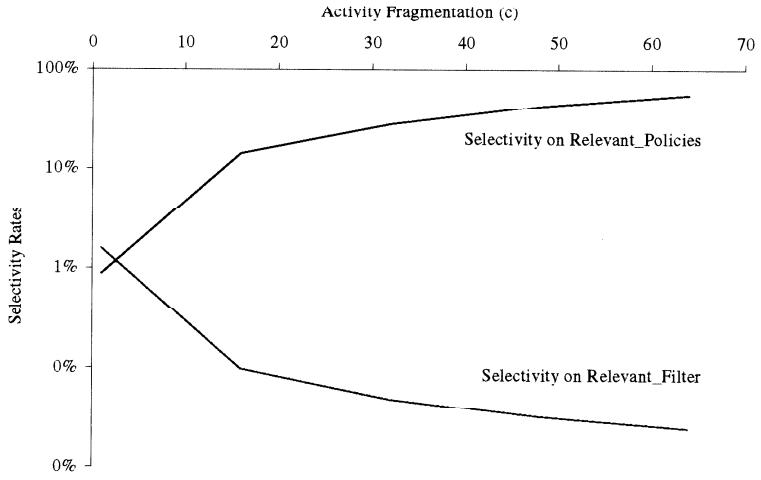

Figure 17: Selectivity Evaluation

Figure 17 depicts the selectivity trends on function of the activity fragmentation, i.e., the average number of different "cases" per pair (resource, activity). When N and |R| are fixed, q is anti-proportional to c. One can observe that the more an activity gets fragmented (c increases), the higher is the selectivity on Relevant_Filter (the selectivity rate getting lower) and the lower is the selectivity on Relevant_Policies. Also, view Relevant_Filter tends to be more selective than Relevant_Policies, in general.

These observations provide some guidelines if one chooses to implement an in-memory query processor not leveraging any commercial in-disk DBMS.

7 Conclusion

We studied several issues related to resource policies in Workflow Systems. A policy language was proposed allowing users to specify policies of different types. To enforce the policy, a resource query is first rewritten based on relevant policies, before submitted to the resource manager for actual retrieval. The originality of the present work is on the resource policy model, the policy enforcement mechanism and policy management techniques including relational representation of, and efficient access to, a large policy set. It seems that the interval-based representation proposed in the paper provides a general framework for effective storage and efficient retrieval of large Boolean expression sets.

A prototype was implemented in Java on NT 4.0, with experimental policies managed in an Oracle database. An alternative implementation would load policies into the main memory (periodically or at start-up time), an in-memory query optimizer ought to be devised in this case. Comparisons of pros/cons of these two implementations are worth further investigating.

References

[1] M. Blaze, J. Feigenbaum and J. Lacy, "Decentralized Trust Management", Proc. of IEEE Symposium on Security and Privacy, Oakland, CA, May 1996.

[2] C. Bufler, "Policy resolution in Workflow Management Systems", Digital Technical Journal, Vol. 6, No. 4, 1994.

[3] C. Bufler and S. Jablonski, "Policy Resolution for Workflow Management Systems", Proc. Of the Hawaii International Conference on System Sciences, Maui, Hawaii, January 1996.

[4] Desktop Management Task Force, "Common Interface Model (CIM) Version 1.0 (Draft)", December 1996.

[5] J. Davis, W. Du and M. Shan, "OpenPM: An Enterprise Process Management System", IEEE Data Engineering Bulletin, 1995.

[6] W. Du, G. Eddy and M.-C. Shan, "Distributed Resource Management in Workflow Environments", Proc. of Database Systems for Advanced Applications (DASFAA), Melbourne, Australia, April, 1997.

[7] R. Grimm, T. Hetschold, "Security Policies in OSI-Management Experience from the DeTeBerkom Project BMSec ", Computer Networks and ISDN Systems, Vol. 28, 1996.

[8] ISO-ANSI working draft: Database language SQL3, 1994. X3H2/94/080 and SOU/003.

[9] M. Sloman, "Policy Driven Management for Distributed Systems", Journal of Network and System Management, Vol. 2, Part 4, 1994.

[10] J. Widom and S. Ceri, "Active Database Systems: Triggers and Rules for Advanced Database Processing", Morgan kaufmann Publishers, Inc., San Francisco, California, 1997.

[11] Workflow Management Coalition, "The Workflow Reference Model", http://www.aiim.org/wfmc/DOCS/refmodel/rmv1-16.html.

[12] J. D. Ullman, "Principles of Databases and Knowledge-Based Systems", Vol. 1, 2, Computer Science Press, Maryland, 1998.

Appendix

Syntax of Resource Query Language (RQL)

<statement>	::=	<select> <for> with <attribute_value_list>
<select>	::=	select <attribute> from <resource> <where>
<where>	::=	<empty> | where <ranges>
<ranges>	::=	<range> | <range> and <ranges>

<range>	::=	<attribute> <op> <value>
<op>	::=	> | < | =
<for>	::=	for <activity>
<attribute_value_list>	::=	<attribute_value> | <attribute_value> and <attribute_value_list>
<attribute_value>	::=	<attribute> = <value>

Syntax of Policy Language (PL)

<statement>	::=	<quality> | <require>| <substitute>
<qualify>	::=	qualify <resource> <for>
<require>	::=	require <resource> <where*>[6] <for> <with>
<substitute>	::=	substitute <resource> <where> by <resource> <where> <for> <with>
<for>	::=	for <activity>
<where>	::=	<empty> | where <ranges>
<with>	::=	<empty> | with <ranges>
<ranges>	::=	<range> | <range> and <ranges>
<range>	::=	<attribute> <op> <value>
<op>	::=	> | < | =

[6] <where*> is the *where* clause, as defined in [8].