Policy Resolution for Workflow Management Systems

Christoph Bußler[*] *Stefan Jablonski*[†]

Digital Equipment GmbH, AIT
Vincenz-Prießnitz-Str. 1
D-76131 Karlsruhe, Germany

Abstract

In this paper we introduce Policy Resolution (PR) for Workflow Management Systems (WFMS) as service to assign work to agents. Policy Resolution is a framework for defining arbitrary role and organization models together with operations suiting the needs of Workflow Management Systems.

1 Introduction

Cooperative Information Systems (CIS) are characterized by cooperating agents, whereby agents are human or non-human users. They have to fulfill simple tasks, which can be accomplished by one agent, but mostly will have to fulfill complex tasks, which require contributions from multiple agents. If these complex tasks are executed in a more or less predefined manner, they are called *business processes*. Examples are order entry processes or travel expense claims [4]. Business processes describe *what* has to be done at what point in time (*when*).

In agreement with many other approaches to process modeling (e.g. [6]), we want clearly distinguish between the aspects *what* and *when* and the aspect *who*. The latter specifies the agents, responsible and eligible to execute (pieces of) processes. The separation allows to focus on the aspect in the first place and to deal with the integration of the others at a later time. This is a benefit since the attention of a modeler is not distracted from other modeling aspects. But most implementations do not separate between process descriptions (*what* and *when*) and agent assignment specifications (*who*) and only support presumed and static agent assignment strategies like simple role models which is not sufficient as we will see later on. In this paper we present a general approach to agent assignment that supports the definition of arbitrary, problem oriented agent assignment strategies, either in the context of processes or in any other context of CIS (e.g. groupware) which is much more powerful than the role models known. Therefore our approach is a generic approach to respond to the general question of *who has to execute a task* in CIS.

The benefit of a general approach is that it can be applied to a large variety of problems in the task assignment space. It also allows to tailor it to the specific requirements avoiding to "adjust the problem to what the system allows to do". So the main contribution of this paper is to introduce generality to task assignment design problems.

This paper concentrates on the discussion of agent assignment in the context of processes. Since workflow management systems (WFMS) are regarded as decent execution platform for processes, we will discuss the specific problem of *who should execute a certain workflow.*

A workflow comprises either a set of steps, each step can in turn be a workflow again, a so-called *subworkflow* (*what*), and a precedence structure (*when*) between the steps to define their execution order. At runtime a workflow management system determines the steps ready for execution for an initiated workflow. After a step is executed (by a person or more general an agent, which also encompasses machinery or software processes) the workflow system determines the next steps of the workflow which are ready for execution. Since we deal with complex workflows they have to be assigned to multiple agents (e.g. people, mechanical machines, software processes) in order to be executed.

A workflow description traditionally references a role to specify agents eligible to perform a task. In most WFMS a role encompasses a set of agents. When a task is ready for execution it is assigned to each agent identified by the specified role. Real applications of WFMS in enterprises however show that the concept of roles is not sufficient to cope with the overall requirements. Other kinds of task assignments to agents are necessary like the direct assignment of a task to an agent, the assignment of a task to a group of agents (this is independent of the role concept), or the assignment to a person relative to another one (e.g.

[*]Current Address: Technical University of Darmstadt, Department of Computer Science, Institute of System Architecture, Alexanderstr. 10, D-64283 Darmstadt, Germany, E-Mail: `bussler@isa.informatik.th-darmstadt.de`

[†]Current Address: University of Erlangen-Nürnberg, Institute of Database Systems, Martensstr. 3, D-91058 Erlangen, Germany, E-Mail: `jablonski@informatik.uni-erlangen.de`

Table 1: Principle concepts of task assignment

Concept	***Example***	***Action of Policy Resolution***
Agent	*Igor*	Check if agent *Igor* exists. In case it does PR returns ok.
Role	*manager*	Search for all agents able to play the role *manager*. Policy Resolution returns all managers.
Classified Role	*manager(agent)*	Compute the *manager* of a given *agent*. Policy Resolution returns the selected manager.
Conditional Expression	if *amount* > *1000* then *manager(agent)* else *manager(manager(agent))* endif;	Compare if *amount* is greater than *1000*. If this is true the *manager* of a given *agent* is computed, otherwise the *manager* of the *manager* of a given *agent* is determined. Policy Resolution returns the manager or the manager of the manager.

manager of a clerk) or to the same person who already executed a previous task. Examples for principle concepts of task assignment are shown in the first two columns of Table 1.

In order to support different problem specific kinds of task/agent assignments (like the shown above) we propose a framework called ***Policy Resolution (PR)***. PR provides modeling elements to model the organizational aspect (*who*). This comprises an organization structure to formally define the organization's structure and population, to define agent profiles (called *concepts* in Table 1) as a way to select elements from an organization structure as well as policies which define for each workflow the agents profiles to be applied.

As said above, steps are assigned to agents. Before a step is assigned, a WFMS calls a system implementing PR, the so-called ***Policy Resolution Engine (PRE)***, to find out the agents the step should be assigned to. This is done for each workflow or step to be executed. If a WFMS calls a PRE, the PRE finds out the appropriate policies, evaluates the agent profiles contained within these policies and returns the result to the WFMS. The result is a set of agents. The WFMS takes this set and assigns the step to each agent within the set.

Column three of Table 1 shows the work PR is doing for the different types of assignment.

As can be seen later on in more detail (Sections 3.5 and 4), PR is not only good for building a conceptual schema but also has a certain semantics in terms of execution of what is modeled. So it is a framework which allows conceptual modeling as well as provides an execution semantics.

If it turns out that workflows are assigned wrongly or should be assigned differently due to changes in the organization, the appropriate task assignments have to be changed in order to adjust the assignment. So PR is the place to look at if the assignment of steps to agents are not as expected.

The paper is structured as follows. Section 2 introduces a comprehensive example of a workflow. In this context sample task assignment strategies are discussed. Section 3 introduces the ***Policy Resolution Model (PRM)*** formally and models some of the sample tasks assignments of the example in Section 2. Section 4 shows the architectural embedding of PRE into a WFMS. Section 5 lists related work.

2 Travel expense claim workflow

In this section the sample workflow *travel expense claim* is introduced. Figure 1 depicts its graphical representation. The goal of this process is to reimburse a member of a company who traveled and spent money.

Before the task assignment specifications (italic text) are explained, we introduce how to read the workflow specification of the travel expense claim. In Figure 1 two kinds of workflows can be distinguished: *composite* and *elementary* workflows. Composite workflows (Travel Claim `TC`, Preliminary Work `PW`) are composed of further workflows, so-called *subworkflows* (`fill`, `check`, `sign`, `reimburse`, `PW`). Workflows which do not consist of further subworkflows are called *elementary* (`fill`, `check`, `sign`, `reimburse`) and are referred to as *steps*. Elementary workflows are linked to *applications*. In case of the travel claim process it is a spreadsheet application. In general however, every elementary workflow can be linked to a different application. The control flow between workflows (denoted by solid arcs) determines the order of workflow execution that implements the expense voucher. Data are exchanged between workflows indicated by dotted arcs (for details see [7]). In our case it is the file the spreadsheet application works on (`SF`).

Each workflow is associated with a task assignment specification (dashed boxes) to describe the agents which are eligible to execute it. In our example we attach this specification as plain text; it will be formalized later on. Two kinds of task assignment specifications can be distinguished: *local* assignments (`L`) with a scope limited to the workflow they are associated with (e.g. **anyone** for `TC`); *global* assignments (`G`) with a scope of all the subworkflows of the workflow they are associated with (e.g. **`members of company`** for `TC`). These two types are introduced to reduce the design specification code as well as to ease the control of correctness.

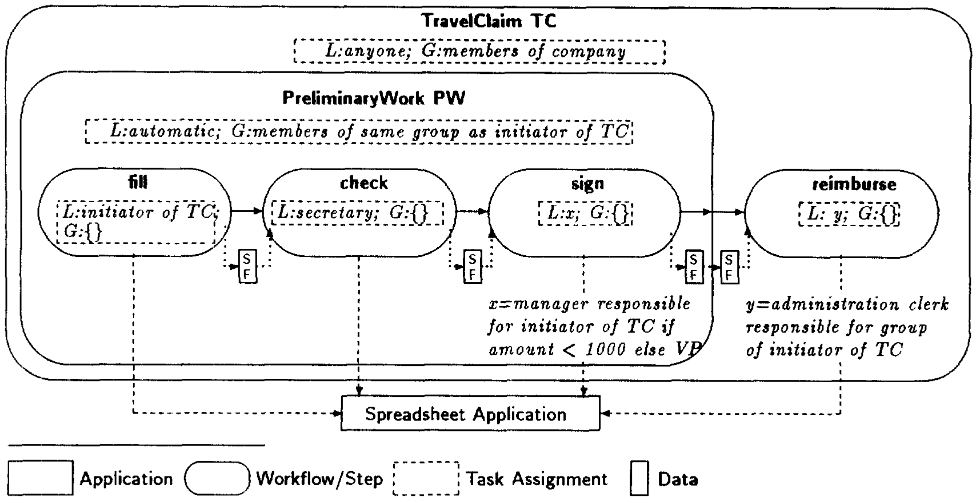
TravelClaim TC
L:anyone; G:members of company
PreliminaryWork PW
L:automatic; G:members of same group as initiator of TC
fill
L:initiator of TC; G:{}
check
L:secretary; G:{}
sign
L:x; G:{}
reimburse
L: y; G:{}
S F
x=manager responsible for initiator of TC if amount < 1000 else VP
y=administration clerk responsible for group of initiator of TC
Spreadsheet Application
Application
Workflow/Step
Task Assignment
Data

Figure 1: Example: Travel Expense Claim Workflow

A travel claim (**TC**) can be started by anyone (**anyone**). Agents working on a travel claim workflow (in subworkflows) have to belong to the company this travel claim is started in (**members of company**). The preliminary work workflow (**PW**) is started automatically (**automatic**). **Automatic** represents a system daemon that can also be regarded as an agent. The subworkflows of **PW** have to be assigned to agents of the same group as the initiator of the whole workflow (**members of same group as initiator of TC**). The **fill** step has to be done by the initiator (**initiator of TC**). The **check** step has to be done by an agent able to play the role **secretary**. The **sign** step has to be performed by a manager responsible for the initiator if the amount claimed is less than 1000 (**manager responsible for initiator of TC if amount < 1000**); otherwise the VP of the group has to be assigned (**else VP**). The reimburse step has to be done by an administration clerk responsible for the group of the initiator of the travel claim (**administration clerk responsible for group of initiator of TC**). Because the latter four workflows are elementary, no global task assignment specification must be provided.

The example shows that further concepts than roles are used to bind agents to workflows:

- roles (e.g. manager, secretary, administration clerk)
- groups (e.g. administration, company)
- history dependencies (e.g. reference to initiator of a process)
- organizational dependencies (e.g. member of company, member of same group, group of initiator)
- data dependencies (e.g. amount of reimbursement)
- competence/responsibility (e.g. responsible for)

The next section introduces the policy resolution model which allows to define these concepts. To illustrate how the model is used to define task assignments some aspects of the example in Figure 1 are modeled.

3 Policy resolution model for workflow management systems

The software life cycle distinguishes an analysis and a design phase. The analysis phase aims at understanding the problem space whereas the design phase aims at describing the implementation of the findings of the analysis phase. To describe an implementation, concepts have to be available, expressed in a specification language. In this paper we deal with the design phase of the organizational aspect of workflow management and provide concepts as well as an language.

This section presents the ***Policy Resolution Model (PRM)*** for WFMS. [2] shows a preliminary form of PR. PR is an independent service that can be used by arbitrary clients. In this paper, PR is explained in the context of workflow management systems to show a typical usage of it.

Together with the model a language is introduced, the ***Policy Resolution Language (PRL)***. Throughout the discussion of the PRM we formalize the task assignment specifications of the travel claim example shown in Figure 1 using PRL. PRL is tailored to the specific needs of PR which are not reflected in other languages. This can be observed looking at the definition of Policies (Section 3.4), where specific keywords are introduced. The language is not defined completely here, since our emphasis is on demonstrating how it is used to fulfill the requirements from the travel claim example of Section 2.

The PRM is introduced in several stages: First, a way of defining arbitrary organization structures is shown to be able to describe objects like roles or groups, dependencies between those as well as competence or responsibility. Next, we introduce agent profiles which are like functions selecting subsets of agents out of an organization database. These are used to select appropriate agents for steps to be executed. Sample role models and complex agent profiles are presented in stage 3 to show how the modeling elements of PR can be deployed. Stage 4 introduces the concepts of policies. Policies are objects describing, which agent selection has to be applied for a particular workflow to find out the appropriate agents. Stage 5 discusses the evaluation process of policy resolution i.e. how all the elements work together. Finally, the dynamics aspect of PR is discussed.

3.1 Organization structure

The example of Section 2 shows that task assignment depends upon the organization structure to certain extend. Data describing who plays a certain role, which group a person belongs to and what are the managers of a person are captured in the organization structure of a company. Since these definitions vary from enterprise to enterprise, PRM allows to model different kinds of organization structures through certain basic model elements.

A general framework like PRM has to enable the definition of arbitrary organization structures. Because we want to use an object oriented model to specify organization structures we have to provide object and relationship types to allow the definition of these structures. The object oriented model is easy to use in context of organization modeling (it naturally maps organization elements and their interrelationships), it is familiar to most designers and handsome if a model becomes complex. Furthermore, it allows to model arbitrary structures. *Object types* describe the *elements* of an organization structure and *relationship types* describe possible *dependencies* between the elements. Examples of object types are *Agent* or *Group*, examples for relationship types are *Manager_of* or *Responsible_for*.

Enterprises using PRM can implement their own organization structure and role model (e.g. [6, 11, 13]) according to their needs without being limited to a presumed set of model elements. This is guaranteed because arbitrary object and relationship types can be defined and connected arbitrarily. Section 3.3 shows how a particular role model as described in [6] can be modeled that way. It is chosen as an example and the modeling elements of PRM described in the following are used to model it. The following two definitions introduce how object types and relationship types are defined. Each object type has a name (**type_name**) and a list of properties (**property-list**). Each property has a name (**property_name**) and is of a certain type (**property_type**).

DEF 1: Policy Resolution Language — Specification of Organization Structure: Object Types

```
OBJECT_TYPE type_name
   PROPERTY property-list

   property-list ::=
      {property_name: property_type;}+
```

As examples we define the types *Manager*, *Secretary*, *Group*[1], and *Agent*. We only declare properties which are relevant for the example in Figure 1.

```
OBJECT_TYPE Manager
   PROPERTY name: String;
            amount_to_sign: Integer;
            /* amount a manager is allowed to
            sign */

OBJECT_TYPE Secretary
   PROPERTY name: String;
            duty: Set(String);
            /* set of duties a secretary has to
            accomplish */

OBJECT_TYPE Group
   PROPERTY name: String;

OBJECT_TYPE Agent
   PROPERTY name: String;
            tel#: Integer;
            /* telephone number of an agent */
            kind: {human, mechanical, program};
```

Each relationship type is characterized by a name (**relationship_type_name**) a list of objects defining which objects can be related (**type_name-list**) and a property list (**property-list**).

DEF 2: Policy Resolution Language — Specification of Organization Structure: Relationship Types

```
RELATIONSHIP_TYPE
   relationship_type_name(from_id, to_id)
   OBJECTS type_name-list
   PROPERTY property-list

   type_name-list ::=
      [object_type_name {from_id | to_id}
      | relationship_type_name {from_id | to_id}]
      {, type_name-list}
```

As examples we define the relationship types *Member_of*, *Plays*, and *Responsible_for*.

[1]In our example we model a group as a set of agents. This does not mean that this is the only possibility. A group could also be modeled as a set of roles.

```
RELATIONSHIP_TYPE Member_of(a, g)
   /* this relationship declares membership of
      an agent to a group */
   OBJECTS Agent a, Group g
   PROPERTY part_time: Boolean;
            /* defines if the membership to a
            group is part time only */

RELATIONSHIP_TYPE Plays(a, ms)
   /* this relationship defines that an agent
      can play the role manager or secretary */
   OBJECTS Agent a, Manager ms, Secretary ms
   PROPERTY {}

RELATIONSHIP_TYPE Responsible_for(a, ag)
   /* this relationship declares an agent respon-
      sible for another agent or for a group */
   OBJECTS Agent a, Agent ag, Group ag
   PROPERTY {}
```

We refer to properties of object types or relationship types by the notations **object_type_name.property_name** or **relationship_type_name.property_name** respectively. The object and relationship types belong to a conceptual schema of an organization structure. This has to be implemented and populated in a system implementing PR (see Section 4) to capture all instances of an real organization in the system.

3.2 Agent profiles

For a workflow it has to be determined, who should execute it. We have to specify which agents out of the organization population are eligible. This is introduced in the following through *agent profiles*. Each workflow is associated (through a policy, see 3.4) with an agent profile. In the example in Figure 1 we attach this profiles as plain text (e.g. **members of company** in TC). Agent profiles should be formalized and specified in a formal language to later on be evaluated by software. Agent profiles define a subset of all available agents through propositions. For instance, the profile **members of same group as initiator of TC** describes all agents which are in the group the initiator belongs to. Agents as well as organization information like *Groups* and relationships between agents are defined and stored in the organization database. Agent profiles are therefore translated into functions selecting agents according to some criteria from an organization database. Often, criteria refer to values of properties (of object or relationship types). For example, human agents (**agent.kind** = **human**) or members of group CAD (**member_of(CAD)**) should be selected.

Agent profiles should be reusable since they probably can be reused in different workflows. This means that agent profiles must be parameterized. The sample profile **members of same group as** could be parameterized with an actual parameter *John* of type *Agent*. This would select members of John's group. In the example of Figure 1 this profile is parameterized by the initiator of the travel claim workflow.

Agent profiles should be independent of objects they are used by (e.g. a workflow). We call these objects the *environment of the use*, shortly *environment*. Therefore, agent profiles are defined independently from the environment. However, sometimes an agent profile depends on values of the environment like in the travel claim example on the amount of reimbursement. This value has to be propagated to the agent profile. How this is achieved is introduced in Section 3.4.

Agent profiles should be described independently from the way they are implemented. This means that we have to provide an abstract language that describes agent profiles and a compiler which translates the statements into executable code of some underlying system. Such an underlying system might be a relational database. In this case an agent profile gets translated into SQL statements. In another case this might be a logic programming system. Then an agent profile is translated into statements of a logic programming language.

DEF 3: Policy Resolution Language — Specification of Agent Profile

```
PROFILE profile_name (interface-description)
   RETURNS variable_name: type_name
   WITH requirement-list

   interface-description ::=
      /*empty*/
      | name: type {, name: type}+

   requirement-list ::=
      <propositions about interface
      variables and return variables>
```

As an example, we will define an agent profile that describes managers that are responsible for an agent:

```
PROFILE Manager_responsible_for (an_agent: Agent)
   RETURNS agent: Agent
   WITH agent:
        agent responsible_for an_agent
        and agent plays manager
        and agent.kind = human;
        /* this specifies all agents such that
        each of them is responsible for an_agent,
        plays the role manager and is human. */
```

This could be implemented using SQL as follows (the definition of tables is omitted for simplicity here):

```
select res_a.id
from agent res_a, plays p, responsible_for rf,
     role ro
```

```
where res_a.kind = "human"
      and p.agent_plays = res_a.id
      and p.role = ro.id
      and ro.name = "manager"
      and rf.responsible_agent = res_a.id
      and rf.responsible_for = an_agent;
      /* an_agent regarded as parameter */
```

In case the organizational database would be an IMS database system an implementation would look very different. The same is true if a logic programming system would be used for implementation. The organization structure would be facts and the agent profile statements of a logic programming language.

3.3 Examples: role models and complex agent profiles

The insufficiency of the role model to deal with more complex situations is already shown in Sections 1 and 2. The next subsections show that agent profiles encompass the role concepts found in approaches throughout the literature but also can express very complex agent specifications that are much more powerful than roles and are absolutely required.

3.3.1 Sample role model; Usually a role model definition is threefold:

- it defines an object type *Role*
- it defines objects types available as *role players*, e.g. users
- it defines *role resolution*, i.e. the way to find objects able to play a certain role (role players).

There are several role models defined in the literature [2, 6, 11, 13]. To show the power of agent profile specifications one role model which is developed in context of process modeling is defined here.

Curtis [6] defines the type role as "a coherent set of process elements to be assigned to an agent as a unit of functional responsibility". Therefore the role object type looks like

```
OBJECT_TYPE Role
   PROPERTY name: String;
            process_elements: Set(Process);
```

Role players are called *agents* in [6]. Curtis assumes that agents are *human* or *non-human (machine)*. However, no other role player types like *Group* are possible. Therefore the object type agent looks like

```
OBJECT_TYPE Agent
   PROPERTY name: String;
            kind: {human, machine};
```

Role resolution is described in [6] as "a single agent can perform multiple roles and a single role may be performed by multiple agents". In order to describe a profile for role resolution we have to define the relationship *Plays* which associates roles with agents as role players.

```
RELATIONSHIP_TYPE Plays(a, r)
   /* this relationship defines that an agent
      can play a role */
   OBJECTS Agent a, Role r
   PROPERTY {}
```

Next, we have to define an agent profile which implements role resolution. This profile is called *Role_players*.

```
PROFILE Role_players (a_role: Role)
   RETURNS agent: Agent
   WITH agent:
        agent plays a_role;
        /* this specifies all agents that play
        the named role */
```

So the definition of the sample role model is complete. As shown the agent profile concept enables to define that role model. If these elements are all we had to model the example of Section 2 we had to give up. E.g. the requirement to model `manager of applicant` or the conditional statement of step `sign` could not be expressed using the elements proposed by Curtis. Also the role models in [11, 13] can be defined with agent profiles. This is however not demonstrated here.

3.3.2 Complex agent profiles; To show how complex agent profiles (which we need for our example) are modeled we want to model the profiles *Members_of_same_group_as()* and *Administration_clerk_responsible_for()* from Figure 1. As we will see later, both specifications reuse the profile *Group_of()*. Since the object types *Agent*, *Group* and the relationship type *Member_of* and *Responsible_for* are already defined we only have to define the object type *Administration_clerk* here.

```
OBJECT_TYPE Administration_clerk
   PROPERTY name: String;
            duty: Set(String);
            /* set of duties of admin. clerk */
```

We have to define the profile *Group_of()*:

```
PROFILE Group_of (an_agent: Agent)
   RETURNS group: Group
   WITH group:
        an_agent member_of group;
   /* this specifies all groups an agent is
   member of. */
```

The two complex profiles can be defined now:

```
PROFILE Members_of_same_group_as(an_agent: Agent)
   RETURNS agent: Agent
   WITH agent:
        agent member_of Group_of(an_agent);
   /* this specifies agents which are member
   of the same groups than an_agent */
```

```
PROFILE Administration_clerk_respon-
      sible_for_group_of (an_agent: Agent)
   RETURNS agent: Agent
   WITH agent:
      agent responsible_for Group_of(an_agent)
      and agent plays administration_clerk;
   /* this specifies all agents which are respon-
   sible for the groups an_agent is member of */
```

The examples show that agent profiles support reusability. For instance, the profile *Group_of()* could be reused in both complex specifications.

3.4 Policies

Having on the one hand side workflow specifications and on the other hand side agent profiles, a way has to be provided to relate these in order to know, which profile has to be evaluated for a given workflow. Therefore the concept of *policy* is introduced to bind a version of an agent profile to an object of the environment it is used, in our example to workflows. For instance, the agent profile *Administration_clerk_responsible_for()* is bound to the `reimburse` workflow through a policy. Since a policy is an object by itself, a new version of it can be introduced without changing a workflow or a profile, if the situation requires it (the same applies for workflows and profiles also, allowing to ease change management). Furthermore, a policy enables to reference data values of the environment in agent specifications.

A policy also enables to define additional *integrity rules* that restrict further the selection of agents. For example, the agent for the `fill` step (Figure 1) has to be the same as the agent who started the travel claim workflow. Another rule which is not applied in the example is the rule *different_from*: agents assigned to different workflows have to be different (separation of duty).

A policy has a name (**policy_name**). The object of the environment (here: workflow) for which the policy applies to has to be declared (**ENVIRONMENT**). Besides, it must be specified if the policy applies to the workflow itself or to its subworkflows (**SCOPE**). A set of *assignment rules* defines which profile an agent has to fulfill for this workflow; more than one rule is possible, to allow conditional task assignment (**ASSIGNMENT**). *Integrity expressions* are specified (**INTEGRITY**). Since policies (as well as agent profiles) are independent of WFMS the terms *workflow*, *history* and so on do not appear in a policy specification. A workflow or workflow history, more precisely variables or parameters belonging to them, are only referenced in the **ASSIGNMENT** clause, to tailor the policy for a specific use.

If values of the environment are needed, methods of the environment objects are called. In the example in Figure 1 the method *TC.agent()* returns the agent who initiated *TC* (*TC* describes the environment object; *agent()* denotes the corresponding method). *sign.amount()* is another example.

DEF 4: Policy Resolution Language — Specification of Policy

```
POLICY policy_name
   ENVIRONMENT object_name /* workflow names */
   SCOPE scope-definition
   ASSIGNMENT rules
   INTEGRITY expression

   scope-definition ::= {LOCAL | GLOBAL}

   rules ::= {if condition : profile ;}+

   expression ::=
      same_as object_name
      | different_from object_name
```

As an example the policy for the *sign* step is modeled:

```
POLICY signing_of_travel_claim
   ENVIRONMENT TC.PW.sign
      /* this policies applies for the sign
      workflow within PW within TC. */
   SCOPE LOCAL
   ASSIGNMENT
      if sign.amount() < 1000:
         manager_responsible_for(TC.agent());
      if sign.amount() >= 1000: VP();
   INTEGRITY {}
```

In Section 2, we introduced the concept of scoping. Local and global scopes are distinguished. In the example of Figure 1, three task assignments have to be considered for the workflow **sign**: the global policy defined in **TC** saying that only members of the company get the subworkflows of **TC** assigned, the global policy defined in **PW** saying that members of the same group as the initiator of the travel claim get the subworkflows of **PW** assigned, and the local policy attached to the **sign** step itself. The scope of policies is declared within the **SCOPE** section of a policy definition. Figure 2 represents graphically the scope of the policies defined within the travel claim expense workflow. The rectangular areas denote the scope of the policies: **workflow_name-L** denotes the locally scoped policy of the workflow **workflow_name**, **workflow_name-G** denotes the globally scoped policy of the workflow.

3.5 The evaluation process of policy resolution

When policies are resolved for a workflow, the nested scoping structures of policies have to be considered. Therefore ***Policy Resolution (PR)*** works as follows. The policy for a workflow currently worked on is evaluated first. If there are global policies applicable from enclosing workflows, they are evaluated starting from

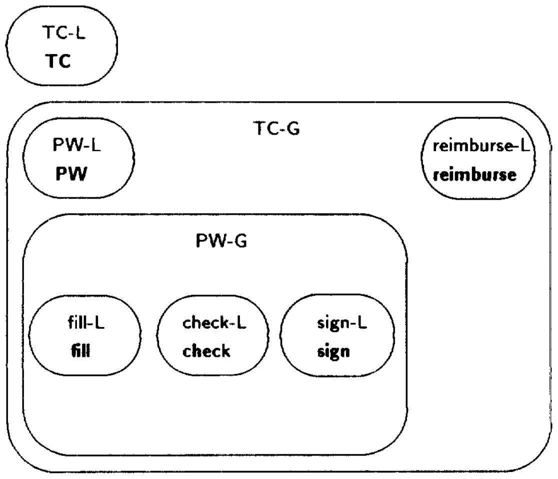

Figure 2: Scope of the Policies for the Travel Claim Expense Workflow

the next outer to the outmost workflow (ancestors). Optimization is possible by evaluating policies applying for subworkflows only once and storing the results for the lifetime of the respective workflow. This avoids re-evaluation whenever a subworkflow is assigned.

In the example of the subworkflow **sign** the policy attached to the **sign** step itself is evaluated firstly. Given the case that the amount is larger than 1000, all VPs of the company are eligible. Secondly, the global policy applying for the subworkflows of PW is evaluated. This reduces the eligible VPs to the VPs of the group the initiator belongs to. Thirdly, the global policy for the subworkflows of TC is evaluated. This does not influence the result of PR if we assume that the organization structure contains only members of the company and no external agents.

The following algorithm shows how policies for a workflow `wf` are evaluated. It is the optimized version since it stores the results of the evaluation of global policies. The stored results of a workflow are deleted as soon as its topmost ancestor workflow is finished.

```
resolve_policy(IN wf:  Workflow,
               OUT res:  set_of(Agent))
  begin
    p_l := get_local_policies(wf);
    res := evaluate(wf,p_l);
    for each ancestor a of wf do:
      res := res ∩ get_global_policy_result(a);
    end_for;
    if not elementary(wf)
    then  p_g := get_global_policies(wf);
          g := evaluate(wf,p_g);
          store_global_policy_result(wf,g);
    end_if;
  end;
```

Since more than one policy can be specified for one workflow they can potentially conflict (***policy conflict***). This is the case when for at least two policies the environment specification as well as the scopes overlap. The easiest resolution is to compute the intersection of all policies applying for one workflow. This is a default ***policy conflict resolution rule***. More sophisticated ones are possible, but not discussed here.

3.6 The dynamic aspect of policy resolution

Organizations are dynamic systems. This dynamics of organizations has to be considered in policy resolution. In the following several aspects are discussed briefly:

- *Changes of Organization Structure.* The organization changes from time to time because of its reorganization or people leaving or joining it. Whenever a change takes place, the representation of the organization, which is the organization structure and its population has to be updated accordingly to reflect reality as close as possible. A system implementing PR has to provide therefore a management interface which enables a user to adjust the organization structure and/or its population.
- *Status of Agents.* In general, there might be several agents eligible for executing a step of a workflow. However, choosing different agents might have an influence on e.g. the execution time. One agent might have a high workload and it therefore takes time until the agent starts working on it. Another agent might be idle and could start working immediately. Whenever a certain property of agents might influence their selection (e.g. to speed up the workflow execution), this property has to be used appropriately in an agent profile (e.g. *workload* $\leq$ *50*).
- *Evaluation Time of Policies.* In general, all the policies of a workflow could be evaluated at the time of their instantiation. However, there are at least three reasons why this might not be a good approach. First of all, not all the data upon which evaluation is based might be available (like history data). If a policy relies on the information which agent executed a certain step in a workflow, this step has to be executed first before the policy can be evaluated at all. Second, changes in the organization structure after the workflow is started are not taken into consideration following this approach. Third, changes of policies or agent profiles after the start of a workflow are not considered in this approach either. To take the latest changes into consideration, policies are evaluated at the latest possible time, i.e. short before a workflow is assigned.
- *Exceptions.* A system can behave *intended* or *not intended.* If it behaves intended, everything is ok. The not intended case (e.g. through a wrong specification) can be subdivided into *expected* and *not expected.* The not expected case (e.g. system break)

is not considered here. However the not intended, but expected case can be covered through policies allowing a controlled exception handling. Imagine, that no agent fits a profile. In this case a workflow can be assigned to some supervisor. This can be achieved by applying a if–then statement in a policy (*if no agent fits then supervisor*). Other expected problems can be handled analogously.

- *Agent Synchronization.* In general policies return for a given workflow a set of agents. All these agents are eligible to execute the workflow (otherwise the policy would have been specified differently). However, in general not all of them should execute the workflow but only a subset of them, in many cases only one. A role can serve as example here. In case a workflow is assigned to a role only one role player should execute it, not all of them. To synchronize the access, *agent synchronization rules* have been introduced (see [3]) as a general way to describe, how many agents out of a set are allowed to access a commonly assigned workflow. Policies are extended with these agent synchronization rules to not only specify to *whom* a workflow is assigned but also how the agents are synchronized.
- *Synchronous Work.* WFM belongs to the class of asynchronous agent coordination. This is because agents work in some sequence determined by a WFMS and not (as in synchronous coordination systems like conference systems or multi user editors) all at the same time. Despite the asynchronous character of WFM, synchronous applications can be embedded as applications (see Section 2). In this case PR determines the initial set of agents which participate in e.g. a conference session (or the initial roles if the systems requires it) and provides it at the time the application is called. If the conference system allows to add or remove agents from the session, this can be done independently of PR of course, since PR is not concerned about matters within applications but only within workflows.

4 Embedding of policy resolution in workflow management

Having explained how the evaluation process of policy resolution works itself, its embedding in a WFMS is shown in the following.

Before going into the details of the embedding, our proposed general architecture of WFMS is introduced in terms of a collection of software servers ([4]). In general, for each functionality required we introduce a software server . E.g. there is one to deal with data (*Data Server*), one to deal with the history (*History Server*) and so on. The one of our concern here is the ***Policy Resolution Engine (PRE)***. PRE contains the organization structure as well as its population, all agent profiles and all policies. In the center of all the kernel orchestrates the software services, i.e. it calls them whenever the provided functionality is required. Figure 3 is a graphical representation of the proposed architecture.

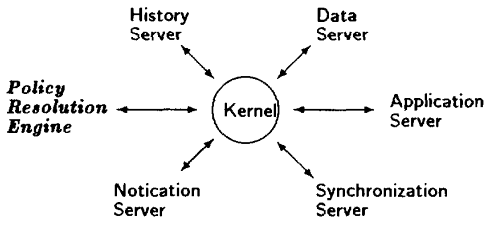

Figure 3: Components of a WFMS

The next algorithm shows, how a kernel makes use of PRE how its results influences the processing of workflows. We concentrated on the parts using PRE.

```
while kernel not stopped do
   if request to start workflow of type t
   then add_to_running_wf(create(t));
   end_if;
   wf:= select_from_running_wf();
   if some subworkflow finished in wf
      or newly_created(wf)
   then s := subworkflows of wf to be started
             next;
        for each swf in s do
            resolve_policy(swf, res);
            for each agent a in res do
                assign(swf, a);
            end_for;
        end_for;
   end_if;
end_while;
```

From the algorithm above it can be seen that PRM is called shortly before the workflow is to be assigned to agents. It is assigned to those agents PRE returns as a result. This ensures that the latest update of the information PRE makes use of is taken (e.g. organization structure or policies).

5 Related work

Exemplary we want to discuss the related approach developed in the office system *Domino* (see [9, 10]). The Domino system is one of very few approaches which factored out the organizational aspect to deal with it separately.

The Domino system allows the specification of an organization structure. Provided are four fixed object types with fixed attributes: *organizational_unit*, *projects*, *jobs* and *employees*. The Domino system does not allow to add or modify the predefined types. There is a set of predefined relationship types like *parent_unit* or *supervisor*. These are, however, not explicit relationships but hidden in attributes of the types (implicit relationships). Other relationships are derived only like the *member_of* relationship. Access to history is not generally possible but only the initiator of an office procedure can be referenced using a predefined keyword *initiator*. Since in Domino the objects and relationships are limited to describe the organizational aspect the language to bind agents to office procedures is limited also. Generally spoken, the approach within the Domino system is very initial and therefore limited.

We also want to relate to some work done much earlier than the Domino system. An early formal model to specify office procedures can be found in [5], called *Office Procedure Model (OPM)*. This model is mainly concerned about data and information flow, based on an APN (augmented Petri-nets), an ICN (information control net) and a form flow model. It describes relationships among messages (forms), databases, alerters and activities based on the idea that an activity is triggert by an alert causing database operations and message passing. OPM is intended to represent office procedures and to coordinate office activities. Chang et. al however go beyond describing modeling concepts and provide an execution mechanism for modeled office procedures with OPM. The execution mechanism is based on three components, a AMS (activity management system), an DBMS (database management system) and a XMS (message exchange system). Interesting in the context of PR is a component of their system, called IC (intelligent coupler). This module of the system interfaces with the outside world e.g. user interaction. So Chang et. al realized that user interaction is necessary in an office environment. However, in OPM there is no modeling concept which allows to model users at all. Their model does not even talk about users. So the execution mechanism provides the notion of user interaction despite the fact that the model does not know about this. Consequently there is no notion of task or activity assignment either.

More recent related work (see [1, 8, 12, 13]) is not discussed here due to space limitations.

6 Summary

In this paper we have introduced Policy Resolution. PRM provides a user-friendly framework for specifying and executing policies as well as agent selections and organization structures. Using an elaborated example we explained how policy resolution can be used to model arbitrary role models and how it can be applied to WFMS. Finally we gave a brief overview on the implementation of the PRE prototype.

A first prototype is implemented. The organization structure as well as the agent profiles and policies are stored in a relational database. The policy resolution algorithm is implemented in C++ on top of the relational database. This algorithm allows a client to evaluate policies for a given object of the environment.

References

[1] Agha, G.: *The Structure and Semantics of Actor Languages.* In: Foundations of Object-Oriented Languages, Springer-Verlag, LNCS 489, 1990.

[2] Bussler, C.: *Capability Based Modeling.* In: Proceedings of the First International Conference on Enterprise Integration Modeling Technology (ICEIMT), Hilton Head, SC, USA, June 1992.

[3] Bussler, C., Jablonski, S.: *Implementing Agent Coordination for Workflow Management Systems using Active Database Systems.* In: Proceedings of the 4th International Workshop on Research Issues in Data Engineering: Active Database Systems (RIDE—ADS'94), Houston, Texas, USA, February 1994.

[4] Bussler, C., Jablonski, S.: *An Approach to Integrate Workflow Modeling and Organization Modeling in an Enterprise.* In: Proceedings of the Third IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE), Morgantown, West Virginia, USA, April 1994.

[5] Chang, S. K., Chan, W. L.: *Transformation and Verification of Office Produres.* In: IEEE Transactions of Software Engineering, Vol. SE-11, No. 8, Aug. 1985.

[6] Curtis, B., Kellner, M., Over, J.: *Process Modeling.* In: Communications of the ACM, September 1992, Volume 35, Number 9.

[7] Jablonski S.: *Data Flow Management in Distributed CIM Systems.* In: Proceedings of the 3rd International Conference on Data and Knowledge Systems for Manufacturing and Engineering, Lyon, France, March 17-20, 1992.

[8] Karbe, B., Ramsperger, N.: *Concepts and Implementation of Migrating Office Processes.* In: Informatik Fachberichte 291, Springer-Verlag, Berlin, 1991.

[9] Kreifelts, T.: *Coordination of Distributed Work: From Office Procedures to Customizable Activities.* In: Informatik Fachberichte 291, Springer-Verlag, Berlin, 1991.

[10] Kreifelts, T., Seuffert, P.: *Addressing in an Office Procedure System.* In: R. Speth (Hrsg.) Message Handling Systems, State of the Art and Future Directions, Proc. IFIP WG 6.5 Work Conference on Message Handling Systems, North-Holland, Amsterdam, 1988.

[11] Lawrence, L.: *The Role of Roles.* In: Computers and Security, Volume 12, Number 1, 1993.

[12] Medina-Mora, R., Winograd, T., Flores, R., Flores, F.: *The Action Workflow Approach to Workflow Management Technology.* In: Proceedings of the ACM 1992 Conference on Computer-Supported Cooperative Work, Toronto, Canada, 1992.

[13] Singh, B., Rein, G.: *Role Interaction Nets (RIN): A Process Description Formalism.* MCC Technical Report Number CT-083-92, MCC, Austin, Texas, USA.