Enterprise Workflow Resource Management

Weimin Du, Jim Davis, Yan-nong Huang, Ming-Chien Shan
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-1999-8
January, 1999

workflow, resource management, resource model

Resource management is a key issue in providing resource independence and efficient use of workflow resources. The paper has outlined a design of such a resource management system for enterprise workflow environments. It is capable of handling a large number of workflow resources that are independently managed by pre-existing resource systems. It integrates these external resource systems at schema level without duplicating individual resource information. Resource specification is greatly simplified by providing process designers with integrated views of enterprise workflow resources at different levels based on a unified resource model. Dynamic behaviors of workflow resources are supported through powerful resource policies.

Internal Accession Date Only

Enterprise Workflow Resource Management

Weimin Du, Jim Davis, Yan-nong Huang and Ming-Chien Shan

Hewlett-Packard Laboratories

1501 Page Mill Rd

Palo Alto, CA 94304

1 Introduction

Workflow is a technology that provides the ability to define and automate the flow of work through an organization to accomplish business tasks. A workflow process involves the coordinated execution of tasks performed by workflow resources (e.g., a person, a computer-based application, or a piece of equipment). One of the important features of modern workflow technology is the dynamic resource allocation, which provides resource independence to business processes. Thus, a business process does not need to be modified when underlying workflow resources change. It also allows more efficient utilization of available resources.

A resource manager is a component of a workflow system that allows run time resource allocation. A resource manager provides a resource model to process designers for resource specification at process definition time. The model provides an abstraction of the physical resources and shields the process designers from the detailed specification of the resource required. A resource manager also manages workflow resources (e.g., keeps track of status of workflow resources) and assigns workflow resources to business steps (or workflow activities) when requested by the workflow execution engine. Resource management is an important and complicated task, especially in enterprise workflow environments.

An enterprise workflow system (such as HP Chang*e*ngine [4]) is a workflow system that is capable of supporting large number business critical processes in an efficient, reliable and secure way. Resource management in an enterprise workflow system has the following characteristics.

- The number of workflow resources can be very large. For example, the employee expense reimbursement processes in HP involve (>100,000) HP employees as workflow resources.
- Workflow resources at different organizations and locations are often managed by different systems independently. These external resource systems can be heterogeneous with respect to resource models, query languages and communication protocols.
- Process designers need different views of workflow resources at different levels. Most business processes only involve local resources. There are also cases where an enterprise-wide view of all workflow resources is needed.

- A company or an organization may need to enforce certain rules regarding resource usage. For example, the 2nd line manager approval is required for all expenses over $500. The rules may change from time to time according to business conditions.

Resource managers of most existing workflow products do not meet the above requirements. For example, the resource management component of IBM FlowMark [5] requires explicit registration of all workflow resources. This not only makes it practically impossible to handle large number of resources, but also causes potential inconsistency between the external resource systems and the internal workflow resource system.

This paper describes the design of a Resource Manager for enterprise workflow management systems. The described resource manager allows integration of existing resource sources and external resource monitoring components. The resource manager also includes a policy engine (and an accompanying policy definition language) to allow flexible resource management [3]. The paper describes the architecture that facilitates integration of external resources. It also describes a unified resource model and a resource definition/query language to allow easy resource specification at process definition time.

2 Architecture

2.1 Overall Hierarchy

Workflow resources at different organizations and locations are often managed by different systems independently (e.g., a database or a corporate directory). These systems were built by different organizations for different purposes and used different resource models and technologies. It is thus useful to present unified and integrated views of all workflow resource to process designers for processes involving multiple external resource systems. We distinguish between Local Resource Managers (or LRMs) that pre-exist and have their own resource models and communication protocols and Global Resource Managers (GRMs) that represent integrated views of part or all of the underlying LRMs. GRMs have the same resource model and communication protocol.

Since enterprise workflow resources can be widely distributed across organizational and physical boundaries, resource management is distributed. To support different views of enterprise workflow resources, GRMs are further subdivided into Enterprise GRMs (or ERMs) and Site GRMs (or SRMs). ERMs represent the enterprise-wide view of workflow resources and interface with underlying SRMs, which represent partial views of workflow resources within an organization or a physical boundary. There can be more than one level of SRMs representing different levels of views, forming a tree hierarchy with ERMs as roots. There can also be more than one ERM, all representing the same view of the enterprise workflow resource, to provide fault tolerance. SRMs at the same level represent views in different organization or physical boundaries, and are also the integrated views of their subordinate SRMs. The lowest level SRMs represent imported and possibly integrated views of one or more external LRMs.

A three level hierarchy of resource managers is outlined in Figure 1.

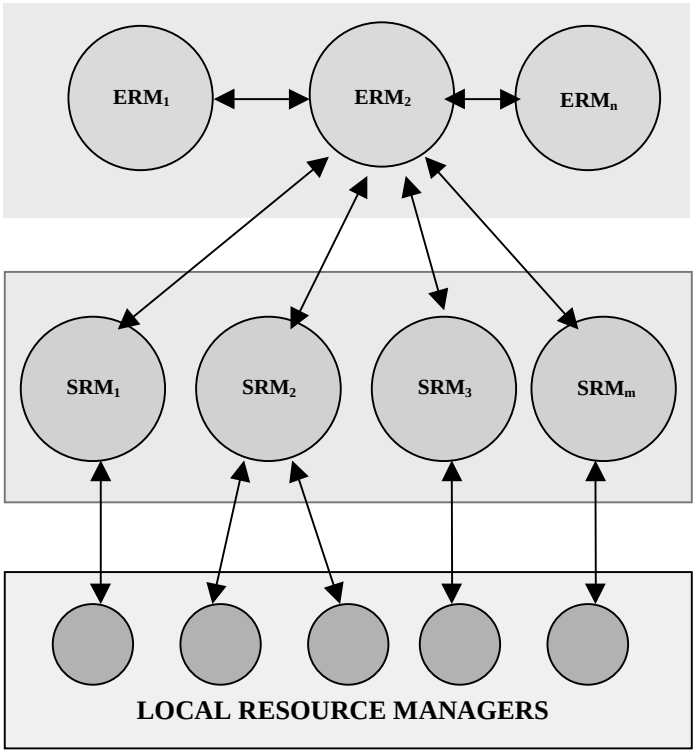

Figure 1: Hierarchy of Resource Managers

2.2 System Architecture

All the SRMs and the ERMs have the same following architecture, which is shown in Figure 2. The GRM has four different layers: the interface layer, the policy manager and resource model layer, the request processing engine layer, and finally the integration layer.

2.2.1 Interface Layer

This layer allows other components like the workflow engine to send requests to the resource manager. The requests are written in RQL (Resource Query Language)[1]. This layer allows tools speaking RPL (Resource Policy Language) and RDL (Resource Design Language) to manipulate the policies and the underlying resource model. The layer is also used for communication between other GRMs. Finally, this layer defines the administrative APIs and uses the underlying security mechanisms.

2.2.2 Policy Manager and Resource Model

This layer implements the policy rules and the resource model (that provides the enterprise view of resources). This layer also provides a database (RM Catalog) with an

[1] See Section 4 for brief descriptions on RQL, RPL and RDL.

extensible schema that is used to store model and historical information and may be used to store other information needed for resource management (i.e. load information for load balancing, etc.).

2.2.3 Request Processing Engine

The Request Processing Engine takes the actual request (after it has been processed by the policy engine) and routes them to the appropriate information source. It also assembles all the results that are returned by the information sources.

2.2.4 Integration Layer

The integration layer manages all the different protocols spoken by local information sources (i.e., LRMs). It allows for the LRMs to be advertised. It handles the request and any result translations required. It also manages the wrappers that need to go around each LRM.

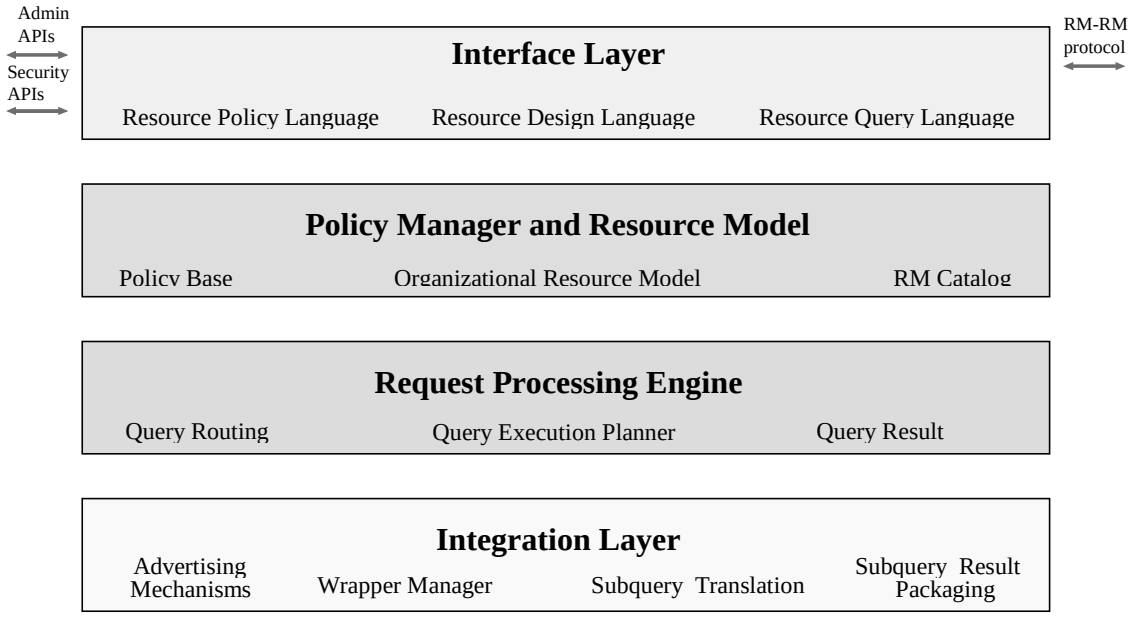

Figure 2: GRM Architecture

2.3 Component Interactions

The interaction diagram for the GRM is shown below in Figure 3. The languages are described in Sections 4. There are three main components to the GRM. The first is the resource manager language interface. It contains the parsers for the resource manager languages as well as a control engine, which controls the process of resolving a resource request. The other two components are the policy engine and the resource engine.

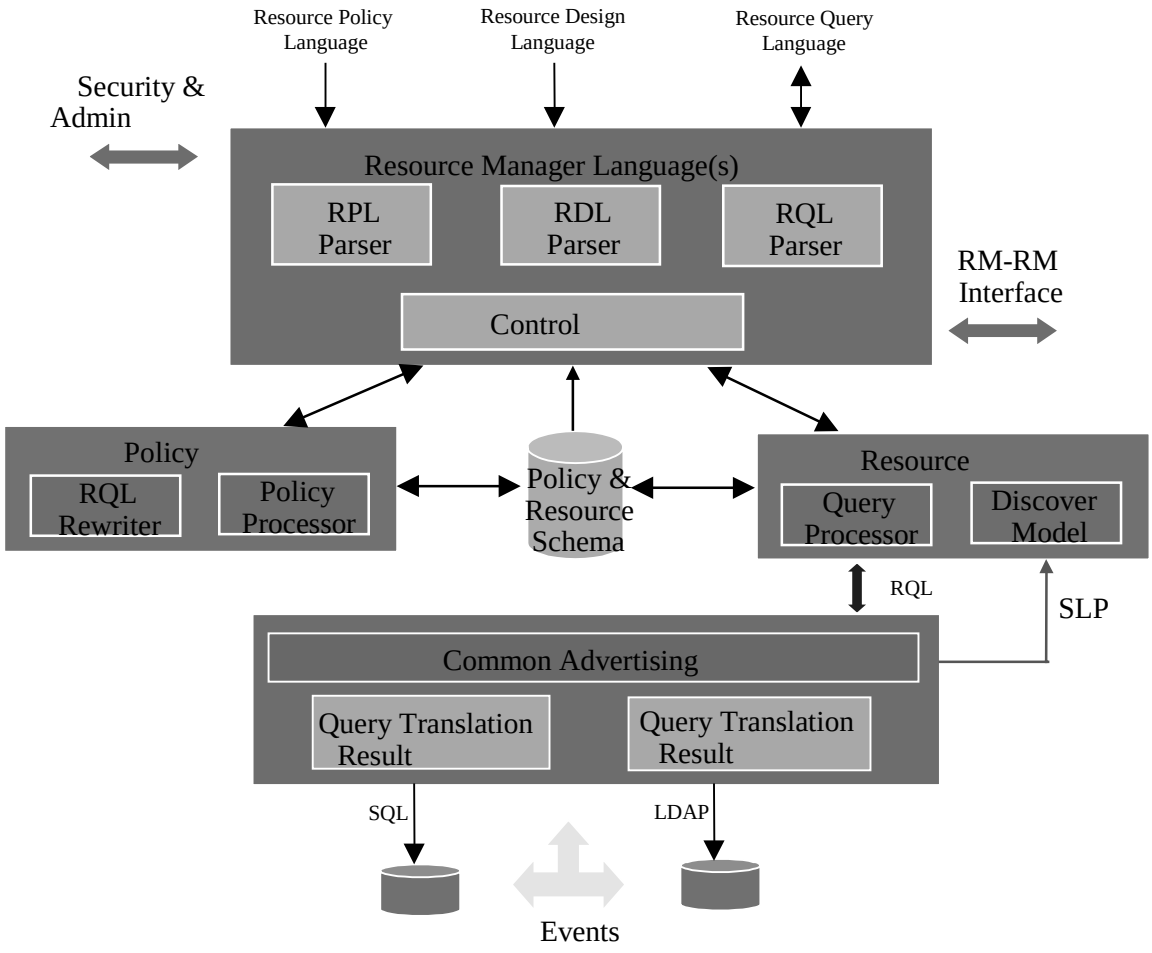

Figure 3: Interaction Diagram

When a resource request comes in to the resource manager, it is first parsed by one of the three language parsers. A resource request is forwarded to the control engine, which performs the following five-step process:

- If the request cannot be handled by this particular GRM (or it is not authoritative), then the control engine uses the RM-RM protocol (described later) to pass the request on to another resource manager.
- If request can be handled then the control engine passes the request to the policy engine component for query rewrite[2].
- After the policy enforcement, the request is forwarded to the resource engine. If a particular resource is found to satisfy the request, the control engine component returns the result as appropriate.

[2] Readers are referred to [3] for detail of query rewrite by policy manager.

- If the resource engine returns NULL, then the request is sent to the policy engine, where substitution policies are applied.
- The request is again sent to the resource engine. If a resource is found, then the result is returned. If the request is still not satisfied and the resource manager has authority over the resource type, then a NULL is returned.

A resource engine has a resource model associated with it, which contains a hierarchical collection (based on capabilities) of concepts representing resource types. A resource model defines static behaviors of resource types (e.g., things they can do) and the relationships among them. Dynamic behaviors and relationships (e.g., a resource is only allowed to do a task under certain changing condition) are specified using policies. For each type of resources, the resource model maintains knowledge of "where" to get instances of that type.

The discovery model component "discovers" local resource managers and the types of resources that they can handle. The query processor in the resource engine makes use of the discovery model to generate sub-queries (possibly multiple) for the LRMs. These sub-queries are then dispatched to the LRMs where the wrappers convert the request (in RQL) into something that is understood by the LRM. A client can talk directly to the resource engine to modify the resource model using RDL. The resource schema is stored in a database, which is shared with the policy engine.

The policy engine takes queries from the control engine and rewrites the request based on applicable policies. The policy engine also manages the collection of policies, which are stored in a database shared with the resource engine. Policies can be added or updated by using the Resource Policy Language.

3 Resource Model

The resource model is a hierarchical collection of resource types. A resource type is intended to denote a set of resource instances with the same capabilities. The resource hierarchy shows resources organized into types. Figure 4 shows a possible resource hierarchy. Each of the types in the hierarchy has a list of capability attributes, which represent its capabilities. Furthermore, a resource type inherits these capabilities (attributes) from its parents. For example, in Figure 4, a *Programmer* inherits all of the capabilities of the *Engineer*. In fact, a *Programmer* is an *Engineer* with some special capabilities.

Each type (for example *Engineer*) in the model also has the following four fields:

1. DO_ADDR (Address): If the resource manager can satisfy requests for Engineer then this variable contains addresse(s) of the LRM(s) that can handle the request. If this list is empty then the resource manager uses the following variables and the RM-RM protocol to send the request to another resource manager.
2. DELEGATE_ADDR (Address): A resource manager that can satisfy Engineer requests and is lower in the RM hierarchy (thus the request can be *Delegated* to them).

3. REFER_ADDR (Address): A cache representing a resource manager that has been discovered (using information returned with the *Report* message) that can satisfy Engineer requests. These are GRMs that are located horizontally in the RM hierarchy (requests can be *Referred* to them). Note: there is no guarantee that this cache is consistent. Consistency is achieved when the resource manager uses this cache.
4. PLEAD_ADDR (Address): A resource manager that can satisfy Engineer requests and is higher in the RM hierarchy (the requests can be Plead to them).

Under the simplifying assumption of three levels of resource managers, we can implement the above as follows:

- SRMs cannot Delegate, therefore, the DELEGATE_ADDR can be left out.
- SRMs can only Plead up to an ERM. So the PLEAD_ADDR can be maintained for an SRM. It need not be maintained for each individual type in the model.
- ERMs cannot Plead and thus the PLEAD_ADDR can be left out.
- ERMs cannot Refer, so there is no need for a REFER_ADDR for the ERMs.

The hierarchy of resources is built using the capability attributes. These attributes represent capabilities or states of resource types that are inherited lower in the hierarchy. A resource type may also contain attributes that are applicable only to it. Resources lower in the hierarchy do not inherit these non-capability attributes.

The resource types can be created using a subset of UML. Using UML as a modeling language allows us to make use of existing UML tools in the market today. The UML representation can then be turned into code (again using existing tools) for better performance. The codified resource type can be compiled directly with the codified protocol above to get optimal performance.

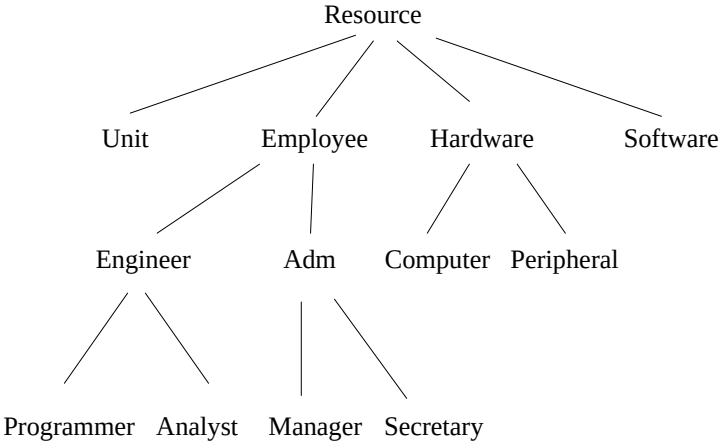

Figure 4: Resource Hierarchy

To allow flexible resource specification in process definition, the GRM also contains knowledge of roles as well as resource types. Roles are logical representations of resource requirement for workflow activities in terms of capabilities. Roles are used by activity definers (when creating new activities) to map activities into resources. Roles are

a boolean expression specifying the resource types needed for the activity. Given this information, the resource manager automatically generates virtual nodes (shaded nodes in Figure 5 below). For example, assume that the activity definer defines two roles *R1* and *R2* for activities A_1 and A_2 as follows:

- A_1: {Role: R_1 = {*Peripheral* and *Software*}}
- A_2: {Role: R_2 = (*Programmer* and *Analyst*) and (*Computer* or *Secretary*)}}

Figure 5 shows how the above roles would be incorporated into the resource model. R_1 can be modeled very simply as inheriting from both *Peripheral* and *Software*. R_2 requires the definition of two "virtual" resources R_2a and R_2b. R_2a represents a set of resources that are either *Computers* or *Secretaries*. R_2b represents resources that are both *Programmers* and *Analysts*. Finally, R_2 represents an "and-ing" of R_2a and R_2b. This method allows for complex boolean expressions of resource types to be expressed in the resource model. The virtual nodes can themselves have additional attributes that can be used during the search process.

The connections between virtual nodes and resource types are labelled to enable the late and early binding of resources. For example, in the example of Role R_1, the link between *R1* and *Peripheral* might be labelled for late binding. When a request for Role R_1 comes to the resource manager, the request would get rewritten by the appropriate policies. The request would not be sent to the LRMs, though.

The virtual nodes (roles) also consist of rules that are used to determine the relationships between the resource types that they represent. Consider a virtual node called *SecureContact* which represents a *Manager* resource type and a secure WorkListHandler. In other words, if one needed to send secure work to a *Manager*, they would use the *SecureContact* role. But the resource manager would need rules to determine how to get (and in which order) the WorkListHandler etc. These rules are only stored in the virtual nodes and thus are only associated with Roles (not resource types).

Role definitions would also need to be merged into the resource model. This should be possible, again using UML. Once role knowledge is represented in the model, one can query GRMs based solely on roles.

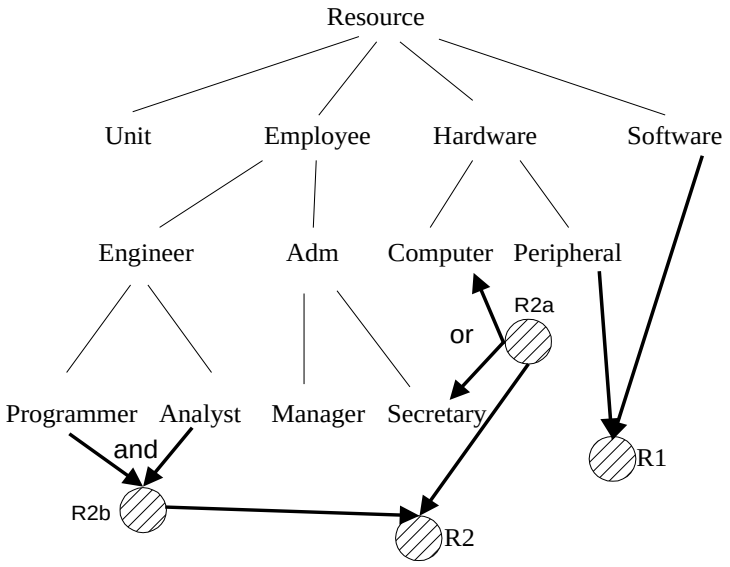

Figure 5: Resource Hierarchy Extended with Roles

4 Design

4.1 Languages and Interfaces

The following simple languages have been defined for resource definition, query and manipulation:

1. **Resource query language (RQL)** – RQL is an SQL like language. Users can use the language to submit resource requests to the resource manager. The language is composed of SQL **select** statements augmented with optional activity specifications (**for** clauses).

select	R_2b
from	SRM_n
where	*Location* = 'PA'
for	*Programming*
with	*NumberOfLines* = 35000 And *Location* = 'Mexico';

Figure 6: Initial RQL Request

The query in Figure 6 requests resources of role R_2b from the SRM_n with additional condition that resources should be located at 'PA'. The resources are for activity *Programming* of 35,000 lines of code and of *Location* 'Mexico'. Note that the **select** clause may contain either a resource type (e.g., *Programmer*) for simple workflow activities that only require a single resource, or a role specification (e.g., R_2b) for complex activities that require multiple resources. For resource specification in a workflow activity, only **select** clause is mandatory (from the system-wide or the process-wide default resource managers).

2. **Resource Policy language (RPL)** – RPL allows managers/supervisors to define resource policies. There are two types of polices: requirement policies and substitution policies. A requirement policy defines additional conditions a resource must satisfy in order to perform a given workflow activity, while a substitution policy specifies possible substituting resources for a given workflow activity in case the originally specified primary resource is not available. They are discussed in more detail in [3].
3. **Resource definition language (RDL)** – RDL allows for graphical modeling and manipulation of resource groups.

There are three additional interfaces for resource management:

- **Security interface** – interface which allows each GRM to use the given security architecture.
- **Administration API** – an API which allows the administration and configuration of the GRM.
- **RM-RM protocol** – a set of messages that allows one GRM to contact another GRM. This interface can allow for decentralized resource management and may allow for the use of a foreign router by putting an appropriate wrapper around it.

4.2 RM-RM Protocol

There are four types of message that a GRM can send to another GRM: *Plead*, *Delegate*, *Refer*, and *Report*. These messages are shown in the Figure 7 below. Note: *Do* is not really part of the RM-RM protocol, since an SRM by definition can access LRMs.

- *Plead***:** Used by a GRM to send a request up to a higher level GRM.
- *Delegate***:** Used by a GRM to send a request down to a lower level GRM.
- *Refer***:** A GRM can cache information about other GRMs that are horizontally positioned. As we will see, the cache does not have to be consistent.
- *Report***:** A response sent back to the original GRM where the request originated. Used to create and update cache entries at the original GRM.

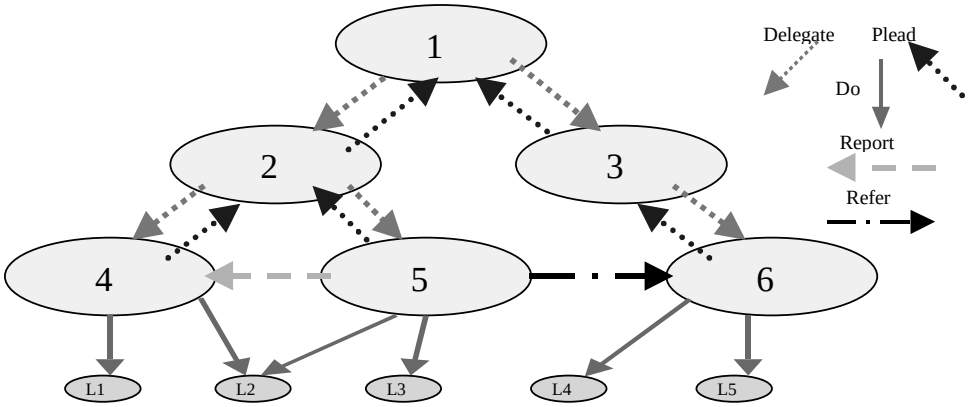

Figure 7: RM-RM Protocol

This protocol can be simplified for three levels of resource management. With three levels of resource management, the top level ERMs can either *Delegate* or they can *Do* (satisfy) the request. The second level of SRMs can *Plead*, *Refer* or *Do* (satisfy) a request. This is shown in Figure 8, where a request comes in to SRM_2. In the simplest case (shown in red), SRM_2 can satisfy the request and does so by using LRM_1

In the second case (shown in blue), SRM_2 does not know how to satisfy the request so it pleads up to ERM_1. ERM_1 maintains high level information about which SRMs can satisfy what. ERM_1 delegates the request to SRM_3. SRM_3 uses LRM_2 to satisfy the original request and replies directly back to SRM_2 using the *Report* message. SRM_2 can create a cache entry to send all requests of this type directly to SRM_3.

In the third case (shown in green), SRM_2 receives a request and uses its cached entry to send the request to SRM_3 using the *Refer* method. But SRM_3 cannot handle the request (i.e., invalid cache entry). SRM_3 might also have a cache entry for the request, but because it was called using a *Refer*, it *Pleads* the request up to ERM_1. Note, if SRM_3 were to use its cached entry on a *Refer* call this could lead to messy loops with inconsistent caches; therefore, SRM_3 pleads the request. An additional consequence of this is that we do not have to implement any cache consistency protocols. ERM_1 delegates the request to SRM_4 which satisfies the request and *Reports* directly back to SRM_2. At this point, SRM_2 can updates its cached entry.

The ordering of how these messages are called can be quite important. The following ordering results in optimal performance.

1. A resource manager first attempts to *Do* or satisfy the request.
2. If this fails, the GRM attempts to *Delegate* the request to another GRM (in a three level design, only ERMs can delegate).
3. The GRM attempts to use its cached entries and tries to *Refer* the request to another GRM.
4. Finally, it *Pleads* the request up one level (in a three level design, only SRMs may *Plead* requests).

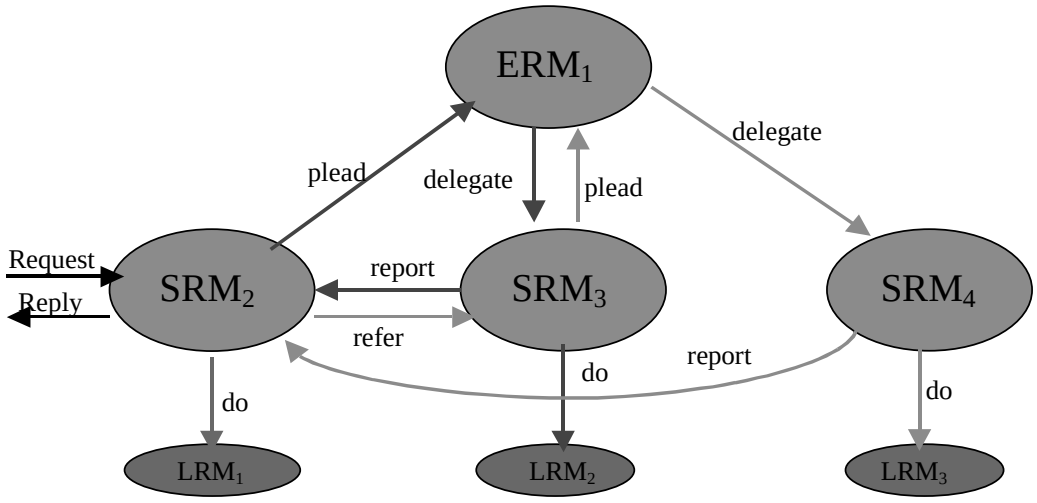

Figure 8: Use of RM-RM Protocol

The following are some rules that need to be followed for each of the messages to guarantee that no loops occur while traversing the hierarchy of resource managers. This simplifies the implementation of the RM-RM protocol and requires little external coordination (i.e.: we do not need to store all the resource managers we have visited).

- Request

  If can_Do then {*DO*; *Reply*}

  Else if can_Delegate then {*Delegate*}

  Else if can _Refer then {*Refer*}

  Else if can_Plead {*Plead*}

  Else {*Report* error}

- Reply

  Return result to initiator of Request

- Do (outlined in control engine section above)

  Ask policy engine for initial query rewrite

  Ask resource engine for result.

  If any result then return result

  Else ask policy engine for substitute rewrite

  Ask resource engine for result and return

- Delegate

  If can_Do then {*DO*; *Report* result}

  Else if can_Delegate then {*Delegate*}

  Else {*Report* error}

- Plead

  If can_Do then {*DO*; *Report*}

  Else if can_Delegate then {*Delegate*}

  Else if can_Plead {*Plead*}

  Else {*Report* error}

- Refer

  If can_DO then {*DO*; *Report*}

  Else if can_Delegate then {*Delegate*}

  Else if can_Plead then {*Plead*}

  Else {*Report* error}

- Report

Update refer table and *Reply* result

The can_DO functionality was described above (involves looking at the resource model). The other test cases (can_Refer, can_Delegate, etc.) can also be determined by using the parameters of the resource model described below.

Note that there are popular protocols used by LRMs. The integration layer (mentioned in Figure 2) manages the interfaces to the LRMs. Some of the more common interfaces might be SQL, LDAP [6], and CORBA Trader [7].

4.3 Consistency Issue

The resource model is loosely consistent among all the resource managers. This means that at any given instant in time, there might be GRMs that do not have the same model as some other GRMs. But over time, all GRMs will have the same resource model. This can be accomplished through the use of protocols such as SLP [8].

One exception to the above statement of loose consistency is the consistency of the model between the SRM and the ERM. If the model is changed in one of the SRMs, then the ERM(s) are immediately notified. The models in the other SRMs can be updated over time. This is required because other SRMs will be pleading requests up to ERMs and they need to have a complete, consistent knowledge of the world.

The SRM models can be subsets of the ERM models. For example assume there are two SRMs (SRM_1 and SRM_2) and they both plead up to ERM_1. Using the model in Figure 4, assume that SRM_1 can satisfy requests of type *Hardware* but does not have any knowledge of any other resource types. If asked for any resource type other than *Hardware*, SRM_1 pleads the request up to ERM_1. Further assume that SRM_2 can satisfy requests of *Employee* type (and all its subtypes). It would be acceptable for SRM_1 to have the following sub-model:

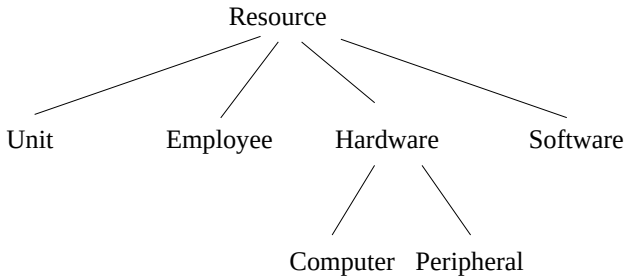

Figure 9: Sub-Model in an SRM

If a request for an *Engineer* came to SRM_1, it would plead the request up to ERM_1. ERM_1 (since it contains the complete consistent resource model) would know how to handle *Engineer* resources and delegate the request to SRM_2. SRM_2 would then report the result directly back to SRM_1. SRM_1 could update its model and cache when it received the result from SRM_2. In this way the model in SRM_2 could be built up over time. In summary, the SRM models are loosely consistent and are built up over time whereas the ERM model is complete and consistent.

5 Conclusions

Resource management is a key issue in providing resource independence and efficient use of workflow resources. The paper has outlined a design of such a resource management system for enterprise workflow environments. It is capable of handling a large number of workflow resources that are independently managed by pre-existing resource systems. It integrates these external resource systems at schema level without duplicating individual resource information. Resource specification is greatly simplified by providing process designers integrated views of enterprise workflow resources at different level based on a unified resource model. Dynamic behaviors of workflow resources are supported through powerful resource policies.

This work is done in the context of OpenPM [1], a workflow research project at HP Labs that results in Chang*e*ngine [4], the HP's current generation of workflow product.

6 References

1. J. Davis, W. Du, and M. Shan. OpenPM: An Enterprise Process Management System, *IEEE Data Engineering Bulletin*, 1995.
2. W. Du, G. Eddy, M. Shan, and J. Davis. Distributed Resource Management in Workflow Environment, *Proc. of the 5th Int. Conf. on Database Systems for Advanced Applications*, Melbourne, Australia, April 1997.
3. Yan-Nong Huang and Ming-Chien Shan. Policies in a Resource Manager of Workflow Systems: Modeling, Enforcement and Management, *Technical Report HPL-98-156*, Sept. 1998.
4. HP. Intrudtuction to HP Chang*e*ngine, *HP Product Document*, http://www.hp.com/go/changengine.
5. IBM. Modeling Workflow: Version 2 Release 3, *IBM FlowMark Document*, 1996.
6. LDAP. Understanding LDAP, http://www.redbooks.ibm.com/SG244986/4986fm.htm
7. OMG. Trader Object Services Specification, ftp://www.omg.org/pub/docs/formal/97-12-23.pdf.
8. SLP. Service Location Protocol White Paper, http://playground.sun.com/srvloc/slp_white_paper.html.