History Management in Workflow Systems *

Pinar Koksal Sena Arpinar Asuman Dogac
Software Research and Development Center
Dept. of Computer Eng. Middle East Technical University
06531 Ankara Turkiye
{*pinar, nural, asuman*}*@srdc.metu.edu.tr*

Abstract. A workflow history manager maintains the information essential for workflow monitoring and data mining as well as for recovery and authorization purposes.

Certain characteristics of workflow systems like the necessity to run these systems on heterogeneous, autonomous and distributed environments and the nature of data, prevent history management in workflows to be handled by the classical data management techniques like distributed DBMSs.

In this paper, we describe history management, i.e., the structure of the history and querying of the history, in a fully distributed workflow architecture realized in conformance with Object Management Architecture (OMA) of OMG. We describe the structure of the history objects determined according to the nature of the data and the processing needs, and the possible query processing strategies on these objects using the Object Query Service of OMG. We then introduce a cost model and discuss the optimization issues in this environment.

1 Introduction

A workflow system can be defined as a collection of processing steps (also termed as tasks or activities) organized to accomplish some business process. A task may represent a manual operation by a human or a computerizable task to be invoked. Computerizable tasks may vary from legacy applications to programs to control instrumentation.

Workflow history management provides the mechanisms for storing and querying the history of both ongoing and past processes for the following purposes:

•*Monitoring purposes:* During the execution of a workflow, the need may arise for looking up some piece of information in the process history, for example, to figure out who else has already been concerned with the workflow at what time in what role, or to monitor the current states of the tasks of an executing process instance [9].

•*Business Process Reengineering purposes:* Aggregating and mining the histories of all workflows over a longer time period form the basis for analyzing and assessing the efficiency, accuracy and the timeliness of the enterprise's business processes [9].

•*Recovery purposes:* If there are executing process instances on a site when the site fails, history is used to recover the information required to continue their executions.

*This work is being supported by the European Commission, Project Number: INCO-DC 972496, by International Bureau Research Center Julich of Germany, by Middle East Technical University, Project Number: AFP-97-07.02.08 and by the Scientific and Technical Research Council of Turkey, Project Number: 197E038.

•*Authorization purposes:* The history information of past and present processes can be used in scheduling the user tasks by the authorization service. For example, it may be necessary to assign a task to a user who did the task previously.

Since workflows are activities involving the coordinated execution of multiple tasks performed by different processing entities, mostly in distributed heterogeneous environments, a distributed workflow scheduler architecture is essential. And in order to fully exploit the advantages brought by the distributed scheduling, history management, workflow relevant data management and the worklist management should also be handled in a distributed manner. Distributed execution necessitates the handling of interoperability problem among heterogeneous resources. The interoperability of applications on heterogeneous platforms can be handled by using CORBA [1] as the communication infrastructure. The workflow history can then be implemented where the history of each activity instance is a CORBA object. In fact, this approach has been implemented within the scope of METUFlow project [3].

In this paper, the querying of workflow history is addressed. Since history information is kept in distributed CORBA objects, the problem converges to querying distributed objects and optimization of query processing. For querying the history objects, Object Query Service (OQS) defined by OMG [4] is used.

The history management for workflow systems has not been extensively studied in the literature. In the ConTract model [7], the set of private data defining an application specific computation state is called *Context* and is preserved for forward recovery. The ConTract manager tries to overcome resource failures and re-instantiates an interrupted ConTract by restoring the recent step consistent state and then continues its execution according to the specified script. In [8], two approaches are proposed. The first approach aims to enrich an audit trail, and the second one takes the viewpoint that a workflow log is a special kind of temporal database.

The paper is organized as follows: Object Query Service is described briefly in Section 2. In Section 3, workflow history structure is explained. Different querying strategies of the workflow history are given in Section 4. Optimization of these queries is discussed in Section 5. Comparison of costs of executing queries using different strategies is provided in Section 6.

2 Object Query Service

The Object Query Service, defined by OMG [4], provides operations of selection, insertion, updating and deletion on collections of objects.

The Query Service (QS) provides a framework consisting of some interfaces to deal with the preparation and execution of a query. These are QueryEvaluator, QueryManager, Collection, Query and QueryableCollection. **QueryEvaluator** (QE) executes the query using the query language user specifies. When a user executes a query, the service returns a *collection* of objects that satisfy the search criteria the user specifies via a select operation. **Collection** defines the operations that let the user add, replace, retrieve and remove members of a collection. The details can be found in [4].

3 Workflow History Structure

Workflow history is managed by using CORBA to support the network and location transparency. History of each process and task instance is a different CORBA object,

invoked by the workflow scheduler in case of initialization, start, abort and commit of an activity which is a process or a task.

The process and task history objects should store the name of the process (task), start, abort and commit time information, the object identifiers of its tasks and subprocesses, and the object identifiers and version numbers of the data used as input and output parameters. Task history objects also store the user name and the role name.

The persistency of history objects are provided by mapping them via storage wrappers to a database or to other available data repositories. *PROCESS* and *TASK* tables store the information about processes and tasks respectively; additional *PROCESS_CHILD* table keeps the children of a process; and *ACTIVITY_DATA* table keeps the parameter information of process and task instances.

4 Querying the Workflow History

A client of a workflow history can be a workflow administrator, the authorization service, or the workflow scheduler. The definition of the history is given in ODL, the Object Definition Language of ODMG [2]. Clients can query the history according to this ODL definition using OQL [2], standard query language of ODMG. An OQL query is evaluated using the Object Query Service.

The characteristics of the queries issued against the history are as follows:

• The queries are evaluated on distributed CORBA objects.

• Any history information can be at any of the data repositories. Therefore, all the repositories should be queried.

• Because the history of a workflow can contain not only the information of the current process instances, but also that of past instances, the size of the data repositories may be very large.

• Most of the queries either find the tasks of a given process or retrieve the information on given tasks. To retrieve the information of tasks of a process instance, first task identifiers of the process are retrieved, then the data repositories are queried with all these task identifiers, since process and task information may stored in different sites.

In the following, we explain basic processing strategies through an example. Later in Section 5, we provide a cost model to determine the most efficient way of executing these queries.

Example: Assume that there is a process *p1* defined as:

```
PROCESS p1 {
    T1 (in i, out j);
    T2 (in j, out k);
    AND_PARALLEL {
         T3 ();
         T4 ();  }
    T5 (in k, out m); }
```

In this process definition tasks T1 and T2 are executed serially and then tasks T3 and T4 are started in parallel within the scope of the AND_PARALLEL block. T5 is another task in this process to be executed serially. More details about the specification language are given in [3].

The following is an example OQL query against the instances of this process definition:

Query: Find the active tasks of the instances of process *p1*. The query written in OQL is; *"select t.name from process p, task t where p.name = 'p1' and t in p.children and t.state = 'EXECUTING';"* where *children* in the where clause are a set of object

identifiers of the activities, which are subprocesses or tasks, that a process instance owns. Because the history is distributed, the detailed task information can be at any of the sites. Therefore, to answer such a query, two passes on the data repositories are needed, the first pass to find the object identifiers of the activities of process instances and the second pass to get the detailed information on these tasks.

If we assume that there are *ns* sites that the history objects are stored, then there should be *ns* history repositories. In addition, there can be many instances of the process *p1* dispersed along these sites. To answer this kind of query, *ns* sources should be queried using Object Query Service.

There are several strategies to evaluate this kind of query. Note that most of the queries against workflow history are in this form.

Strategy 1. In the first pass, *ns* query evaluators (QE) should retrieve the children of the process 'p1' by performing join operation between *PROCESS* and *PROCESS_CHILD* tables. Then a collection of task identifiers is retrieved from each repository. Each QE should send its collection to the other *ns-1* QEs so that each of the *ns* QEs contains the whole set of task identifiers.

In the second pass, using the collection containing the set of task ids of the process instances, *tasks_coll*, the information of tasks is retrieved by sending as many select queries as the number of task ids in the collection of *tasks_coll* to each of the repositories since the task information can be at any of the data repository. After all the subresults are obtained from the QEs, a union of these subresults gives the final result.

Strategy 2. The only difference between this strategy and the previous one is that, after *ns* QEs have retrieved the task identifiers from the repositories, they do not send their collections to the other *ns-1* QEs, instead all the collections of the QEs are collected at one collection, and then complete collection of the task identifiers are sent back again to the *ns* QEs. The rest of the execution is the same as Strategy 1.

Strategy 3. In this strategy, again there are two passes. First pass is same as previous strategies. Before the second pass starts, we assign a threshold value to each of the task identifiers in the collections and set this value to one. The second pass starts for each of these collections in parallel in which the detailed task information is retrieved from the native systems. These retrievals do not follow a specific order of task identifiers, but are done randomly. If a task information is found in a repository, this site informs all the other sites so that task is deleted from all of the collections. If the task information is not found in a repository, then that task is deleted from the collection of this site and threshold value of that task is increased by one in all of the other collections. The reason why we increase the threshold in other sites is that the possibility of finding detailed information about the task in the other repositories has increased. When all the threshold values are no longer equal to one in a collection, the task whose threshold value is the highest is given higher priority in the retrieval.

Strategy 4. In this strategy, while child identifiers of a process are retrieved from repositories into a collection, all the task information in the underlying repositories are retrieved into another collection concurrently. Afterwards, these two collections (task identifiers and task information) are joined explicitly in a query evaluator.□

If the process is nested, i.e., it contains a subprocess, then these strategies become more complex. More passes are necessary to the data repositories to obtain information on the activities in various nesting levels.

5 Optimization Issues in Query Processing

Our primary aim is to minimize the cost of execution of the queries on workflow history. In this section, various strategies, given in the previous section, are analyzed in detail and cost functions are derived for each strategy.

We use the following parameters in the cost functions:

• *Communication cost,* ψ_{CC} *:* Cost of sending a unit between QEs.

• *Appearance cost,* ψ_{AC} *:* Time required to retrieve the result from DBMSs.

• *Cost of explicit join operation,* $\psi_{\bowtie}$ *:* The cost of join operation, if the join is performed at the QE level, not at the database level.

• ns is the number of sites on which objects are alive.

• k is the nesting level. Top process is at the nesting level 0, the subprocesses that are called by the top level process are at level 1, and so on.

• nc_i, is the total number of children of the process instances at nesting level i.

• nt_i, is the total number of task identifiers of the tasks of the process instances at nesting level i.

• nsp_i, is the total number of subprocess identifiers of the subprocesses of the process instances at nesting level i.

The cost analysis of executing queries, according to the given strategies in the previous section, is given in the following:

Strategies 1 and 2. In these strategies not only task identifiers, but also the subprocess instances are sent to all sites. The task identifiers of the subprocess instances are also gathered recursively.

1. First ns number of QEs are created and the *process* information is sent to these QEs, with the communication cost ψ_{CC}.

2. Since the total number of children of the process instances is nc_0 and the number of sites that objects are alive is ns, the number of child instances at one site is nc_0/ns on the average. Therefore the cost of retrieving child ids information becomes $(nc_0/ns)\ \psi_{AC}$. At this point ns QEs have the collections of child ids of their sites.

3. The ns QEs should have nc_0 child identifiers. This can be achieved in two ways. Either each site sends its collection of child identifiers to the other $ns\text{-}1$ QEs as in Strategy 1 with the cost of $nc_0 \cdot \psi_{CC}$, or the collection of child identifiers of each of the sites is collected in a single collection and then it is sent to ns QEs as in Strategy 2 with the cost of $2 \cdot nc_0 \cdot \psi_{CC}$. The one with the minimum cost will be chosen among these two possibilities in the execution of the query as $min\{nc_0 \cdot \psi_{CC},\ 2 \cdot \mathrm{nc}_0 \cdot \psi_{CC}\}$. According to this result, we can conclude that Strategy 1 performs better than Strategy 2 since this item is the only difference between these two strategies. At this point, ns QEs have all the nc_0 child identifiers.

4. The collection of nc_0 child identifiers contains the object identifiers of subprocess instances and the object identifiers of task instances of the process. While child identifiers of the subprocesses are being retrieved from the sites, task information can also be retrieved at the same time. If there is totally nc_1 number of children of these subprocesses, then the cost of retrieving child identifiers is $(nc_1/ns)\ \psi_{AC}$. To retrieve the task information each site creates nt_0 QEs. If the repository allows parallel execution of these nt select queries then the cost of this operation is it ψ_{AC}, as is the case of most of the DBMSs. However, if the repository does not allow parallelism as in files, then the cost becomes, $nt \cdot \psi_{AC}$. Therefore, we can generalize these two cost functions as, if the probability of a repository of executing in parallel is p_p, $p_p \cdot \psi_{AC} + (1\text{-}p_p) \cdot (nt\ \psi_{AC})$. At this point, ns QEs contain the task information of the top level process and the child

identifiers of the subprocesses at the corresponding sites.

5. For the child identifiers of the subprocesses, items 2 and 3 should be repeated recursively until there is no subprocess identifiers left in the collection of child identifiers. If we generalize this recursive operation, we obtain the following cost function: $c_i = nc_i \cdot \psi_{CC} + max\{(nc_{i+1}/ns)\psi_{AC} + c_{i+1}, p_p \cdot \psi_{AC} + (1-p_p)(nt_i \cdot \psi_{AC})\}$ where 0<=i<=k-1 and $c_k = nc_k \cdot \psi_{CC} + p_p \cdot \psi_{AC} + (1\text{-}p_p)(nt_k \cdot \psi_{CC})$.

6. After all the task information is retrieved, the collections of the tasks of each nesting level will be combined. The cost of this operation is $(nt_0 + nt_1 + \ldots + nt_k)\psi_{CC}$.

The general cost formula of this strategy is as follows: $\psi_{Sch_{1,2}} = \psi_{CC} + (\mathrm{nc}_0/\mathrm{ns})\psi_{AC} + \mathrm{c}_0 + \sum_{j=0}^{k} (\mathrm{nt}_j\ \psi_{CC})$.

Strategy 3. which assigns threshold values to the child identifiers.

1. The cost of sending *process* information to the *ns* QEs is ψ_{CC}.

2. The cost of retrieving child identifiers information is $(nc_0/ns)\ \psi_{AC}$.

3. The cost of sending child identifiers of QEs to other QEs is $nc_0 \cdot \psi_{CC}$.

4. The child identifiers of the subprocesses are gathered by executing item 2 and 3 recursively upto the k^{th} nesting level. This can be generalized as $\sum_{i=1}^{k} ((\mathrm{nc}_i/\mathrm{ns})\ \psi_{AC} + \mathrm{nc}_i \cdot \psi_{CC})$.

5. Retrieving task information from data repositories can be done in parallel with item 4. If we assume that nt_i/ns task information is at one repository, then the cost of retrieving task information recursively becomes: $\sum_{i=1}^{k} (\mathrm{p}_p \cdot \psi_{AC} + (1\text{-}\mathrm{p}_p)(\mathrm{nt}_i/\mathrm{ns})\psi_{AC} + \mathrm{nt}_i \cdot \psi_{CC})$.

6. At this point, each QE has nt/ns task information. To give the result of the query to the user, they are collected at one QE with the cost of, $nt\ \psi_{CC}$.

The general cost formula of this strategy is: $\psi_{Sch_3} = \psi_{CC} + (\mathrm{nc}_0/\mathrm{ns})\psi_{AC} + \mathrm{nc}_0\ \psi_{CC} + \sum_{i=1}^{k}(\max\{(\mathrm{nc}_i/\mathrm{ns})\psi_{AC} + \mathrm{nc}_i \cdot \psi_{CC}, \mathrm{p}_p \cdot \psi_{AC} + (1\text{-}\mathrm{p}_p)(\mathrm{nt}_{i-1}/\mathrm{ns})\psi_{AC} + \mathrm{nt}_{i-1} \cdot \psi_{CC}\} + \mathrm{p}_p \cdot \psi_{AC} + (1\text{-}\mathrm{p}_p)(\mathrm{nt}_k/\mathrm{ns})\psi_{AC} + \mathrm{nt}_k \cdot \psi_{CC} + \mathrm{nt}\ \psi_{CC}$.

Strategy 4. In this strategy, all the task identifiers of the subprocesses are collected and then they are joined with the *TASK* table at the QE level explicitly. The execution cost of this strategy is given below:

1. The cost of sending *process* info. to the *ns* QEs is ψ_{CC}.

2. The cost of retrieving child ids information from the sites is $(nc_0/ns)\ \psi_{AC}$.

3. Then, the collections of child identifiers are combined at a single collection which contains both the subprocess identifiers and the task identifiers of the process instances. The cost of this operation is $nc_0\ \psi_{CC}$.

4. To retrieve the child ids of the subprocesses, *ns* data repositories should be queried again. First, the collection of subprocess ids are sent to *ns* QEs with the cost of $nsp_0 \cdot \psi_{CC}$. Then, they are queried with the cost of $(nc_1/ns)\ \psi_{AC}$.

5. The items 3 and 4 are repeated recursively until there is no subprocess identifiers in any of the collection of child identifiers. This can be generalized as: $\sum_{i=0}^{k} (\mathrm{nsp}_i \cdot \psi_{CC} + (\mathrm{nc}_i/\mathrm{ns})\psi_{AC} + (\mathrm{nc}_i/\mathrm{ns})\psi_{CC})$. At this point all the task identifiers of all the subprocesses are gathered in k collections.

6. The k collections of task identifiers are gathered in one collection to join with the cost of, $nt\ \psi_{CC}$ where nt is the total number of task identifiers of all subprocesses.

7. While retrieving the task identifiers of the subprocesses, the *TASK* table can be read sequentially at the same time from *ns* data repositories. If the number of blocks of *TASK* table of one repository is b on the average, then the cost becomes $b \cdot ebt$.

8. Then, this information should be gathered at one QE. If the cardinality of a table is |T| on the average, then the cost of this operation is, $ns\ |T|\ \psi_{CC}$. In fact, retrieving task information from repositories (items 7 and 8), can be executed in parallel with the

retrieval of task identifiers of process instances (items 1 to 6). Therefore, in the general cost formula, the maximum of these costs should be considered.

9. Join operation should be performed between the collection of task identifiers of process instances at one QE, and the collection of task information at another QE. Different join strategies can be considered at this step. For the time being, cost of this operation is taken as, $\psi_{\bowtie}$.

The general formula of this strategy becomes:

$\psi_{Sch_4} = max\{\psi_{CC} + (nc_0/ns)\psi_{AC} + \sum_{i=0}^{k}(nsp_i \cdot \psi_{CC} + (nc_i/ns)\psi_{AC} + (nc_i/ns)\psi_{CC}) + nt \cdot \psi_{CC}, b \cdot ebt + ns \cdot |T| \cdot \psi_{CC}\} + \psi_{\bowtie}$

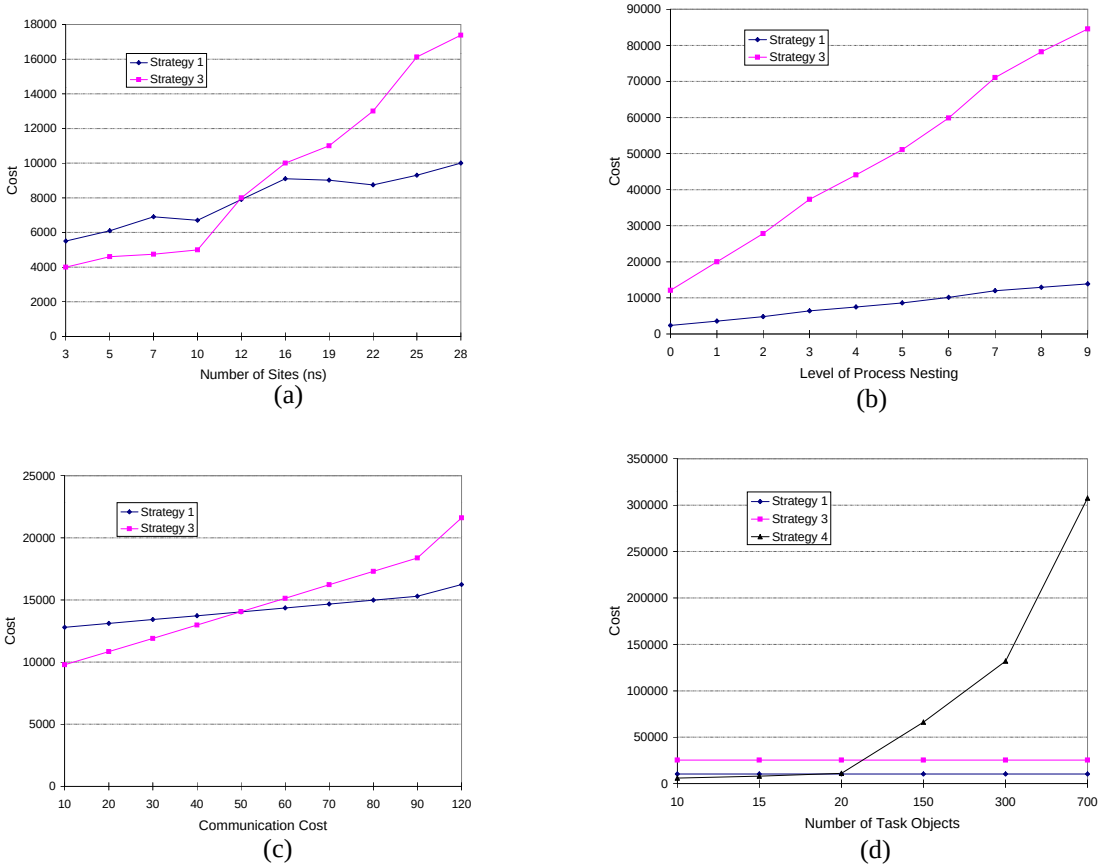

Figure 1: Execution Costs

6 Comparison of Strategies

The execution costs of the different query processing techniques given in the previous section, are compared by considering the following metrics: *1.* number of sites involved, *2.* number of nesting levels in a process, *3.* communication cost, *4.* number of task objects in data repositories. For each of these criterion, we calculate the execution costs according to the formulas given in Section 5. In each calculation, the parameters involved are randomly chosen with different variances for a total of 10 test cases and the results are averaged.

These formulas are plotted into the graphs given in Figure 1. Since Strategy 3 has a high communication overhead, for small number of sites, up to 12, this strategy performs better as seen in Figure 1a. However as the number of sites increases, Strategy

1 produces lower execution cost. This implies that the high execution cost of Strategy 1 is remedied when the number of sites increases.

Strategy 4 performs the necessary join operations at the Query Evaluators which do not have index structures to help with this operation. Therefore Strategy 4 performs so poorly that we have removed it from Figures 1a, b, c so that the performance difference of the other techniques can be more clearly seen in the figures.

Figure 1b demonstrates the performance of the strategies as a function of the number of nesting levels in a process, and Strategy 1 performs the best. Since the communication between sites increases with process nesting, Strategy 3 performs poorer than Strategy 1 due to this communication overhead.

Figure 1c demonstrates how the variances in the communication cost effects the cost of strategies. For low communication cost, Strategy 3 performs better than Strategy 1. Unlike in Strategy 1, in Strategy 3 the communication cost is dominant rather that appearance cost. This explains the better performance of Strategy 3 over Strategy 1 for low communication overhead.

When the number of task objects are varied as shown in Figure 1d, Strategy 1 performs the best. Note that we assumed that indices are available in the underlying repositories for retrieving task objects and therefore Strategy 1 and Strategy 3 use these indices and thus their appearance time is not effected by the number of task objects. However for Strategy 4, since a Query Evaluator performs the join operation and indices are not available at this level, the cost gets higher.

In evaluating a query the History Manager considers these strategies to find out a cost effective plan. However since Strategy 2 and 4 consistently perform worse than Strategy 1 and 3, they are not taken into consideration. Details of this work is provided in [5] and a shorter version of this paper can be found in [6].

References

[1] *The Object Management Architecture Guide, Version 2.1.* OMG Pubs, 1992.

[2] R. Cattell. *The Object Database Standard: ODMG-93, Release 1.2.* Morgan Kaufmann, San Francisco, 1994.

[3] Dogac, A., Gokkoca, E., Arpinar, S., Koksal, P., Cingil, I., Arpinar, B., Tatbul, N., Karagoz, P., Halici, U., and Altinel, M. Design and implementation of a distributed workflow management system: Metuflow. *Proc. of NATO-ASI on Workflow Management Systems and Interoperability*, pages 60–90, August 1997.

[4] O. M. Group. The common object services specification. 1(OMG Document Number 94.1.1), January 1994.

[5] Koksal, P., Arpinar, S., and Dogac, A. Workflow history management. *Middle East Technical University, Software Research and Development Center, Technical Report*, (1), January 1998.

[6] Koksal, P., Arpinar, S., and Dogac, A. Workflow history management. *ACM Sigmod Record*, 27(1), March 1998.

[7] Wachter, H. and Reuter, A. The contract model. *Database Transaction Models for Advanced Applications*, Morgan Kaufmann Pub., 1992.

[8] G. Weikum. Workflow monitoring: Queries on logs or temporal databases? *Position paper in HPTS'95*, 1995.

[9] G. Weikum. *Personal Communication*, 1996.