Logging and Post-Mortem Analysis of Workflow Executions based on Event Histories*

Andreas Geppert and Dimitrios Tombros

Department of Computer Science, University of Zurich
{geppert,tombros}ifi.unizh.ch

Abstract. Logging and post-mortem analysis of workflow executions are important tasks of a workflow management system. In this paper, we show how both tasks are addressed in EvE, a prototypical distributed system implementing event-driven workflow execution. Essentially, both tasks benefit from the concept of an *event history* as it is present in active database systems. By using event-based workflow execution, the workflow log actually corresponds to the event history, and post-mortem analysis is accomplished through querying the event history. In our approach, an analysis of the required information has led to the design and development of a workflow analysis query service.

1 Introduction and Motivation

Workflow management [11, 15] is concerned with the specification and enactment of business processes. Workflow management systems (WfMSs) are software systems that implement these tasks. In addition to the primary tasks of specification and execution of workflows, a workflow management system also has to support additional functionality, such as the simulation, analysis, and evolution of workflow specifications, as well as analysis and maintenance of audit trails of workflow executions. Keeping an audit trail (or synonymously, workflow log) is an essential task for several reasons. First, many organizations, such as banks, are obliged to keep records of their business operations. Second, workflow specifications need to be maintained as any other kind of software, too. The need to modify workflow specifications can stem from changing requirements or inefficiencies and bottlenecks detected in old specifications. Such problems as well as possible optimizations can be determined based on the analysis of the workflow log. The other way round, without precise information about previous workflow executions, it can only be speculated which optimizations and improvements should be undertaken.

In this paper, we show how logging and analysis is performed in EvE [13], a distributed system providing event-driven workflow execution. In event-driven workflow management [4, 6, 8, 14, 16], the technology of active database management systems (ADBMSs) [1] is used to express certain aspects of workflow management, such as control flow enforcement [14] or agent synchronization [4].

* In Proc. 3^{rd} Int'l Workshop on Rules in Database Systems (RIDS), Skoevde, Sweden, June 1997. Lecture Notes in Computer Science 1312, Springer-Verlag, 1997

Below, we show how ADBMS-technology can be adapted for logging and analysis of workflow executions. Concretely, we utilize the concept of an event history, as it is maintained by some ADBMSs (e.g., SAMOS [10] and REACH [3]).

While we adopt the concept of event history from ADBMSs, the implementation of events and the event history must be modified/extended in three ways:

- further attributes of event occurrences must be defined and stored (e.g., the workflow in which it occurred, distribution information, etc.),
- the event history must be maintained in append-only mode (whereas an ADBMS typically removes event occurrences upon event consumption),
- the event history must be made explicit to allow queries.

Based on these extensions, workflow logging and analysis can be provided in a quite elegant way, resulting in synergies of active systems technology and workflow management concepts.

The remainder of this paper is organized as follows: in the next section, we present an overview of our approach to workflow management. In section 3, we introduce the notion of event history in EvE, which is the fundamental concept for logging and analysis. Section 4 describes how workflow analysis is performed using the event history, as well as the design and implementation of an analysis query service. Finally, section 5 concludes the paper.

2 Overview

2.1 Workflows

A *workflow type* is the enactable specification of a (business or design) process. The workflow specification defines the process steps. Execution dependencies between the steps are also defined. There are ordering, temporal, and output-data dependencies. A step can either be an elementary activity or in turn a workflow. In the first case, the activity is performed by a person or a software system. Both are henceforth summarized under the term *processing entity* (PE). The workflow specification assigns responsible PEs to activities. In the second case, a step is a subworkflow and thus has in turn a complex structure and defines steps, constraints, and responsible PEs. A *workflow instance* is a concrete workflow executed according to the definition of its type. A WfMS is a software system that supports workflow specification and enactment.

In Figure 1[2] we present as an example the processing of a health insurance claim (HIC). The workflow is initiated once the HIC mail arrives at a local insurance agency. A clerk working at the local agency creates a file containing the diagnosis, treatments, costs, and the insurance number (activity `accept_case`). If the claimed amount is below 300 Swiss Francs the HIC is directly accepted

[2] The meaning of the symbols is as follows. 1-in-N-split: exactly one of the successor activities is executed. AND-split: all the successor activities are executed. OR-join: at least one of the predecessors must have terminated with a certain result. AND-join: all the predecessors must have terminated with specific results.

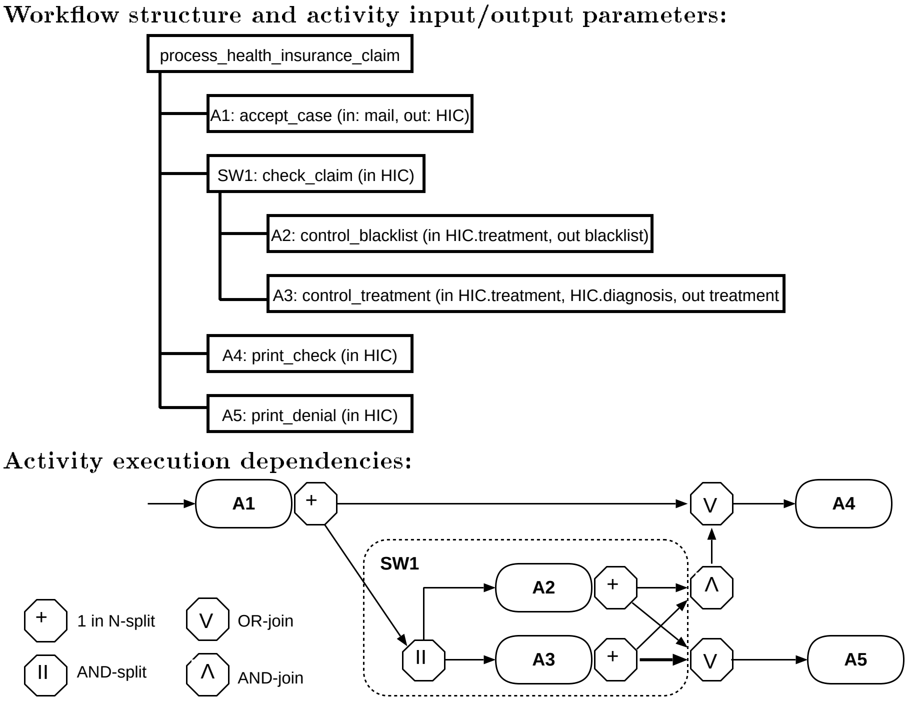

Fig. 1. The health insurance claim example workflow

and paid (activity `print_check`). Otherwise, the HIC has to be processed at the central company clearing center. There a sub-workflow is started, which consists of activities that may be performed in parallel. One is to check whether some of the treatments are contained in a blacklist, in which case their coverage will be denied (activity `control_blacklist`). An activity of type `control_treatment` controls whether the treatment actually suits the diagnosis. If any of these controls fails, an entry is made in the customer record and a notification of the rejection is printed at the local agency (`print_denial`). Otherwise, a payment check can be printed at the local agency (`print_check`).

2.2 Brokers and Services

Besides workflow specification and enactment, further requirements such as the ability to integrate external systems and to reuse old artifacts (parts of specifications, mediation software) are important for WfMSs. All these requirements can only be met if various aspects of a WfMS-architecture are exactly defined:

– *components* representing PEs that take part in one of the workflows—the structural and informational aspect,

- *tasks* that can be performed by those components—the functional aspect,
- *parameterized* events through which components interoperate—the operational aspect, and
- *rules* determining how, when and under which constraints the components can/must perform the activities—the behavioral and temporal aspects.

We propose the Broker/Services Model (B/SM) [20] as a model for such architectures. Workflow specifications are mapped onto B/SM and are automatically augmented with the necessary elements to attain executability. B/SM is described in detail in [20] so that this section gives only a short description for the sake of comprehension.

Brokers represent components of a WfMS, i.e., they model PEs involved in the execution of one or more workflows in a way that allows them to interoperate. They are *reactive* and offer *services* to other brokers (their clients). Brokers can access the functionality of their peers through service requests. The set of activities that can be executed within a WfMS (i.e., its functionality) is described by services. A service is specified by a *signature* consisting of the service name, a set of typed input parameters bound at request time, a possible reply with its output parameters, as well as exceptions the request may cause (e.g. due to a service being temporarily unavailable). Broker behavior is defined in terms of event-condition-action (ECA) rules, which define their basic capabilities, i.e., reaction to request events, or to complex workflow-specific events such as the termination of multiple activities within a workflow. The supported composite event constructors are conjunction, sequence, disjunction, negation, repetition, and concurrency (i.e., two occurrences generated at different sites cannot be ordered temporarily). In our example workflow we use the following brokers:

- a broker representing customers generating the requests to process health insurance claims
- insurance agents which provide the `accept_case` service (represented by the brokers `Worklist-Meier` and `Worklist-Hefti`). Their behavior is expressed by the rule:

```
ON   request(accept_case, mail)
IF   (HIC = create_claim(mail)) AND HIC.amount > 300
DO   request(control_blacklist, HIC.treatment)
     request(control_treatment, HIC.treatment, HIC.diagnosis)
```

- multiple PEs situated at the clearing center: a database application (broker `DBApp-RSA`) to check acceptance of the treatment (`control_blacklist`) and several experts (e.g., broker `Exp-Seehofer`) which can check the compatibility between treatment and diagnosis (`control_treatment`). If both controls are successful, the check can be printed as expressed by the rule:

```
ON   reply(control_treatment, treatment) AND
     reply(control_blacklist, blacklist)
IF   treatment="OK" AND blacklist="NO"
DO   request(print_check, HIC)
```

– a demon (broker `Printer`) providing printing services on printers at the insurance agency (`print_check`, `print_denial`).

A specific service is provided by one or more brokers and can be initiated by client brokers through request events. Service execution is terminated when the reply or one of the possible exception events defined for the service is generated. The brokers which provide a service requested during the execution of a workflow are determined by way of their basic capabilities as well as through *responsibility* predicates, which express organizational constraints (e.g., task delegation) or task-specific requirements (e.g., authorizing the payment of a large amount of money). Responsibilites only define the eligible PEs; in case there are multiple brokers eligible at runtime, it can be specified which of them should serve the request (e.g., randomly, or based on the current load of the candidates).

In typical cases, workflow activities consume input data, produce output data, and read and manipulate data items stored in some sort of data store. Thus, *production data* must flow through a workflow according to the workflow's definition. Most of the production data will be managed by external systems and in the general case, it is not justified to assume that the WfMS and the production data store are tightly integrated. However, the B/SM supports access of processing entities to these data through operations defined for their types.

In B/SM, workflows are defined by a name, a set of initiating requests and a set of terminating replies. The workflow structure and related execution constraints are defined by the reaction of certain brokers to specific situations (e.g., reaching a deadline). Workflows are executed by the execution of the services provided. A workflow starts executing when its *initiation event* occurs which is a request of the first service to be provided.

2.3 EvE

Brokers and consequently, workflows are executed through the use of the distributed event engine *EvE* [13]. A comprehensive description of EvE is beyond the scope of this paper; we thus introduce only its general concepts as far as needed for the understanding of this paper.

EvE's major purpose is to support brokers by providing event management, detection, storage, and broker notification in distributed environments. It needs several database functions (e.g., persistence, transactions, object management). Preferably an object-based system is used in order to prevent the impedance mismatch between the object-oriented broker model and the persistent platform. Furthermore, as previous experiments have shown, object-oriented ADBMSs are of little benefit, because they do not provide the sufficient functionality (e.g., with respect to event types), so that we decided to build EvE on top of the object manager Shore [5] and to implement the missing functionality ourselves.

EvE consists of various server nodes to which brokers are connected. The principle by which brokers use EvE is the following: broker ECA-rules are translated to rules stored and executed in EvE. When a broker generates a primitive event, it notifies its local EvE-server about the occurrence. EvE then performs composite event detection and determines brokers that have registered a notification interest for such primitive and potentially detected composite event occurrences which may result from this event. EvE then performes *task assignment* by determining those brokers that should react to the event (e.g., by executing a requested service). These brokers are then appropriately informed and react as defined by their ECA-rules (whereby these reactions can in turn generate new events, and so forth). An important task of EvE upon event detection is the maintenance of an event history in which all primitive and composite event occurrences are represented.

In the present paper, we will not consider in detail the technical aspects of event generation and (composite) event detection in EvE. It is sufficient to note that these actions take place in a distributed environment, a fact that has to be considered for the correct definition of the semantics of events [21].

3 The Event History in EvE

In this section, we introduce the event history in EvE and show how it can be used to accomplish workflow execution logging.

3.1 Structure of the Event History

Figure 2 how the relevant structures of event occurrences and the event history in OMT-notation (operations, attributes, and subclasses have been omitted).

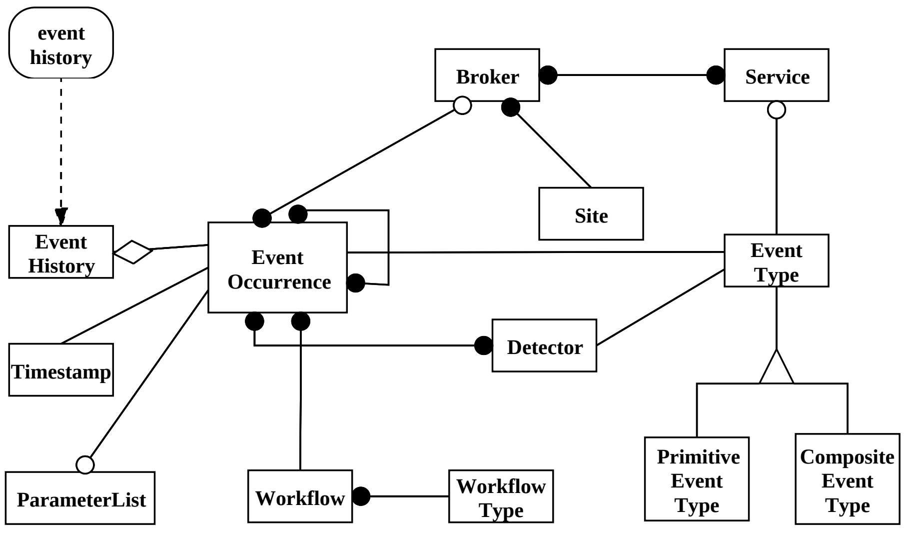

Fig. 2. Object Model of EvE Data Structures

An event occurrence has several attributes:

- a reference to the event type for which it occurred,
- a reference to the workflow in which it occurred,
- a reference to the broker which generated the event,
- a timestamp specifying at which point in time the event has been detected by an EvE-server,
- service-specific parameters in case of service request or reply events,
- a list of references to component events (in the case of composite events).

For each event type exists an event detector. In case of composite events, the detector maintains references to those component event occurrences that can still be used for event composition. Ultimately, the event history contains a sequence of event occurrence objects as components. Figure 3 shows the class definitions of event occurrences (`EventOcc`) and the event history in SDL-syntax [17].

The current naming and format of log entries in EvE is not compatible to the Workflow Management Coalition audit data specification standard [22] which also requires the logging of activity state information. It is however, straightforward to map the event information stored in EvE to the proposed abstract specifications. Although our model does not have to explicitly define activity states, activity instance state modification events can be mapped to service request, reply and exception events.

```
interface EventOcc {
      public:
            attribute ref<Broker>                origin;
            attribute ref<EventOcc>              refers_to;
            attribute ref<Timestamp>             timestamp;
            attribute ref<EventType>             etype;
            attribute sequence< ref<EventOcc> >  comp;
            attribute ref<Workflow>              wfid;
            attribute ref<ParameterList>         parameters;
            ...
}

interface EventHistory {
      private:
            attribute sequence< ref<EventOcc> >    occs;
      public:
            int  addOcc(in ref<EventOcc> newocc);
            int  dump(in lref<char> filename) const;
            ...
}
```

Fig. 3. Interfaces EventOcc and EventHistory

3.2 Event Detection and Workflow Logging

Events are detected in EvE by dedicated event detectors. For composite event detection, we have adapted the technique of event graphs (which has originally been proposed for Sentinel [7] and has also been used in REACH [3]) to distributed systems. For event composition, the *chronicle* parameter context [7] is used, since this mode offers precisely the semantics required by workflow management (in the chronicle mode, the oldest appropriate component is chosen whenever there are multiple candidates for event composition).

During event detection, each detected (primitive or composite) event occurrence is also inserted into the event history. ADBMSs typically maintain only parts of the event history, e.g., by storing candidate component events in composite event detectors (e.g., SAMOS [10]). Event occurrences typically are removed as soon as the event occurrence has been consumed for event composition. In EvE, however, event occurrences are never removed (except upon explicit request from an administrator).

Summarizing, workflow logging actually means to maintain the event history. In fact, workflow logging is (operationally) for free, as it is done during event detection. We do not expect that EvE in general and the event history in particular will turn out as bottlenecks, because EvE is solely responsible for coordination of PEs (who perform the "real" work). In other words, EvE's tasks will consume only a fraction of the time needed by PEs to provide their services, and thus EvE will be able to serve large numbers of PEs in a timely manner. Moreover, EvE's approach to workflow enactment based on compiled ECA-rules is more efficient than those interpretating workflow specifications.

Note that over time the event history might grow very large and that it can contain information about workflows that have terminated long ago. If this information is no longer needed for analysis, it can be dumped to a file and removed from the history.

Table 1 shows a sample event history for the HIC-workflow. The example reflects three workflows (3245 through 3247).

4 Workflow Analysis

We now discuss the analysis of workflow executions based on the event history.

4.1 Evaluation Criteria

A number of criteria/queries can be used for the analysis and evaluation of an event history at a given time:

1. (a). How many workflows are currently active? (b). How many have terminated successfully? (c). For how many is termination still pending (i.e., no termination event has occurred)?

workflow id	event type	activity/service	origin	time
3245	Request	process_HIC	Customer	13.5./10:01
3245	Request	accept_case	Worklist-Meier	13.5./10:01
3246	Request	process_HIC	Customer	13.5./10:04
3246	Request	accept_case	Worklist-Hefti	13.5./10:14
3247	Request	process_HIC	Customer	13.5./10:16
3247	Request	accept_case	Worklist-Meier	13.5./10:26
3247	Reply	accept_case	Worklist-Meier	13.5./10:37
3247	Request	control_blacklist	Worklist-Meier	13.5./10:37
3247	Reply	control_blacklist	DB-App-RSA	13.5./10:49
3247	Request	print_denial	Worklist-Meier	13.5./10:51
3247	Reply	print_denial	Printer	13.5./10:52
3245	Reply	accept_case	Worklist-Meier	13.5./10:57
3245	Request	control_treatment	Worklist-Meier	13.5./10:57
3245	Request	control_blacklist	Worklist-Meier	13.5./10:57
3246	Reply	accept_case	Worklist-Hefti	13.5./10:59
3246	Request	print_check	Worklist-Hefti	13.5./11:00
3246	Except.(NOTAVAIL)	print_check	Printer	13.5/11:02
3245	Reply	control_blacklist	DB-App-RSA	13.5./11:04
3246	Request	print_check	Worklist-Hefti	13.5./11:55
3246	Reply	print_check	Printer	13.5./11:57
3245	Reply	control_treatment	Exp-Seehofer	14.5/08:43
3245	Request	print_check	Worklist-Meier	14.5./09:12
3245	Reply	print_check	Printer	14.5./09:13
3245	Reply	process_HIC	Worklist-Meier	14.5./09:13

Table 1. A sample event history

2. (a). How often did an instance of a specific workflow type execute. (b). How often did it terminate with a resulting termination event of a certain type?
3. What is the average/maximal/minimum execution duration of various workflow types?
4. What is the average/maximal/minimum execution duration of individual activities?
5. How often has a specific service (activity) been requested, or a subworkflow executed?
6. How often has a certain resource (PE) been involved in a workflow?
7. (a). What is the average/minimum/maximum delay time until a resource (processing entity) is available for the provision of a service? (b). Which resources become more often bottlenecks?
8. (a). How many and what types of exceptions have been raised during the execution of a specific workflow type? (b). Which requests cause more often exceptions?

Several of these criteria can be applied to all workflows, or to workflows of a specific type. The information obtained can typically be used in several ways:

- to find inadequate, incomplete, or incorrect specifications,
- to determine parts of workflow specifications that are worthwhile to be optimized, and
- to determine the existence and identify the location of bottlenecks.

4.2 Design of a Query Service for the Event History

The information that should be evaluated can be obtained by formulating and running queries on the event history created by the system. Some of the queries can be evaluated during the execution of a particular workflow to be analyzed, while others are only meaningful for postmortem evaluation. These queries can be categorized according to their purpose.

A first group of queries concerns the detection of semantic (specification) errors. The fact that workflows of a certain type often terminate with undesired results—representing a semantic failure (see query 2)—can indicate that the specification is incomplete, e.g., because conditions have been omitted. An overly high number of raised exceptions (query 8) occurring during the execution of a workflow type can indicate that exceptional situations are not addressed appropriately in the specification or that certain services are often not available in an adequate or timely manner.

The second group of queries helps to identify those workflows whose optimization is worth considering. For instance, if the answer to query 2 shows that a rather small number of instances of a certain workflows is executed (with respect to the total number of executed workflows), then optimizing this workflow type will probably not increase the overall throughput very much. The same conclusion can be drawn from query 3: if the average execution time is rather small, then optimizing the corresponding workflow specification might not improve overall system performance.

Bottleneck situations and inefficiencies may be identified by several queries (e.g., 5, 6, 7, and 8). Potential bottlenecks are processing entities, the resources they use for activity execution, as well as system infrastructure. Bottlenecks can be caused due to the following factors:

- high workloads such as many requests for time-consuming tasks as specified in the workflow,
- scarcity of resources leading to a limited choice of responsible and capable processing entities for given tasks,
- unavailability of system infrastructure (e.g., a network node on which the only capable broker for a specific service resides) leading to frequent exceptions and consequently new requests, and
- suboptimal task assignment strategies.

The presented queries can be combined to help the identification of service requests which cause more often an exception, which services are the most time consuming or have the largest variation in their execution time, and which processing entities are involved in the execution of such "problematic" services.

In order to support analysis, the WfMS has to support the extraction of the relevant information from the event history. For that matter, a set of base queries has been identified for EvE that is sufficient for this purpose. In a first step, these queries are specified in OQL. The next section shows how these queries are implemented.

Figure 4 shows a query that computes the times consumed by executions of an activity or workflow of a specific type (see query 4 above). The begin of the activity/workflow is given by the event named `W-request`, and its end is given by the event of type `W-reply`. This query computes execution times of the activity/workflow within executions of the workflow type WF. If execution times should be considered independent of the workflow within which they were executed, then the predicate in line 4 would be omitted.

If the average (minimum/maximum) execution time should be computed, then the average (mimimum/maximum) operators would be applied to the result of this query (i.e., avg, min, or max).

```
select  occ2->timestamp - occ1->timestamp
from    occ1 in eventhistory, occ2 in eventhistory
where   occ1->etype->name = W-request
        and occ1->workflow->wf_type->name = WF
        and occ2->replies_to = occ1
        and occ2->etype->name = W-reply
```

Fig. 4. A sample OQL-query against the event history

Further analysis of the queries shows that a set of "base" queries can be determined that occur in many complex queries used for analysis. For instance, the base query `searchWorkflow(W, E)` finds all event occurrences of `E` that have occurred in workflow executions of type `W`. These base queries can be combined and reused in a meaningful way in many different analysis scenaria. We used this fact to simplify the implementation of an interface which can be used for the analysis of workflows.

4.3 Implementation of the Query Service

Ideally, the storage system would provide for a query language such as OQL or SQL. However, object managers typically do not support queries, and among the object-oriented DBMSs only O_2 [2] fully implements OQL. We reclined to move to that platform due to, e.g., better extensibility in Shore [5]. Likewise, using a relational DBMS was not considered either, for similar reasons and in order to avoid the impedance mismatch between the B/SM and the storage platform.

Thus, the queries either had to be hard-wired in the analysis interface, or otherwise a full-blown query language would have to be implemented. Due to the fact that queries are always directed against the same extension (namely, the event history), we decided against the second alternative and pursued a mixed approach. The base queries mentioned above have been implemented as methods of the class `EventHistory`. All these queries produce as their result sets of event occurrences which can thus be easily combined using normal set operations (also implemented in the analysis interface). In order to improve response time for analysis queries, we use the indexing and clustering facilities of Shore.

Figure 4.3 shows the signatures of the base queries. Each query returns a set of event occurrences. As noted above, complex queries can be formulated, by applying on the results of the base queries the conjunction (`AndJoin`) or disjunction (`OrMerge`) operators.

```
interface EventHistory {
     ...
     set<EventOcc> SearchWorkflow(in lref<char> wfName,
                                  in lref<char> eType) const;
     set<EventOcc> SearchBetween(in long fromTime, in long toTime,
                                 in lref<char> eName) const;
     set<EventOcc> SearchBefore(in long fromTime,
                                in lref<char> eName) const;
     set<EventOcc> SearchAfter(in long untilTime,
                               in lref<char> eName) const;
     set<EventOcc> SearchReferring(in lref<char> eName,
                                   in ref<EventOcc> refTo) const;
     set<EventOcc> SearchBroker(in lref<char> bName,
                                in lref<char> eName) const;
     set<EventOcc> SearchSite(in lref<char> siteName,
                              in lref<char> eName) const;
     set<EventOcc> AndJoin(in set<EventOcc> set1,
                           in set<EventOcc> set2) const;
     set<EventOcc> OrMerge(in set<EventOcc> set1,
                           in set<EventOcc> set2) const;
}
```

Fig. 5. Signatures of Query Implementations

The provided base queries are used to implement any of the queries described in section 4.1. For example, query 1 (a) can be implemented by combining `SearchWorkflow()` and `SearchReferring()` to find out which of the initiated workflows (i.e., those for which the initiation event set is not empty) have successfully ended (i.e., also a termination event is logged in the event history). Similarly, `SearchBroker()` can be used in combination with `SearchWorkflow()` and `SearchReferring()` to determine which brokers are most often involved in an unsuccessful request (causing an exception).

The complex queries as well as computations and aggregation operations are implemented in the analysis application programs.

In order to prelimiarily test the performance of the analysis query interface, we have run several test queries on a synthetic event history (see Fig. 6). The tests were executed on a SUN Ultra 1 (128 MB main memory, 167 MHertz processor) under Solaris; results are given as wallclock time in seconds.

Query Q1 retrieves all occurrences of a certain type having occurred within a given time interval. For this query, a full scan of the event history is neccessary. Q2 retrieves all occurrences of a certain type that have been generated within an instance of the HIC-workflow (query `searchWorkflow`). Finally, Q3 retrieves all occurrences of a certain type that have been generated by a specific broker within an instance of the HIC-workflow (queries `searchWorkflow` and `searchBroker`, then combined by `AndJoin`). 100 different workflow types have been defined. The workflow instances (500, 1000, 2000, and 5000, respectively) have been distributed evenly among these five types. Each workflow instance generated an average of 8 event occurrences.

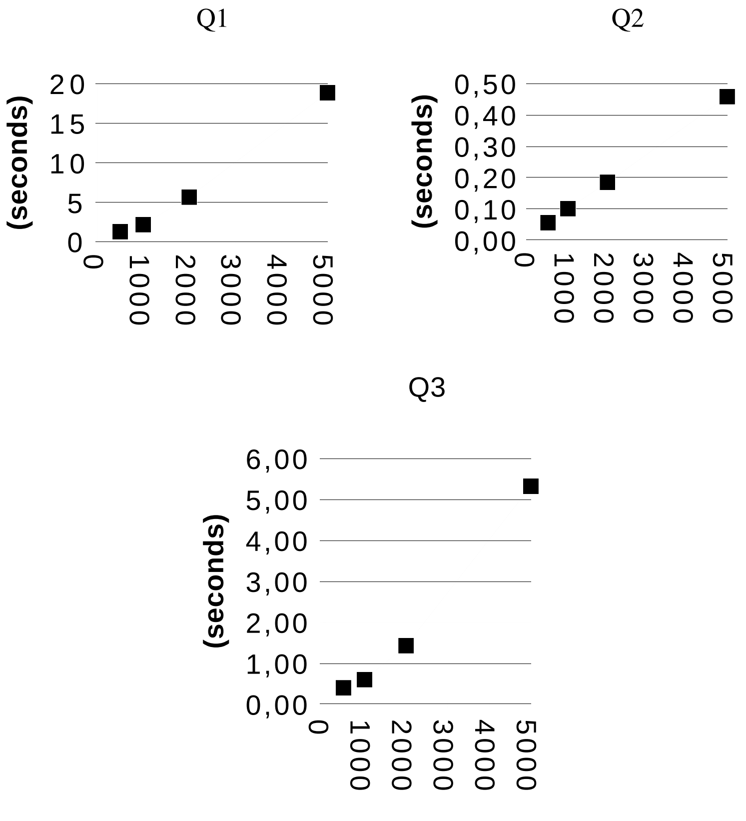

Fig. 6. Results of Analysis Query Performance Tests

Results are not satisfying when the whole event history has to be searched (as in Q1). If restrictions on workflow types, event types, and/or brokers are specified, then the query execution time can benefit from indexing. In addition, the rather high number of workflow types leads to more selective queries and therefore to smaller execution times (alternatively, if only 5 workflow types are defined but the same number of instances, then results are much worse especially when indexing is used). In our future work, we will investigate how to improve the performance of analysis queries.

In future experiments, we will compare these results with those obtained when using an OODBMS with a full-fledged query language. We will also test the performance of the event detectors using the Beast-benchmark [12].

4.4 Example

We conclude this section by showing through a sample scenario how workflow execution analysis helps to improve specifications and throughput.

Assume a workflow administrator has the impression that system throughput is rather low. He can check this by computing the average/minimal execution times of instances of the HIC-workflow (query 3.). He suspects that the activity `control_treatment` is responsible for delays. He can validate this hypothesis using queries 5. (for determining whether this activity has actually been executed frequently) and 4. (to determine how large the average execution time of the activity is). He might then discover that a large fraction of the execution of HIC-workflows is due to this activity. Finally, he uses query 6. for the people (insurance experts) that are responsible to execute the activity. The administrator then comes to the conclusion that the expert represented by the broker `Exp-Seehofer` is the bottleneck, because the activity is assigned very often to this resource (e.g., as this expert is the only one with certain capabilities such as expertise in specific treatments and diseases).

After this bottleneck has been discovered, the appropriate reaction would be to add one or more additional experts with the same capabilities, and to modify the task assignment such that the load is better balanced among these experts.

5 Conclusion

In this paper, we have shown how logging and analysis of workflow executions can be elegantly performed in a system that supports the notion of event history. Although logging and analysis are considered as important tasks of a workflow management system (e.g., [15]), and the Workflow Management Coalition has already drafted a standard for workflow logs [22], little work has been done in other event-driven approaches to workflow management. Non-event-driven commercial systems (e.g., Staffware [19]) usually provide rudimentary logging information in the form of an audit trail which is dumped to a file, whereas others such as COSA Workflow [18] store log records in a relational DBMS. To the best of our knowledge none provide sophisticated analysis tools.

In order to perform logging and analysis adequately in our context, several extensions and modifications have been necessary with respect to event histories maintained by ADBMSs:

– the event history must be made explicit, such that it can be queried,
– further event attributes have to be added for event occurrences (e.g., a reference to the component who generated the event, etc.),
– event occurrences should not be deleted from the history upon event consumption (this is similar to the model proposed in [9]).

From the point of view of active systems, our approach to workflow logging exploits synergies, since EvE as an active system has to maintain the event history anyway. Analysis is then possible by allowing queries against the event history. Further ways to use the knowledge in the history are possible; e.g., "replays" of single workflow executions based on the event sequence pertaining to the particular workflow. These tasks and further usages of the event history and the implementation of a graphical analysis tool are part of our future work.

Acknowledgements

We are indebted to our colleagues Klaus Dittrich and Markus Kradolfer for fruitful discussions and our students Patrick Brunschwig, Natalie Glaus, and Tom Haas for their great contributions to EvE. We also acknowledge the comments of the anonymous reviewers which helped us to improve the paper, as well as the funding of D. Tombros by the Swiss National Science Foundation in the context of the SPP-IuK project Swordies (no. 5003-045356).

References

1. ACT-NET Consortium. The Active Database Management System Manifesto: A Rulebase of ADBMS Features. *ACM SIGMOD Record*, 25(3), September 1996.
2. F. Bancilhon, C. Delobel, and P. Kanellakis, editors. *Building an Object-Oriented Database System. The Story of O2.* Morgan Kaufmann Publishers, 1992.
3. A. Buchmann, J. Zimmermann, J. Blakely, and D. Wells. Building an Integrated Active OODBMS: Requirements, Architecture, and Design Decisions. *Proc. 11^{th} ICDE*, Taipeh, Taiwan, March 1995.
4. C. Bussler and S. Jablonski. Implementing Agent Coordination for Workflow Management Systems Using Active Database Systems. *Proc. 4^{th} RIDE-ADS*, Houston, TX, February 1994.
5. M.J. Carey, D.J. DeWitt, M.J. Franklin, N.E. Hall, M.L. McAuliffe, J.F. Naughton, D.T. Schuh, M.H. Solomon, C.K. Tan, O.G. Tsatalos, S.J. White, and M.J. Zwilling. Shoring up Persistent Applications. *Proc. ACM-SIGMOD*, Minneapolis, May 1994.
6. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Deriving Active Rules for Workflow Management. *Proc. 7^{th} DEXA*, Zurich, Switzerland, September 1996.
7. S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. *Proc. 20^{th} VLDB*, Santiago, Chile, September 1994.
8. U. Dayal. Organizing Long-running Activities with Triggers and Transactions. *Proc. ACM-SIGMOD*, Atlantic City, NJ, May 1990.

9. A.A.A. Fernandes, M.H. Williams, and N.W. Paton. A Logic-Based Integration of Active and Deductive Databases. *New Generation Computing*, 1996.
10. S. Gatziu and K.R. Dittrich. Detecting Composite Events in an Active Database System Using Petri Nets. *Proc. 4^{th} RIDE-ADS*, Houston, TX, February 1994.
11. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, September 1994.
12. A. Geppert, S. Gatziu, and K. Dittrich. A Designer's Benchmark for Active Database Management Systems: OO7 Meets the Beast. In *Proc. 2^{nd} RIDS*, Athens, Greece, September 1995.
13. A. Geppert, M. Kradolfer, and D. Tombros. EvE, an Event Engine for Workflow Enactment in Distributed Environments. Technical Report 96.05, Department of Computer Science, University of Zurich, May 1996.
14. A. Geppert, M. Kradolfer, and D. Tombros. Realization of Cooperative Agents Using an Active Object-Oriented Database Management System. *Proc. 2^{nd} RIDS*, Athens, Greece, September 1995.
15. S. Jablonski and C. Bussler. *Workflow Management. Modeling Concepts, Architecture, and Implementation.* Intl. Thomson Computer Press, London, 1996.
16. G. Kappel, P. Lang, S. Rausch-Schott, and W. Retschitzegger. Workflow Management Based on Objects, Rules, and Roles. *Bulletin of the IEEE Technical Committee on Data Engineering*, 18(1), March 1995.
17. The Shore Project Group. *Shore Data Language Reference Manual. Version 1.0.* Computer Science Department, University of Wisconsin, Madison, August 1996.
18. Software-Ley. *COSA Workflow, Product Description, Version 2.0 (in German)*, 1996.
19. Staffware House. *Staffware Workflow, System Overview*, 1996.
20. D. Tombros, A. Geppert, and K.R. Dittrich. Design and Implementation of Process-Oriented Environments with Brokers and Services. In B. Freitag, C.B. Jones, C. Lengauer, and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*. Kluwer Academic Publishers, 1996.
21. D. Tombros, A. Geppert, and K.R. Dittrich. Semantics of Reactive Components in Event-Driven Workflow Execution. *Proc. 9^{th} CAiSE*, Barcelona, Spain, June 1997.
22. *Workflow Management Coalition Draft Audit Specification*, Document Number WFMC-TC-1015 (http://www.aiai.ed.ac.uk/WfMC/DOCS/if5/if59611.html), August 1996.

This article was processed using the LaTeX macro package with LLNCS style