Scientific Workflow Management by Database Management* †

Anastassia Ailamaki Yannis E. Ioannidis‡ Miron Livny
Department of Computer Sciences
University of Wisconsin
Madison, WI 53706
{natassa,yannis,miron}@cs.wisc.edu

Abstract

In several working environments, production involves repeated executions of certain procedures. A workflow *describes the individual tasks performed in these procedures and their interrelationships. Current Workflow Management Systems (WFMSs) use a Database Management System (DBMS) to store task descriptions, and implement all workflow functionality in modules that run on top of the DBMS. Motivated by scientific workflows, we propose a much more DBMS-centric architecture, in which conventional database technology provides much of the desired scientific WFMS functionality. A key element of our approach is viewing the workflow as a web of data objects interconnected with active links that carry process descriptions. The workflow is fully defined as a database schema, and its execution is the gradual buildup of an instance of this schema through the active object links. For our work, we use the modeling and querying tools of* `Horse`*, the object-oriented DBMS that we have developed in the context of the* `Zoo` *Desktop Experiment Management Environment.*

1. Introduction

Several procedures in many working environments are repeated over and over again. At an insurance company, every time a claim is filed, a standard procedure is followed for its evaluation; at a car rental agency, there is a sequence of steps followed when a customer asks for a car; and in laboratories, the same scientific experiment is executed when fresh input data are available. These procedures usually consist of a set of smaller tasks that represent self-contained units of work, which are naturally dependent to each other. The set of tasks involved in a procedure along with their interdependencies and their inputs and outputs is called a *workflow*. *Workflow management systems* (WFMSs) are used to define, execute, and monitor workflows.

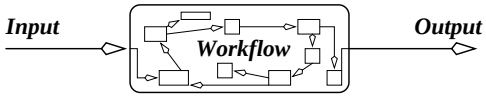

Figure 1. *Transactional view of workflows.*

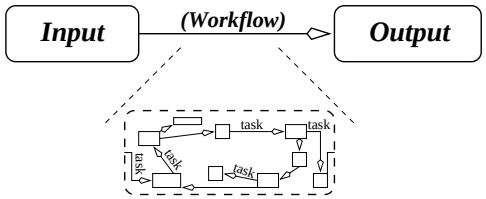

Figure 2. *Object view of workflows.*

In the typical conceptualization of workflows, the focal point is the action, i.e., the processes that take place during workflow execution (Figure 1). Workflows are considered as *transactions*, with the information that they manipulate playing a "subordinate" role – it is a side-effect, so to speak. This *transactional* view of workflows leads to (almost imposes) the architecture depicted in Figure 3a, which is the one followed by the majority of existing WFMSs (commercial systems or research prototypes). A dedicated workflow-specific software system runs *on top of* a Data Base Management System (DBMS), i.e., process management is separate from data management. The DBMS simply stores the

*Work supported in part by the National Science Foundation under Grants IRI-9224741 ("Scientific Databases Initiative"), IRI-9700799, and DBI-9604906.

†We would like to thank Martha Anderson for her invaluable help with the 'cranberry workflow', Dimitris Georgakopoulos and Thimios Panagos for providing us with many insights into workflows and many references to earlier work, and Andy Therber for implementing an external Condor application to be used in various workflows.

‡Additionally supported in part by the National Science Foundation under Grant IRI-9157368 (PYI Award) and by grants from DEC, IBM, HP, AT&T, Oracle, and Informix. Author's present address: Department of Informatics, University of Athens, Hellas (Greece).

information about the workflow tasks, while the next software layer uses that information to conduct the workflow execution.

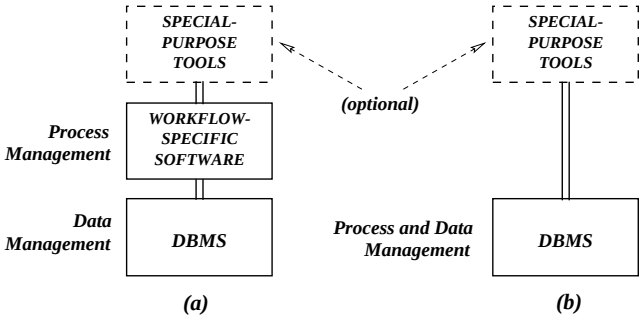

Figure 3. *Two WFMS architectures*

Motivated primarily by scientific applications, we propose a different conceptualization of workflows, where the focal point is the information, i.e., the data used and generated during workflow execution (Figure 2). Workflows are considered as graphs of *objects*, with the processes that created them being expressed through the links between them. This *object* view of workflows suggests the architecture depicted in Figure 3b. Provided that its data model can express the *active* workflow aspects, the DBMS orchestrates the workflow execution as a first-class WFMS.

In this paper, we essentially build a case for the object view of scientific workflows. We demonstrate that a DBMS is itself a WFMS, offering much of the needed functionality with respect to process management without additional software (Figure 3b). Moreover, we show that any critical functionality missing from conventional DBMSs can be easily provided with minimal, natural extensions that remain faithful to the philosophy of database technology. By providing all needed process management within the DBMS, we reap many benefits:

- *Reduced implementation effort:* For much of the needed workflow functionality, there is no need for implementing special-purpose 'workflow tools'.
- *Increased optimization opportunity:* The entire operation of a workflow is controlled from within the DBMS proper, so the optimizer has global knowledge of all database interactions.
- *Uniformity in workflow management:* From specification to execution and monitoring, all workflow functionality is exported to the user through one unified access language.
- *Immediate information availability:* 'Drill down' requests from the end result to all aspects of the workflow that generated it are direct database queries.

As a proof of concept for the object view, we describe how workflows have been captured in the `Horse` object-oriented DBMS, which we have built as part of the `Zoo` Desktop Experiment Management Environment [8][1]. The key characteristics of our overall approach are the following:

- A workflow is defined as an object-oriented database schema
- An instance of the workflow schema is created during execution
- Invocation of workflow processes is captured and triggered by active rules of a restricted form
- External applications implementing workflow processes are scheduled through updates to system catalogs
- Status and other kinds of information about running or finished workflow processes is obtained by database queries
- Information on the workflow data is obtained by database queries

The rest of the paper is organized as follows. In Section 2, we provide an example workflow to be used throughout the paper. In Section 3, we discuss the functionality required from a WFMS to support scientific workflows. In Section 4, we briefly describe the `Moose` data model and the `Fox` data definition and query language on which `Horse` is based. In Section 5, we discuss how `Horse` achieves the desired functionality and becomes a WFMS. In Section 6, we present the workflows operated in two scientific laboratories where we are actually applying our tools for workflow management, and in Section 7, we discuss some related work. Finally, in Section 8, we present a summary of our contributions and our future work plans.

2. A Workflow Example

The example depicted in Figure 4 is an actual scientific workflow that captures the operation of an experimental study in the Soil Sciences Department of the Univ. of Wisconsin [1]. The objective of the experiment is to produce daily forecasts of near-surface temperatures in cranberry bogs in Wisconsin. These forecasts give cranberry farmers advance warning of over-night frost conditions[2], so they can take action to protect their vines from frost damage.

1. Around noon each day, satellite and ground-based meteorological observations are processed in the Atmospheric Sciences Department of UW, generating a 24-hour weather forecast at several heights in the atmosphere for the whole United States;
2. This US forecast is fed into a Bog Forecast Extraction program that extracts forecasts for points that are 25 meters above specified cranberry bog locations;

[1] Although our main interest and emphasis is on scientific workflows, the results apply equally well to business workflows also.

[2] ...as is often the case in Wisconsin!

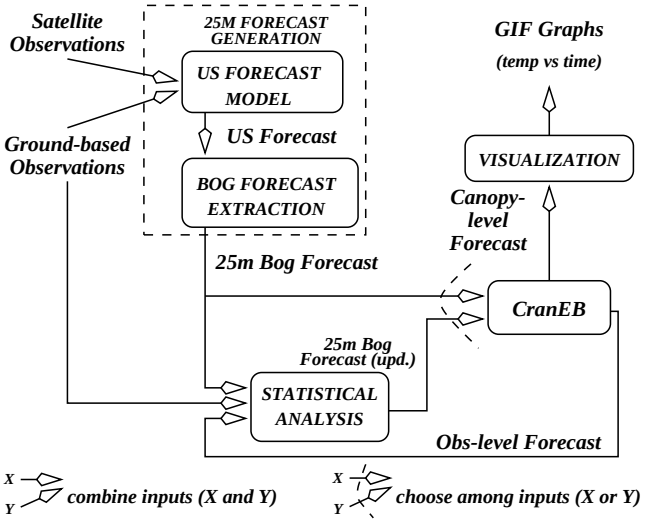

Figure 4. *The cranberry workflow*

3. These forecasts are sent to the Soil Science Department where they are processed by CranEB, to derive a forecast for the level of the cranberry vines (canopy level);
4. Later in the day, as new weather observations become available, the initial 25m bog forecast can be updated:
   - Scaled CranEB output forecasts are compared with new observed weather conditions in a package of statistical routines.
   - Appropriate corrections to the original 25m bog forecast are determined, and CranEB is rerun.

   With this feedback mechanism, the canopy-level forecast is updated continuously throughout the day.
5. Text files generated by CranEB are fed into the DEVise Visualization tool [9] that generate GIF plots of canopy temperature vs. time. These plots are then published on the Web, where they can be readily accessed by cranberry farmers throughout Wisconsin.

In the sequel, we refer to this example as the "cranberry workflow". It is used throughout the paper for reference.

3. Basic WFMS Functionality

In this section, we attempt a closer look at workflows and analyze the following basic functionality that a WFMS must definitely provide:

- workflow *specification*
- workflow *execution*
- workflow *evolution*
- workflow *auditing*

The above functionality is necessary for all types of applications, but is also sufficient for some as well, e.g., those that arise in single-user or mostly-read environments. Scientific laboratories (physical or virtual) tend to belong in these categories in general; since they are our main motivation for this work, our discussion focuses almost exclusively on the above functionality. Clearly, in multi-user and very dynamic environments, like those found in the business sector, there is additional workflow functionality that is necessary, including transaction management, workflow recovery, workflow interaction (for cooperative work), and others. We believe that the object view of workflows has many benefits for the support of this additional functionality as well, but demonstrating this remains part of our future work.

3.1. Workflow Specification

The specification of a workflow consists of three items:

- *Process*: This includes the workflow tasks and how they are related. There should be enough flexibility to allow various forms of task interrelationships:
  1. tasks operating in series or in parallel
  2. tasks receiving input from or providing (possibly distinct) input to multiple other tasks
  3. tasks choosing to receive input from among many possible tasks that provide it
  4. tasks receiving input directly or indirectly from themselves (feedback)
  5. tasks being abstracted and grouped into higher-level tasks

  Even in the simple cranberry workflow, we see the need for most of the above. For example, task Statistics Package accepts multiple inputs (1,2); task CranEB chooses its input between what task Statistics Package generates and what Bog Forecast Extraction generates (3), part of which is indirectly affected by what it produces (4); and 25m Forecast Generation abstracts a sequence of two tasks into one as the detailed steps are often (although not always) unimportant (5).
- *Data*: This includes the input and output data of the workflow tasks.
- *Invocation*: This includes the mechanism (rule) that triggers the execution of each task. There are two main choices: *explicit* invocation, in which a human initiates the task, and *implicit* invocation, in which the task begins immediately upon creation of its input as long as any specified conditions are satisfied. For example, in the cranberry workflow, one may choose some or even all of the tasks to be associated with implicit invocation, depending on how much automation is desired. (Currently, the entire process is completely automated.)

3.2. Workflow Execution

Execution of a workflow involves dealing with the three main elements of its specification as follows:

- The *process* typically involves execution of applications outside the WFMS. For example, task CranEB in the cranberry workflow is executed by *CranEB*, an external surface energy budget program. The WFMS must follow the logic of the workflow and at any point interact with the appropriate external system to transfer the control of execution as needed.
- The *data* may need translation during execution. For instance, the WFMS should be able to translate *25m Bog Forecast* data from the WFMS internal format to *CranEB*'s input format, and Canopy-level Forecast data from *CranEB*'s output format back to the WFMS internal format.
- The *Invocation* may be automatic or not, depending on the specification.

Even for a completely automated workflow, users retain ultimate control during its execution, monitoring and even influencing its operation:

- *Status monitoring*: At any point users may ask for the execution status of the entire workflow or parts of it. Status information may be provided by the WFMS or any external system used in the workflow. Such information does not obey to any universal format, but is very important because it users in deciding if and how they should intervene to the execution. For example, an inquisition on the status of the cranberry workflow may reveal that the incoming weather observations are garbled, which could lead in the (temporary) deactivation of the feedback loop through the Statistical Analysis task and the use of an earlier 25m bog forecast as input to the CranEB task.

- *VCR functionality*: User intervention in the execution of a workflow is reminiscent to the functionality of a VCR: the user can `stop` execution, `pause` execution and `resume` soon afterwards, or `rewind` execution up to a certain point and `resume` from there.

3.3. Workflow Evolution

Changes in a workflow may be an every-day routine in a working environment. Such changes are of three types:

- *Modification*: new workflow has same objective but different logic and replaces old one.
- *Versioning*: as before but new workflow does not replace old one, but co-exists with it.
- *Extension*: new workflow has different objective and therefore additional logic and replaces old one.

In addition, some environments require dynamic rather than static workflow evolution, i.e., changing one part of the workflow while another part is running.

3.4. Workflow Auditing

Workflow executions are related to several pieces and kinds of information, including the original input and the final output data, the results of intermediate tasks, and the interim and final status of the WFMS and the relevant external systems. Users are constantly auditing workflows by accessing and exploring all this information, analyzing the workflow results, obtaining reports on efficiency, validating the used process models, etc. For scientific workflows, this is often the primary time when 'science is done'.

4. The Moose Data Model and the Fox Query Language

As a vehicle to demonstrate the power of our approach, we use the `Horse` object-oriented DBMS that we are developing as part of the `Zoo` desktop experiment management environment. `Horse` is based on the `Moose` [3] object-oriented data model and the `Fox` [4] query language. Understanding the rest of the paper requires some familiarity with `Moose` and `Fox`, so their most important features are described below. More details about `Moose` and `Fox` can be found elsewhere [8, 14].

4.1. Moose

There are various *kinds* of object classes in `Moose` (tuple, collection or primitive). Objects from these classes are connected via binary relationships, three of which are relevant to this paper. The structure of a tuple class is defined by an arbitrary number of *has-part* relationships, each pointing to a single object. *Association* relationships connect individual objects in two classes of any kind. An *is-a* relationship between two classes has the usual meaning. All relationships are bidirectional, i.e., they can be traversed in either direction.

Any relationship between two classes C_1 and C_2 may be specified as *derived* from C_1 to C_2 or from C_2 to C_1 or both. In the first case, for each C_1 object, the related C_2 object is constructed or identified based on objects that are (indirectly) connected to the C_1 object via other relationships (similarly for the other cases). The construction or identification is through a *rule*, which may be any `Fox` command that returns an object. This includes the **exec** command, which invokes an external system that, in this case, receives as input a file containing (parts of) these other objects. The semantics of a derivation rule from class C_1 to class C_2 is that it is invoked every time an object is inserted in C_1, and whatever C_2 object it produces (if any) is placed in the C_1

[3] *M*odeling *O*bjects *O*f *S*cientific *E*nvironments
[4] *F*inding *O*bjects of e*X*periments

object as its value for the corresponding relationship and vice versa. Thus, derivation rules in Moose are a rather restricted form of *triggers*[4], in the sense that the only event that can trigger a rule is an insertion in a specific class.

The data definition language of Moose provides statements to create, destroy, and rename classes and relationships, associate rules to relationships, deactivate rules (temporarily modifying the schema as if the rule does not exist), and reactivate them back. Rule deactivation holds only for the current user session and does not affect other users of the schema.

4.2. Fox

Fox is the declarative query and data manipulation language of Moose. One may refer to an object in a Fox command by its unique object id (assigned by the system), its *name* (optionally assigned by the user), or the universal keyword **this** in cases an object is uniquely identified by some context in which the Fox command is nested. In addition, one may specify an object variable in a Fox query and bind it to members of a class extent or a collections defined by a path expression. Path expressions in Fox are used to navigate through interrelated classes. A path expression starts from a known object specification (constant or bound variable) and follows relationships from that object. The basic structure of a Fox query is derived directly from SQL:

for <range-binding-list>
select <projection-list>
where <qualification>
as <name>;

The **for** clause (optional) defines a set of objects using object variables or constants, or both. The **select** clause defines projections, as in SQL, which could be path expressions. The **where** clause (optional) involves a condition that defines the selection among the results. The **as** clause (optional) specifies a name for future reference to the query results.

There are five data modification statements in Fox: **insert**, **delete**, **update**, **load**, and **exec**. The middle three are not used in this paper, so they are not described any further (**load** is for bulk insertion of data from a file, generally into multiple classes at once). The **insert** command generates new objects for a class with values for their relationships specified either directly in a list or as the result of a Fox query, as in SQL. The **exec** command schedules the execution of a program ('agent') external to Horse. Its arguments are the name of the program and a list of path expressions, which form the program input. Examples of Fox commands are given in the following sections in the context of describing how workflows are captured within Horse.

5. A Database Way to Workflow Functionality

In this section, we describe how we capture in Moose, Fox, and Horse all aspects of WFMS functionality presented earlier (Section 3), i.e., workflow specification, execution, evolution, and auditing.

5.1. Workflow Specification

One of the most important characteristics of our approach is that workflows are directly represented as database schemas. This offers tremendous flexibility and makes many other aspects of the desired functionality fall out for free. The essence of the workflow-to-schema mapping is as follows:

- tasks, input data, and output data are all represented as ordinary Moose classes;
- task interconnections are represented by ordinary Moose relationships; and
- task invocation is expressed by assigning rules on the appropriate relationships.

The details of this mapping with respect to workflow data, process, and invocation are presented separately below.

Simple task and data: A workflow task is a process that operates on some input and produces some output. Depending on which aspects of the task one wants to capture, there are different schemas that can be used. The input and output data are always represented as Moose classes in a straightforward way. In some cases, the process itself is also of interest, e.g., to store information about the duration of each execution. Then, the corresponding task is represented in the schema as a Moose class as well, connected via has-part relationships with the corresponding input and output classes. In other cases, the task presents no interest, so it does not appear in the generated schema. Then, the corresponding input and output classes are connected with an association relationship.

The first row (row 0) of Figure 5 shows a single task with its associated data and a schema that can represent it for the first of the two cases discussed above (when the process is of interest). The other case is similar: the explicit task class is missing and the input and output classes are connected via an association. This holds for the remaining rows of the figure, where for each of the workflows in the left column and the corresponding schema that captures the workflow tasks (second column) are shown. In the sequel, workflows and schemas in row N are referred to as Figure 5(N).

The action that produces the task's output is specified by a *derivation rule* associated with the appropriate relationship of the output class. This is either the has-part relationship connected to the task class or the association connected to the input class. In Figure 5, this is indicated by a D(<input-class>) label on the appropriate relationship and

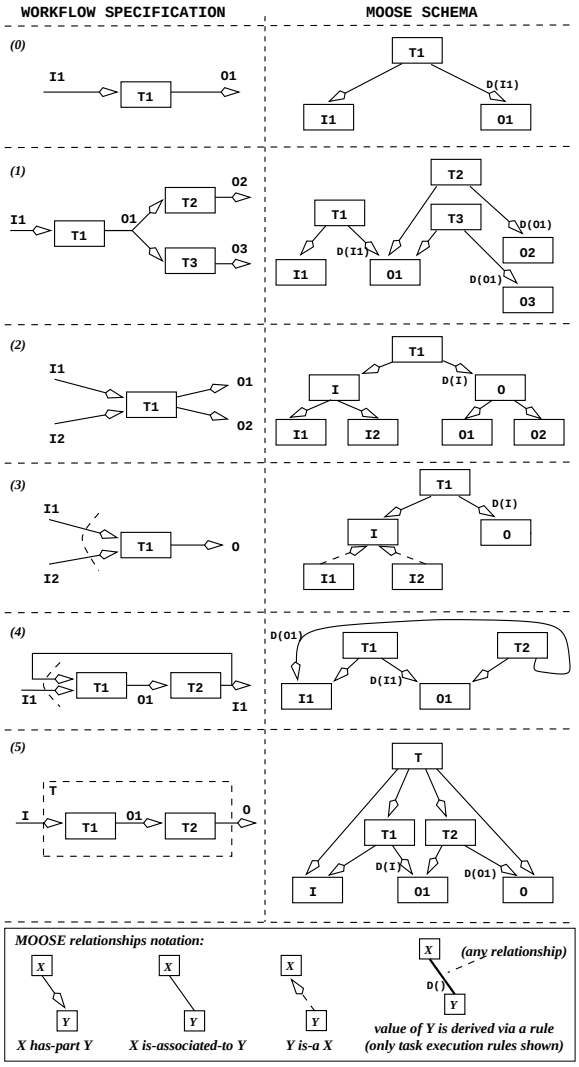

Figure 5. *Task interconnections in the* Moose *model*

close to the output class. Assume that the task T1 of Figure 5(0) is implemented by an external application aProgram on its input. The corresponding rule in Fox is

exec aProgram**(this**.I1**)**

When the task is invoked, the system should send the appropriate I1 object (possibly with its parts) to the external application to generate the output.

Task interrelationships: Figure 5 concentrates on a set of 'atomic' workflows (i.e., workflows of the simplest form), which capture five possible task interrelationships. The figure shows one natural and consistent way to represent each workflow with a Moose schema (but this is not forced upon the designer). By combining the appropriate schemas shown, one can construct others that represent arbitrarily complex workflows (like the cranberry workflow).

The workflow in Figure 5(1) has tasks T1 and T2 operating in series (similarly with tasks T1 and T3), and tasks T2 and T3 operating in parallel. The corresponding schema is straightforward from the associated schema for the simple-task workflow (Figure 5(0)) and needs no further explanation.

The workflow in Figure 5(2) has task T1 receiving multiple inputs and providing multiple outputs. The basic schema is now enhanced with two auxiliary classes, I and O, which act as input and output concentrators, respectively.

The workflow in Figure 5(3) has task T1 receiving two *alternative* inputs, of which it uses one each time. Assuming the general case that the two inputs are of different type, this is modeled through inheritance, by making the classes capturing the two input types subclasses of an auxiliary general input class I. If the alternative inputs are of the same type, no inheritance is necessary.

The workflow in Figure 5(4) has task T1 receiving input indirectly from itself. The schema corresponding to such a feedback cycle are direct derivatives of the corresponding single-task schema. For simplicity, we have assumed in the figure that the output of task T2 is of the same type as the input of T1, so inheritance does not appear in the schema.

The workflow in Figure 5(5) has a simple workflow (series of task T1 followed by task T2) abstracted and grouped into a higher level task T. This is simply modeled by connecting the classes of the input of the entire series to its output, either through an explicit task class T (shown in Figure 5) that has the individual task classes and the overall input and output classes as parts, or through a direct association. In the schema shown, T.I and T.O are derived relationships whose rules essentially retrieve the path expressions T.T1.I and T.T2.O, respectively. These rules are not indicated in 5, to bring out the rules that capture task execution.

From the above exposition, it should be clear that workflows of arbitrary complexity can be captured in ordinary Moose schemas without using any special constructs, almost effortlessly. For example, workflows that require constructs like *if-then-else* are captured as parallel tasks (Figure 5(1)). The if-condition is expressed in the qualification of the derivation rule of the then-task, while the complementary condition is in the qualification of the derivation rule of the else-task. Likewise, workflows that require programming constructs like *for* and *while* are captured with loops in the workflow schema (Figure 5(4)). The desired condition of the construct is expressed in the qualification of the corresponding derivation rule that initiates the loop, e.g., the rule that inserts an object in task T1 (Figure 5(4)). As long as the condition is satisfied, the rule fires and the loop continues; the first time the condition is not satisfied, the loop stops.

Moreover, all desired information on workflow executions is uniformly and centrally captured in a schema, including the data manipulated, details of the execution itself, the exact steps followed by the workflow and the mechanism used to invoke each one, etc. The schema becomes the *formal document* describing every aspect of the workflow. The explorability of workflow history thus afforded is one of the greatest benefits of our approach that make the implementation of workflow management inside a DBMS very attractive.

Invocation: Consider the simple example of Figure 5(0). In the corresponding schema, by the semantics of derivation rules (Section 4), the task is invoked as soon as an object is inserted in class T through the triggering of the rule in the relationship from class T to class O. An additional rule in the relationship from class I to class T of the form

insert into T**() instance()**

captures an *implicit* invocation of the task. As soon as an object is inserted in I, the above rule inserts an object in T, which in turn fires off the task. On the other hand, the absence of such (i.e., if the relationship is not derived) captures an *explicit* invocation of the task. Task execution begins only when a human (or some application program) explicitly inserts an object in T and connects it to the appropriate I object.

Clearly, the above can be generalized to arbitrary schemas, and the decision about implicit or explicit invocation can be made independently for each task by defining or not the appropriate derivation rules. Again, the ease of capturing such behavior through purely database means is clear.

As an example of the entire methodology described above, Figure 6 shows a schema for the cranberry workflow. For simplicity we do not show any of the actual rules or their invocation mechanism; we simply indicate the existence of rules by using their names as labels and again this only for those that capture task execution. Note that tasks US Forecast Model and CranEB have been captured with explicit classes due to their importance, while the remaining ones have not, as the workflow designer from the Soil Sciences Department expressed no interest in incorporating in her own schema any information on their execution.

5.2. Workflow Execution

Given a schema that captures a workflow as described above, executing it becomes almost trivial, as it reduces to simple database insertions; the rules do the rest! As soon as objects for the initial workflow inputs are inserted into the appropriate classes, execution starts immediately (if the first tasks are associated with implicit invocation), or after a human inserts task objects in the appropriate classes (if they are associated with explicit invocation). This continues throughout a workflow execution with explicit or implicit insertions causing further task firing.

Workflow monitoring is also accomplished by Fox queries, featuring a novel path-expression connector. In particular, the traditional connector '.' indicates moving along relationships of a specific kind from a given (or retrieved) object to those related to it. In the presence of derived rules, such relationships may be 'under construction'. Querying about the execution status of such a construction task is essentially querying about the status of the corresponding derived relationship. We take advantage of this mapping between tasks and relationships and introduce a novel connector, '?', which can be used anywhere appropriate in path expressions. Informally, the new connector indicates retrieval of the status information generated by the (external) system processing the corresponding task. This information is stored as an object in a possibly independent database created by the individual system for that purpose, which neither itself nor its schema are necessarily known to the workflow user.

The connector also indicates normal traversal of the object relationships if they have already been constructed, so that '?' connectors further down the path expression may be correctly interpreted as well. If the '?' connector is placed on relationships that are not derived or do not involve external execution, it is equivalent to the '.' connector.

For example, consider the cranberry workflow schema of Figure 6. Assume that one is interested in the status of the US Forecast Model execution (rule R_{FM}) initiated by the most recent weather data, i.e., the most recently added object in 'Weather Observations'. Assume that this object has been named 'now'. Obtaining the status information of interest is simply achieved by the query

select now.UsForecastModel?UsForecast

Its result is an object of whatever the status schema maintained for executions of the US Forecast Model task happens to be. Likewise, if one is interested in the status of the entire workflow initiated by 'now' (excluding any visualizations or feedback), the appropriate query would be

select now.UsForecastModel?UsForecast?
25mBogForecast.CranEB?CranEBForecast

Note that the query contains a '?' connector for relationships derived through workflow tasks, and the regular '.' connector for all others. Its result includes anywhere from zero to three objects, depending on how many of the corresponding tasks have been initiated. For example, if the query is posed while the Bog Forecast Extraction is running (rule R_{BFE}), two objects will be retrieved: the final status object for the execution of US Forecast Model (rule R_{FM}) and the current status object for Bog Forecast Ex-

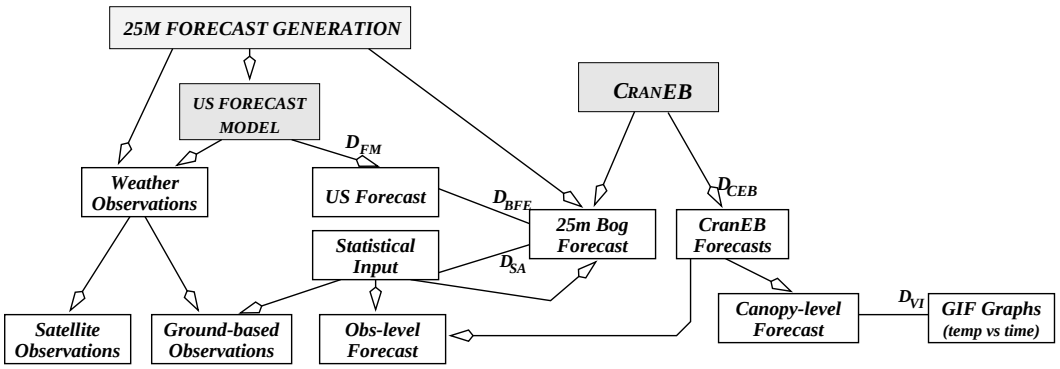

Figure 6. *A schema for the cranberry workflow*

traction.

Finally, any form of VCR functionality corresponds to simple database manipulation. A `stop` or a `pause` on an execution corresponds to deactivating a set of derivation rules. A `resume` corresponds to activating these rules back. A `rewind` up to a certain task followed by a `resume` corresponds to an insertion of a new object in the relevant task class that has the same input object as before. Overwriting the old data is simply a set of delete commands.

5.3. Workflow Evolution

By representing workflows as schemas, all flavors of workflow evolution reduce to schema evolution and are thus obtained for free. Workflow modification and extension reduce to changing the schema, while workflow versioning reduces to obtaining versions of the schema, both operations being well studied and understood in the database world. Moreover, schema evolution does not require isolation of a database from its users (except maybe for the portion that is being evolved), so workflow execution can be done both statically and dynamically.

5.4. Workflow Auditing

All information relevant to a workflow and its executions are stored in a database populating the workflow schema, and can thus be accessed by queries. A workflow execution can be viewed as a 'web' that holds all the information produced on the way from the input to the output. For example, assume that the entire cranberry workflow is abstracted in a single task that accepts a set of weather observations files as input and produces a set of graphs as output (Figure 7). These graphs are naturally connected to all the objects containing execution information for the individual tasks that make this happen. With a single query about a specific output graph, all the information that is connected to the corresponding object is brought along to the user. In addition, as schemas are objects populating a meta-schema (of a schema database), any information about the workflows themselves is obtained through queries as well. Clearly, our schema representation of a workflow makes auditing fall squarely into database technology.

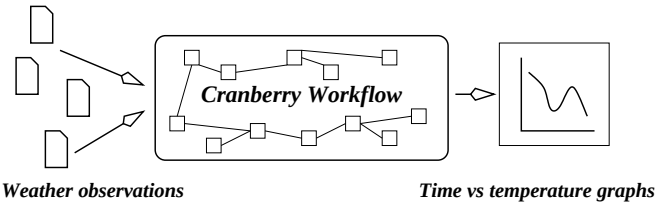

Figure 7. *Cranberry workflow abstraction*

6. System Status and Customized Installations

Over the past few years, we have been implementing the `Zoo` system [8] , which has several features that are geared towards supporting the WFMS functionality discussed in Section 3. Currently, all aspects of specification, execution, and auditing are operational, including invoking external applications and status monitoring. Workflow evolution has been designed, but implementation has only just begun. In addition to the cranberry workflow, we currently have two experimental installations of the system, one at the Soil Sciences Department and one at the *National Magnetic Resonance Facility At Madison (NMRFAM)*, hosted by the Biochemistry Department of the University of Wisconsin-Madison.

The `Zoo` installation at the Soil Sciences Department runs an experiment that predicts watershed response to rainfall, runoff, and sediment delivery at an area of interest, given a specification of vegetation and soil properties. The `Zoo` installation at the NMRFAM runs an experiment that uses a powerful spectrometer, and through further processing of the resulting NMR data by several software packages,

elucidates the three-dimensional structure of a biomolecule. The overall experience of the scientists of both groups with various tests of the system has been very encouraging.

7. Related work

Over the past few years, workflows have been a favorite topic in both the commercial and the research worlds. Since workflow management involves a very broad area of issues, the systems that have resulted from all this activity present a considerable diversity in their goals and approaches, making it often difficult to compare our work with the entire field. The main, essentially universal, difference of our approach with existing workflow systems is in the architecture (Figure 3). Independent of their goals, primary applications, and workflow type, the common denominator of all these systems appears to be the use of a DBMS (or other storage manager or file system) as a data repository, on top of which one or more software modules implement the desired workflow functionality. To the contrary, we use the DBMS itself for all workflow activities, reaping the benefits of the database technology maturity and obtaining much of the desired functionality for free.

A similar approach is taken by HiPAC [11], an active DBMS that uses database rules to trigger database operations as well as external applications. HiPAC offers a more complete rule system than ZOO, which includes only rules that seem to be needed to run workflow tasks (as is the case with almost all of the active DBMSs). We use this technology to design workflows and other workflow necessities as well, such as monitoring, VCR functionality, and dynamic evolution.

The state-of-the-art in the workflow area is determined by commercial products [6], whose goals, however, are very different from ours and include cooperative work, task routing, and data sharing in business environments. Probably reacting to the predominant lack of commercial attention to issues like scalability, reliability, concurrency control, and recovery [7], most research efforts have focused on interoperability, transaction management and high availability for business workflows, none of which is again among our interests. Characteristic is the fact that interoperability is the main goal of the Workflow Management Coalition [7], a standardization bureau that provides a generic reference model for workflows.

In general, existing commercial and research systems offer most of the functionality outlined in Section 3. There are two critical capabilities, however, that are missing from these systems (with few exceptions): invocation of ad-hoc external software and dynamic execution monitoring and reporting. There do exist some systems that offer interaction with specific office applications (specific vendor, operating system, and platform), but the ability to deal with ad-hoc systems is largely not offered. Likewise, existing systems do permit the retrieval of statistics about the workflow execution, but these are of a fixed, predefined form whose collection is intertwined with the workflow execution in a predetermined fashion. The ability that we offer to users to ask at any point for status information that may be defined independently by arbitrary external systems or for the workflow results and any information related to their creation is not there.

Scientific workflows have been explicitly addressed mainly by two projects: *WASA* [12] and *LabFlow-1/LabBase* [2]. But again, the goals and approach of these efforts have been different from ours. *WASA* uses a commercial WFMS on top of a DBMS extended by advanced features, and several user-interface, decision support, and analysis tools that offer a useful front-end to scientific workflow management. *WASA* offers most of the functionality described in Section 3, although possibly restricted by the capabilities of the underlying WFMS [13]. *LabBase* is a DBMS specialized for use by genome laboratories, and *LabFlow-1* is a database benchmark that tests the usability of storage managers to serve as a basis for WFMS development on top of them. Both of these systems fall beyond the scope of our work, and in general we are aware of no other system uses a DBMS to provide run-time workflow management.

Finally, there are a few efforts in the scientific database area that have similarities with some aspects of our work, although they have not been directly addressing workflow issues. The OPM effort at LBNL is probably the closest to this effort [3]. OPM is prominent in the genome database community and is used to implement some of the most important international genome databases. It has the same philosophy as ours in that workflows are represented as schemas in the OPM data model, which offers two kinds of classes, one for data and one for protocols, to facilitate the definition of experimental processes. However, protocol classes do not capture any active aspects of the corresponding workflow tasks, but are simply data containers, indicators of executions of the corresponding tasks. The workflow execution is not driven by OPM schemas but externally, and the resulting data are then stored under OPM. This is the key difference between OPM and Zoo. In essence, the current OPM-centered tools approach workflows based on the traditional architecture of Figure 3a, whereas by using the architecture of Figure 3b, we are able to provide run-time support to workflows through the DBMS itself. With respect to data models, the Extended Entity-Relationship (EER) model has also been enhanced with several features and used to model processes [10], again with no active features captured by the schema. Finally, with respect to invocations of external systems, *computational proxies* [5] have been proposed for interaction between a scientific DBMS and external chem-

ical models. This is very similar to how the corresponding `Zoo` module operates to achieve the same goal. One of the differences is our use of a generic translation tool in reading declarative specifications to translated between database object structures and external formats.

8. Summary and Future Work

In this paper, we have introduced the object view of workflows and have demonstrated that much of the needed workflow functionality can be supported within a DBMS, through regular database operations, with no need for developing specialized workflow software. The key enabling element of our database-centric approach is the use of a DBMS whose data model can express the active aspects of workflows as well (like `Moose`). This allows the DBMS to have control over workflow executions, and therefore to provide complete run-time support to workflow management: interacting with external systems implementing workflow tasks, obtaining their status during their execution, modifying a workflow during its execution, optimizing workflow tasks as globally as possible, etc. Moreover, the object-oriented schema representation of a workflow provides an integrated view of all workflow-related information that captures in a natural way the connection between the workflow process and the data it manipulates, thus permitting several important types of queries and analysis of workflow execution. Our implementation of most of the desired functionality in the `Horse` DBMS and our experimental installations in two scientific laboratories indicate that the object-view of workflows has many benefits and can serve the needs of several environments well.

The main goal of our future work is to demonstrate that the object view for workflows and the resulting DBMS-as-a-WFMS architecture can provide the remaining workflow functionality that we have not addressed in this paper, e.g., transaction management, workflow recovery, and workflow interaction. We believe this is indeed the case, and a possible success in this endeavor will be very important and beneficial to several business environments. Other future tasks include completion of the implementation of `Horse` with respect to the features necessary for workflows, investigation of additional forms of derivation rules (triggers) and their potential benefits for workflow management, and developing a visual user interface suitable for designing workflows at a level higher than the schema.

References

[1] M. Anderson, February 1997. Soil Sciences Department, University of Wisconsin, Madison (personal communication).

[2] A. Bonner, A. Shrufi, and S. Rozen. LabFlow-1: a database benchmark for high-throughput workflow management. In *Proc. Fifth International Conference on Extending Database Technology*, pages 463–478, Avignon, France, March 1996.

[3] I. Chen and V. Markowitz. The Object-Protocol Model: Design, implementation, and scientific applications. *ACM Transactions on Information Systems*, 20(5), 1995.

[4] A.-N. Consortium. The active database management system manifesto. *SIGMOD Record*, 25(3), September 1996.

[5] J. Cushing, D. Maier, M. Rao, D. Abel, A. Feller, and M. DeVaney. Computational proxies: Modeling scientific applications in object databases. In *Proc. Scientific and Statistical Database Management*, 1994.

[6] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.

[7] D. Hollinsworth. The workflow reference model. Technical Report TC00-1003, Workflow Management Coalition, Avenue Marcel Thiry 204, 1200 Brussels, Belgium, December 1994.

[8] Y. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti. ZOO: A desktop experiment management environment. In *Proc. 22nd International VLDB Conference*, pages 274–285, Bombay, India, September 1996.

[9] M. Livny et al. Devise: Integrated querying and visualization of large datasets. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 301–312, Tuscon, AZ, May 1997.

[10] V. Markowitz. Representing processes in the extended entity-relationship model. In *Proc. IEEE International Conference on Data Engineering*, 1990.

[11] D. McCarthy and U. Dayal. The architecture of an active data base management system. In *ACM SIGMOD International Conference on Management of Data*, pages 215–224, Portland, Oregon, June 1989.

[12] C. Medeiros, G. Vossen, and M. Weske. WASA: A workflow-based architecture to support scientific database applications (extended abstract). In N. Revell and A. Tjoa, editors, *Proc. 6th DEXA Conference*, pages 574–583, London, England, 1995. Springer LNCS 978.

[13] J. Meidanis, G. Vossen, and M. Weske. Using workflow management in DNA sequencing. In *Proc. 1st International Conference on Cooperative Information Systems (IFCIS)*, pages 114–123, Brussels, Belgium, June 1996.

[14] J. Wiener and Y. Ioannidis. A Moose and a Fox can aid scientists with data management problems. In *Proc. 4th International Workshop on Database Programming Languages*, pages 376–398, New York, NY, August 1993.