Exp-WF: Workflow Support for Laboratory Information Systems

Brian Gabor Bettina Kemme
School of Computer Science
McGill University
Montreal, Canada
email: {bgabor,kemme}@cs.mcgill.ca

Abstract

As scientific experiments and their analysis become increasingly automated, the need for workflow management support rises. Many workflow management systems tailored to scientific workflows have been developed. However, they are mainly stand-alone systems and ignore that many scientific research groups already store and manage experimental data in laboratory information systems, called LIMS. LIMS are typically web-based and allow researchers to insert and view experiment related data. In this paper we describe Exp-WF, a workflow management module that is designed to be incorporated into a typical LIMS such that workflow management and the traditional functionality of a LIMS co-exist. Scientists describe the execution order of experiments as a workflow model. Exp-WF then automatically selects experiments for execution and dispatches them to the people or machines responsible for performing them. Exp-WF extends existing workflow models to be able to handle the particularities of scientific experiment workflows. Exp-WF can be incorporated into an existing web-based LIMS in a nonintrusive way by using servlet filter technology to observe user actions and act upon them. Exp-WF uses agent-based technology with asynchronous communication to dispatch tasks to remote machines.

1. Introduction

The increasing prevalence of automated experimental techniques has allowed research scientists to dramatically increase the amount of data processed over the course of a typical experiment [2]. Such large volumes of data must be reliably stored, and processed for monitoring and data mining purposes. Therefore, many research groups have developed information systems that keep track of the experiments conducted, record which ingredients and instruments are used, and any observations and results achieved [8, 13, 5, 25, 10]. These systems are often referred to as *laboratory information management systems (LIMS)*. LIMS often follow a 3-tier architecture, where users view and modify data via web-based forms. Some also allow programs to access the LIMS, e.g., via a web-service interface. The application logic in the middle-tier receives and processes the user requests, and accesses the appropriate data in the database. The design of current LIMS reflect the need of constant interaction between users (the scientists) and the data management system.

However, current LIMS, especially those of smaller research groups, do not adequately support the current automatization in scientific research. In the wet lab, robots are able to run hundreds of experiments in the same time a human technician is able to run one or two. The large amounts of data produced in the wet lab must then be fed into programs for detailed analysis. These programs are often compute intensive and require adequate computing resources. In order to support such automated execution, a *workflow management system (WFMS)* [9] is needed. In our context, a *workflow* describes the set of tasks to be performed within a complex experiment suite, the order in which these tasks are conducted, and the data that flows from one task to the next. A WFMS must provide a language to define workflows, and a workflow engine that starts workflow instances, and then automatically dispatches tasks in proper order to the people, machines or programs that are responsible for performing them.

Many WFMSs tailored to scientific workflow management have been proposed (e.g. [2, 3, 11, 14, 17, 30, 21]). However, many of these systems are tailored to specific research areas. Furthermore, most of these tools are stand-alone systems, provide their own interfaces and cannot be easily integrated with an existing LIMS.

This paper presents Exp-WF, a workflow management module designed to address these concerns. It makes several contributions. First of all, it demonstrates an approach by which workflow management capabilities can be added to an existing 3-tier LIMS with minimal impact on the components of the original system. Exp-WF has been integrated

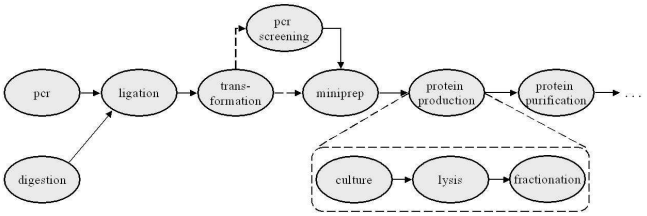

Figure 1. Part of a protein creation workflow.

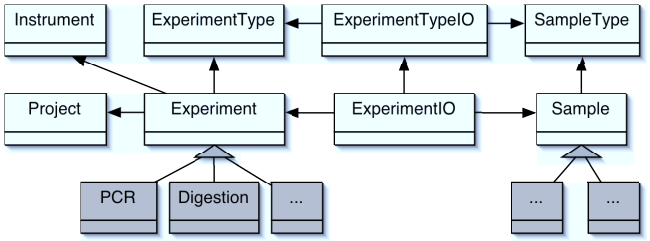

Figure 2. Data model for Exp-DB

with Exp-DB, a lightweight, extensible LIMS [25, 20]. Exp-WF uses servlet filter technology, as provided by many web containers, to intercept interactions between the web-based user and components of the original LIMS. This allows Exp-WF to respond to these requests transparently. At the same time, it is able to maintain the interactive nature in which LIMS are used. Furthermore, Exp-WF allows for a flexible integration of external systems (e.g. robots performing experiments, computing resources running analysis tasks) by providing a software agent framework. We are confident that other web-based LIMS applications could be augmented with Exp-WF in a similar fashion. Additionally to this integration, Exp-WF extends traditional workflow models to be able to model more advanced experiment execution patterns, as they are often found in scientific research. This allows Exp-WF to support a more complex array of scientific workflows.

2. Workflow Example

Figure 1 shows part of a workflow for protein creation which we use as an example throughout the paper. Protein creation is needed, e.g., in the context of protein structure prediction. The nodes represent experiments to be performed. We consider a rather wide definition of experiment. It can, for instance, be an experiment in the wet lab (which can be performed by a robot or by a human), or be the execution of a program (e.g., BLAST). The execution order is defined by the directed edges between experiments. The flow of data between experiments must also follow the arrows. Dashed arrows indicate *conditional* routing, meaning that some condition must hold after the source experiment completes in order for the destination experiment to be executed. Solid arrows mean that the succeeding experiment must be executed unconditionally. Finally, workflows may be nested within other workflows, as in the case of the *protein production* experiment which internally consists of more refined experiments.

3. Exp-DB

We use Exp-DB as an example LIMS [25, 20] Exp-DB has a typical three-tier architecture. Users (client tier) interact with the system through a standard browser. The middle tier is responsible for the presentation logic and the application semantics. The runtime environment is an Apache Tomcat web server [4] using Java Technology [15]. The backend tier can be any relational database system (currently PostgreSQL [29]).

3.1. Data model

A simplified representation of the data model is depicted in Figure 2. The design was inspired by a model proposed by scientists conducting crystallography research [27]. The model presented here is simplified and leaves out many details. The model has a set of core tables (light grey). They define the general framework. Each research group extends this basic set with individual tables that contain the information related to the particular experiments they perform and the programs they use (dark grey).

The `Project` lists general information about all projects being conducted. The `Experiment` table stores the basic information pertaining to every experiment conducted, including its experiment id, project id, creation date, etc. In addition, each type of experiment has its own table (e.g. PCR) with type specific information. The experiment-type tables are child tables of the `Experiment` table and inherit its primary key. These tables are added by a research group and reflect the specific experimental setup.

The `ExperimentType` table stores the names of all experiment types for which there is a dedicated table. It must be updated whenever a new experiment type table is added. For example, for the schema in Figure 2, it holds values *Pcr* and *Digestion*. It allows Exp-DB to dynamically identify a table name as being an experiment type, which is used, for instance, for automatic web-page generation.

The `Sample` table stores general information about experiment input and output data objects. As with experiments, each sample type has its own table storing its type-specific information. The names of the different Sample

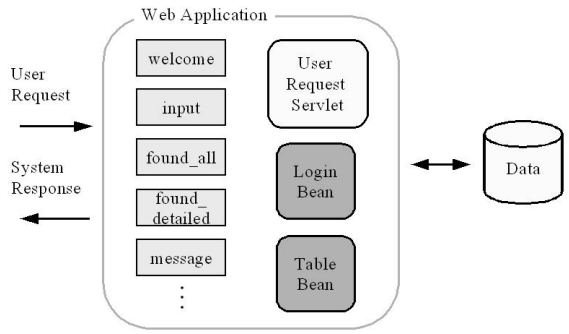

Figure 3. Web application architecture

type tables are stored in the `SampleType` table with a similar role as the `ExperimentType` table. A sample type related to a wet lab experiment type could be the description of a primer, an enzyme or similar. If related to a program, it is the description of input and output parameters.

`ExperimentTypeIO` stores input and output requirements for experiment types. The table contains an entry for each input and output sample type of a experiment type. The `ExperimentIO` table contains the specific input and output samples of the experiment instances. Each entry contains a reference to an experiment and a sample used for this experiment. It also contains a reference to the `ExperimentTypeIO` entry which it represents. This ensures that only input and outputs samples of the correct type are stored.

3.2. Web application

Figure 3 shows the internal architecture Exp-DB's middle tier. It follows the well-known MVC (Model View Controller) architecture [28]. The presentation logic provides the web-based user interface (*view*) using JavaServer Pages (JSP). The Java Servlet `UserRequestServlet` is the *controller*. It handles all incoming requests from the JSP pages. It calls the JavaBean `TableBean` (*model*) if necessary, and then redirects the response to the JSP responsible for returning a new web-page to the client. Web-services are currently not supported by Exp-DB.

The `TableBean` functions as a single, generic interface to all the tables in the database. It provides methods for querying, inserting, updating, and deleting data from a table. In order to handle non-trivial relationships between tables (e.g., between `Experiment` table and child `PCR` depicted in Figure 2), `TableBean` checks available meta-information as can be found in the `ExperimentType`, `ExperimentTypeIO` and `SampleType` tables. For example, suppose a user submits a request to read the records in `PCR`. The `TableBean` will find *Pcr* in the `ExperimentType` table, and therefore, perform corresponding reads both on `PCR` and the `Experiment` table. When adding new experiment or sample types to the data model, `TableBean` remains unchanged. The other components of the web application are similarly generic.

Exp-DB provides generic interfaces and procedures for the four basic operations read, insert, update, and delete. In a read operation, the user indicates via special JSPs a table and some search criteria, and the system retrieves and displays all the records in the given table that fulfill the search criteria. For insert operations, the user selects the table into which a new record will be inserted. The system retrieves the schema information for that table, and generates a corresponding web-form. The user completes and returns the form, and the system performs the insert. Update and delete operations require to specify the table to be updated, enter a search criteria, and select the desired target record from among the results matching the search criteria. For updates, also the new values must be specified.

While it is expected that research groups will develop their own interfaces over time, the basic operation types (1) read records from a tuple according to a search criteria, (2) insert a new record into a table, and (3) update/delete a given record, will likely remain the same.

4. Workflow Model of Exp-WF

4.1. Basic Model

The basic workflow model of Exp-WF is a simplified version of [18] and has three parts.

Workflow Specification The workflow specification is used to define *workflow patterns*. A workflow pattern consists of a set of *tasks* and a set of *transitions*. Each task is a place-holder for an experiment to be performed. In the following, we often use the words task and experiment interchangebly. A transition defines the control flow (i.e., execution order) between a *source* task and a *destination* task. Each data object passed between two tasks must be represented by its own (additional) transition. Additionally, tasks can have input objects not being produced by source tasks and output objects not consumed by destination tasks. A transition may be labeled with *conditions* which allows the modeling of iterative loops or branching. A condition will be evaluated once the destination task is considered for execution. Only if the outcome is *true* the transition can be used to reach the destination task. The specification model also includes *agents* which represent the people or robots to perform tasks. Our model also supports sub-workflows but they are not further described for space reasons. In Fig. 1, *pcr*, *digestion*, etc. become tasks of the workflow pattern for protein creation. The pattern also contains a reference to the

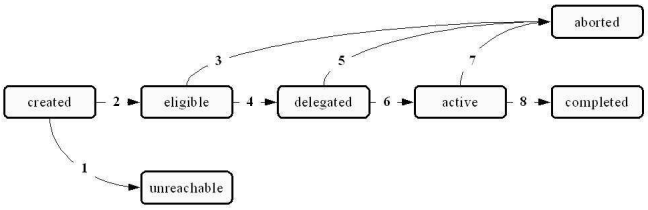

Figure 4. Execution Model

sub-workflow pattern encapsulated in the *protein production* task. The arrows *PCR-ligation*, *digestion-ligation* etc. are replaced by transitions for control flow and data objects. Conditions are attached to the transitions *transformation-PCR screening* and *transformation-miniprep* defining the criteria under which each transition should be followed.

Instance-level model The instance-level model describes how a given workflow pattern can be used to create multiple workflow instances each one representing a run-through. A workflow instance (for short, also "workflow") contains general information (date of creation, its current status, etc.), and *task instances*. A task instance stores execution details about each individual experiment and a reference to the agent performing it.

Execution Model The execution model employed by Exp-WF is somewhat similar to [23] and depicted in Fig. 4. It describes the states in which a task instance can be. A task instance is first *created*. If a required source task of any incoming transition aborts or conditions are not met the task and tasks that depend on it become *unreachable*. Otherwise, once all necessary input objects are available, the instance enters the *eligible* state. This state allows for the possibility to obtain user authorization before the task actually starts in order to make sure that the input objects are of good enough quality. Some automatic quality control may be implemented using conditions, but many decisions require an evaluation by humans or other agents. For tasks requiring authorization, the worklfow manager sends a request to an authorized agent and waits for response. If authorization is not given, the task instance enters the *aborted* state, otherwise it enters the *delegated* state (which is immediately entered if no authorization is needed). A delegated instance might be aborted, or start and enter the *active* state. Once execution has finished and all results are entered into the database, the instance enters the *completed* state. An active instance might also abort.

4.2. Supporting Multiple Task Instances

In the basic model each task of a workflow may only have one instance. This is not adequate for modeling scientific workflows for several reasons. Firstly, experiments may fail due to a variety of causes (e.g., culturing bacteria may produce an insufficient amount of bacteria). In a lab setting, such experiments might be repeated until a satisfactory result is obtained. Although we could model this as an iterative loop from a task to itself, this would require to add such loops to virtually any task in a workflow, making it a clumsy solution. Secondly, several experiments of the same type are often carried out simultaneously (and not serially), whereby only the best results are used as input for destination tasks. Since the basic model only allows a single instance per task, we would need to model all these experiment instances as individual (parallel) tasks sharing the same source and destination tasks. This is not only inelegant but also inadequate if the number of experiment instances to create is not known before runtime. Thirdly, at any point in a workflow, researchers may attempt to improve on the quality of their results by repeating some portion of the workflow. They then choose a previous task in the workflow at which to begin this repetition and resume from that task on. This would require to add backtracking transitions from every task to each one of its predecessors.

To address these issues, we have developed an extended model that supports multiple task instances for a given task in a workflow instance, and allows for less rigorous specification of task ordering. We faced two challenges. (1) How can the workflow manager determine when enough instances of a source task have completed for the next task to begin execution? (2) How to deal with transitions between tasks that might now have multiple instances associated with them. In particular, if one task uses another task's output for its own input, how to determine which instance's output to use for which instance's input.

In the **Extended Specification Model**, each task definition includes a default number of instances for the task. This is the number of "parallel" instances that will be automatically started when this task comes up for execution.

The **Extended Execution Model** distinguishes between a *task execution model* that describes the state of the execution of all instances pertaining to a given task, and a *task instance execution model* to refer to the execution of individual instances for a task. Both are variations of the basic execution model. The main difference for the task execution model is that it moves from eligible directly to active without a delegated state which only exists for task instances. Additionally, each active state contains pointers to all existing instances for this task. The task instance execution model contains all the states of the basic execution model except of unreachable and eligible, since they have already been determined for the task itself.

When a task becomes active, the default number of instances are created and executed according to the task in-

stance execution model. Additional instances may later be created arbitrarily by users, e.g., if existing instances are yielding unsatisfactory results. As long as any instance has yet to reach the completed or aborted state, the task will remain in the active state. Eventually, all instances of the task will either complete or abort. If every instance aborts, then the task itself aborts, otherwise it completes.

In order to allow backtracking, a task that has aborted, completed or is unreachable may be "restarted" either by a user or subsequent to the restarting of another task. Restarting sends a task back to the eligible state, and the eligibility requirements are reevaluated. This is necessary because the re-execution of a task may be triggered by the re-execution of one of its source tasks, and the conditions that previously enabled the task to execute may not longer hold true.

In the basic model the completion of a task possibly triggered the creation and execution of its destination tasks. In the new model, if a task waits for the previous task to complete, it risks waiting even while satisfactory inputs are present. On the other hand, if it begins executing too soon, it risks missing out on as-yet-unavailable results that may be of higher quality than those at its disposal. Our current setting requires that at least the default number of instances of the source tasks have completed. In conjunction with specifying that authorization is necessary for a task, this allows the system to begin any tasks without undue delay, while giving users the power to delay that execution if more source task instances are desired.

The final hurdle is to determine which instance's outputs to pass along to a destination task. One extreme would send the output of all completed instances and let the agent of the destination decide which one to take. This might overwhelm the receiver. The other extreme lets Exp-WF pick a single instance as the output provider. This requires an automated quality control mechanism. Our solution is a compromise forwarding outputs from all"successfully" completed source instances to the destination task. Success of an instance must now be specified explicitly by the executor of the task instance. Indicating success means that the instance's outputs are of "good enough" quality, and the instance is considered completed, otherwise aborted. The agent of the destination task still must determine among the outputs of successful source tasks, but ideally the range of choices will be limited to a more reasonable number. The choice of inputs must be specified when the agent sends the instance's results back to the system.

The final task of a workflow instance might now be reached although other tasks are still being processed. In order to control workflow termination, the final task of a workflow now requires authorization to be performed.

This extended execution model has been designed with wet lab experiments in mind. In this environment, although many experiments are now automatized and conducted by robots, experiment execution is still controlled by technicians that look at output results and decide which ones to use for further experiments. Hence, the workflow management system can take advantage of this interactive process to let the technician regulate how many task instances are eventually created and when a task really has completed.

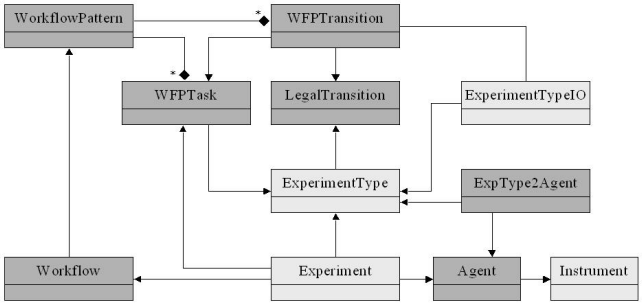

Figure 5. Workflow data model

4.3. Workflow data model

The workflow data model is also stored in the database. In fact, in order to integrate with the legacy LIMS it extends Exp-DB's original data model. Fig. 5 depicts the extensions. The challenges here lay in taking advantage of existing information and connecting it to workflow related information in a non-intrusive way. The diagram indicates workflow-related objects in dark and the objects in the original model in light.

Workflow patterns, tasks, and transitions are represented by the `WorkflowPattern`, `WFPTask`, and `WFPTransition` tables, respectively. `LegalTransition` specifies the execution order of experiment types. The entries in `ExperimentTypeIO` already ensure that tasks will actually have the input or output samples specified by the transition. Hence, data flow is already captured by Exp-DB's original data model. The `Agent` and `ExpType2Agent` tables hold information about the agent objects. An `Agent` represents external systems (e.g., users or robots) that are able to support experiment execution, and `ExpType2Agent` keeps track of the experiment types those agents are authorized to perform. Workflow instance information is stored in the `Workflow` table. Task instance information is simply stored in the `Experiment` table. This is the only table of the original data model that has been extended in order to point to the workflow and task the experiment belongs to, and the executing agent.

5. Workflow Manager

The workflow manager component of Exp-WF is responsible for the execution of previously defined workflows. Once a workflow is instantiated (supported by the user interface), the workflow manager has to detect eligible experiments and dispatch them to an agent for execution.

5.1. Integration Overview

The challenge is to embed workflow functionality in a modular way into the original LIMS without significantly changing the existing infrastructure. We use two basic mechanisms to facilitate the integration. The first allows the system to keep track of the state of workflows, the second provides a generic interface for agent communication.

5.1.1 Monitoring the workflow state

In Exp-DB, all information in the database is currently accessed via the web-based interface. Requests coming from this interface might affect the state of workflows. For example, data entered by a user might indicate the termination of an experiment, which should trigger the execution of other experiments.

This means we have to intercept the request processing of Exp-DB as described in Section 3.2, and redirect requests or responses that might impact the state of a workflow to the workflow engine. This includes update requests involving any data pertaining to workflow definitions, experiment types, experiments, samples, experiment I/O, and agents, as well as the responses to any such requests. Non-workflow-related actions (e.g., read-only operations) would be allowed to proceed normally.

Exp-WF uses Java servlet filter technology [15] to perform these interceptions. A servlet filter is a program that is associated with one or more web application resources such as servlets, and is invoked by the web container prior to and after each invocation of any of them. The filter can inspect and modify the http requests and response objects. This process is completely transparent to the targeted resource. Filter-resource associations are defined in the web applications's deployment description file, making it simple for users to apply the technology to any additional components they may add to their system. Note that a filter can also intercept requests and responses forwarded *within* the application. These characteristics lead us to believe that servlet filters provide an intuitive, flexible, and non-intrusive way to add workflow functionality to any LIMS based on web-server technology.

5.1.2 Agent framework

In order to automate experiment execution, the workflow manager requires a framework for registering and communicating with the external systems that will perform the experiments. Exp-WF uses software agents that act as wrappers for the external systems. A dedicated agent is required for each such system.

Our current implementation of Exp-WF uses persistent messages for agent communication. It uses OpenJMS [26], an open-source implementation of the Java Message Service (JMS) AP. JMS provides reliable and asynchronous communication. That is, message delivery is guaranteed even if communication partners are not connected all the time.

While data is eventually stored in relational format, external systems usually have their own proprietary data format. In order to provide a general data transfer format, we use XML. We use the system in [19] (modified to adjust to our needs) to automatically extract task input data from the relational database and represent it in a general XML format, and similarily to translate XML data back into the relational format. Each agent now must translate the XML input information into a format readable by the external system it represents. Exp-WF provides a template agent class that provides all necessary messaging functionality and provides several other helpful methods including default message handling procedures, simplifying the creation of a customized agent for an external instrument. For example, we used the template agent class to build an agent to represent an automated liquid handling robot used in one of the labs we have been working with. The only customization needed was the specification of the robot's required input and output format, which was of a typical comma-separated format. The location of the agent depends on the setup. Robots are often controlled via PCs that are directly connected with the robot. In this case, the agent could run on this specific PC.

5.2. Workflow engine

Architecture The architecture of the web application with the addition of the workflow engine (dark) is shown in Fig. 6. The Exp-WF components follows the same MVC design pattern as the original Exp-DB: the `WorkflowBean` represents the model and implements the core functionality of the workflow engine; the original JSPs build the view; and the `WorkflowServlet` and the `WorkflowFilter` serve as controller.

The `WorkflowBean`'s primary responsibility is to keep track of the state of workflow instances and tasks, and to direct the workflow execution, e.g., determining a task's eligibility, sending tasks to the `AgentManager`, or writing instance information to the database. The `WorkflowFilter` intercepts requests destined for the

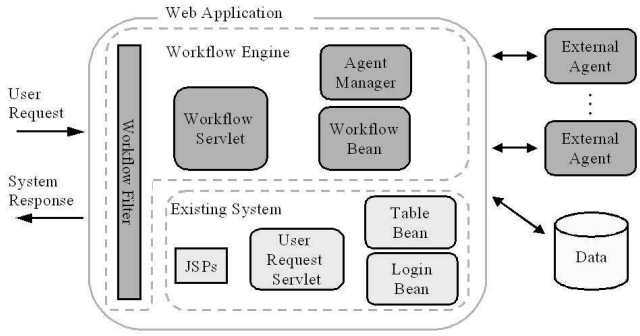

Figure 6. Workflow engine architecture

`UserRequestServlet` and responses destined to JSP pages. If a request or response is not relevant to workflow management, it simply proceeds to its original destination, otherwise it is redirected to the `WorkflowServlet`. The `WorkflowServlet` then invokes the relevant `WorkflowBean` methods. After that it may either forward the (potentially modified) request to its original destination, or send a (potentially modified) response to the user. The `AgentManager` is responsible for (1) choosing an appropriate agent for a task, (2) extracting the relevant input information from the database, (2) sending messages to the agent (e.g., containing task input data or abort notifications), (3) handling messages coming from the agents (e.g., containing output data or notifications as that the agent has started a given task instance), and (4) extracting output information and sending it to the `WorkflowBean` for insertion into the database.

Workflow and Task execution The workflow manager keeps track of the status of all active workflows and initiates the execution of any eligible tasks; thereafter, task execution and completion is triggered by user actions and agent messages. In order to check task eligibility the workflow manager keeps track of changes to tables like `ExperimentIO`. An active task instance is forwarded to the agent that controls execution. A human being is informed via email, and must then enter the results via the web interface. In case of a robot or external program, the corresponding agent performs the coordination, and results are automatically inserted into the database. Once all output has been entered, and the task is considered successful, the task instance has completed. The workflow manager then tests any destination tasks for eligibility, and the process repeats.

Request processing Once a user request has been identified as workflow-related by the `WorkflowFilter`, it may be handled in one of the three following ways as shown in Figure 7. (a) A user request may be redirected to the WorkflowServlet before it reaches its destination. There, it may be *preprocessed*. The workflow engine may then decide to forward the request to its original destination (if it is a valid action) or deny the request (if it violates the conditions on the state of workflow and task instances). (b) Alternately, a redirected request may actually be *processed* by the WorkflowServlet. In this case, the workflow manager is responsible for performing the requested action and generating an appropriate response. The original destination is bypassed entirely. For example, the workflow manager could assume responsibility for all insertions into experiment-type tables in order to facilitate the updating of workflow state information. (c) Finally, the `WorkflowFilter` may not redirect the request at all, but instead diverts the intended response for *postprocessing*. In this case, the request is handled by its intended resource, but the results contained in the response are examined by the workflow manager to see if any further action is necessary. For example, if a user enters the outcome of an experiment in `ExperimentIO`, the response will alert the workflow manager that the action succeeded, prompting it to check whether any workflow or instance state has to be changed. The workflow manager may modify the response sent back to the user with details about its own actions. Only successful user actions need to be post-processed, since failed operations do not change the state of the workflow.

Performance Evaluation We evaluated the performance of the system for various operations including various workflow and non-workflow related requests. Response times vary from 400 ms to 2000 ms. In general, little time was spent in the WorflowFilter, WorkflowServlet or WorkflowBean. Instead, the response time was mainly determined by the number of database read and write accesses. One has to be aware that a simple insert into an experiment related table can trigger several database reads in order to check whether this modification changes any task or workflow state. Sending messages to a persistent message queue also has some time overhead. Still, even a response time of 2000 ms is acceptable for user interaction.

6. Related Work

There are a number of existing workflow systems designed to support the control of scientific experiments. ZOO [1, 14] is probably the system most similar to Exp-WF in scope. It provides generic, modular experiment management support and an agent-based framework for incorporating external systems. However, data and workflow management in ZOO are almost completely integrated and managed within the database system. Our approach, in contrast, uses middleware based filter technology, and keeps the data management of the original LIMS system basically unchanged. Opera [2, 3] delegates all its data man-

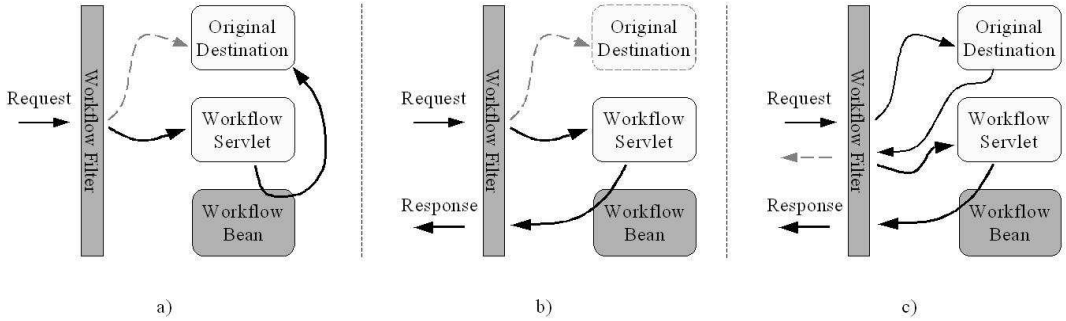

Figure 7. a) preprocessing b) processing c) postprocessing of a user request

agement requirements to its underlying database management system (DBMS), and provides an application layer to act as an interface between the system and its data repository. Customizing Opera to a specific application is a major undertaking that requires a good deal of expertise. IntelliGen [17] is a distributed WFMS tailored specifically to managing protein-protein interaction workflows. Rather than an agent-based framework for incorporating legacy applications, it requires a workflow specification to include task invocation or resource binding information that can be used in constructing wrappers for the systems that will invoke the task at run-time. The authors also discuss dynamic changes to workflows in order to be able to handle unpredicted results in experiments. In contrast, our extensions to the workflow model make the fact that experiments might be repeated many times explicit in the workflow description. In any of these approaches, the integration of an existing LIMS has not been discussed, and would be a major effort.

A different suite of workflow systems for scientific research [12, 6, 7, 22, 21] provide frameworks to process and relate diverse data sources and software programs like BLAST, and/or to control program execution across a grid. The goal is to ease workflow definition and automatize workflow execution. The workflow system is able to access data in different data sources, transform and transfer it to bioinformatics tools which perform computations, and feed the results of one tool into other tools or databases in order to execute what [22] calls a knowledge discovery workflow. The main challenges are to connect the diverse data sources and tools, to use distributed computing facilities for compute intensive tasks and to take advantage of advanced language abstractions like web-services. In contrast, the focus of our work is rather on the support of experimental workflows which produce the data that is later used in such knowledge discovery workflows. As such the issues addressed in our system are quite differed. The main challenges in our system are to provide a workflow model that is able to adequately model experiments as tasks, and to integrate the workflow engine with an already existing LIMS system that stores the experiment related data. However, the agent framework which already facilitates the execution of programs at remote locations, could be extended to manage compute intensive tasks, e.g., over a distributed computing environment.

There exist many workflow languages and models (e.g., [30, 24, 23]) often providing features that are currently not supported in Exp-WF like adaptive workflows and failure handling. In principle, such features could be integrated into Exp-WF. However, we are not aware of any model that is able to model multiple task instances as one task in the workflow pattern.

7. Conclusions

We have demonstrated that a workflow manager can be seamlessly integrated into an existing three-tier laboratory information management system in a modular, non-intrusive manner by using filter-based request and response interception and a modular agent framework. The original LIMS can be used as before. The integration was performed with no modifications to the components of the existing web application; moreover, only a single existing table (namely, the `Experiment` table) was modified when workflow concepts were added to the existing data model. We believe that workflow management capabilities can be integrated in a similar way into other data management systems sharing a similar, web-based multi-tier architecture. Furthermore, we have formulated a workflow model in which a task in the workflow description can represent many task instances. We believe this is an effective and convenient way to express the fact that many experiments are executed multiple times until sufficiently good output is produced.

We are currently investigating whether aspect-oriented programming can replace filter technology in case of systems that are not web-based. An aspect-oriented programming language like AspectJ [16] allows the specification of precise interceptor points, e.g., when a particular method of

an object is called. This is similar to filters but provides more alternatives as to where to intercept calls.

Acknowledgements

We thank the Macromolecular Structure Group at the Biotechnology Research Institute (BRI) of the National Research Council (http://www.bbri.nrc-cnrc.gc.ca/) for the many meetings and discussions we had.

References

[1] A. Ailamaki, Y. E. Ioannidis, and M. Livny. Scientific workflow management by database management. In *Int. Conf. on Scient. and Stat. Database Mgmt*, 1997.

[2] G. Alonso, W. Bausch, C. Pautasso, A. Kahn, and M. Hallett. Dependable computing in virtual laboratories. In *Int. Conf. on Data Engineering*, 2001.

[3] G. Alonso and C. Hagen. Geo-Opera: Workflow concepts for spatial processes. In *Int. Symposium on Advances in Spatial Databases*, 1997.

[4] Apache tomcat. http://jakarta.apache. org/tomcat/.

[5] P. Bertone, Y. Kluger, N. Lan, D. Zheng, D. Christendat, A. Yee, A. M. Edward, C. H. Arrowsmith, G. T. Montelione, and M. Gerstein. SPINE: an integrated tracking database and data mining approach for identifying feasible targets in high-throughput structural proteomics. *Nucleic Acids Research*, 29, 2001.

[6] S. S. Bhowmick, V. Vedagiri, and A. V. Laud. Hyperthesis: the grna spell on the curse of bioinformatics applications integration. In *Int. Conf. on Information and Knowledge Management (CIKM)*, 2003.

[7] M. C. Cavalcanti, R. Targino, F. A. Baião, S. C. Rössle, P. M. Bisch, P. F. Pires, M. L. M. Campos, and M. Mattoso. Managing structural genomic workflows using web services. *Data & Knowl. Engineering, Elsevier*, 53(1), 2005.

[8] J. Frew and R. Bose. Earth system science workbench: A data management infrascrutcure for earth science products. In *Int. Conf. on Statistical and Scientific Database Management*, 2001.

[9] D. Georgakopoulos, M. F. Hornick, and A. P. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2), 1995.

[10] N. Goodman, S. Rozen, L. D. Stein, and A. Smith. The LabBase system for data management in large scale biology research laboratoriess. *Bioinformatics*, 14, 1998.

[11] N. Goodman, S. Rozen, L. D. Stein, and A. G. Smith. The LabBase system for data management in large scale biology research laboratories. *Bioinformatics*, 14(7), 1998.

[12] Z. Guan and H. M. Jamil. Streamlining biological data analysis using bioflow. In *Int. Symp. on BioInformatics and Bioengineering (BIBE)*, 2003.

[13] P. W. Haebel, V. L. Arcus, E. N. Baker, and P. Metcalf. LISA: an intranet-based flexible database for protein crystallography project management. *Acta Crystallographica D57*, 2001.

[14] Y. E. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti. ZOO: a desktop experiment management environment. In *VLDB Conference*, 1996.

[15] Java Technology. http://java.sun.com/ products/.

[16] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. Getting started with AspectJ. *Communications of the ACM*, 44(10), 2001.

[17] K. J. Kochut, J. Arnold, A. P. Sheth, J. A. Miller, E. Kraemer, I. B. Arpinar, and J. Cardoso. IntelliGEN: A distributed workflow system for discovering protein-protein interactions. *Distributed and Parallel Databases, Sp. Issue on Bioinformatics*, 13(1), 2003.

[18] M. Kradolfer and A. Geppert. Modeling concepts for workflow specification. Technical report, Department of Computer Science, University of Zurich, 1997.

[19] D. Lee, M. Mani, F. Chiu, and W. Chu. NeT & CoT: Translating relational schemas to XML schemas using semantic constraints. In *Int. Conf. on Information and Knowledge Management*, 2002.

[20] X. Li, N. A. Naeem, and B. Kemme. Fine-granularity access control in 3-tier laboratory information systems. In *IDEAS*, 2005.

[21] D. T. Liu and M. J. Franklin. The design of GridDB: A data-centric overlay for the scientific grid. In *VLDB*, 2004.

[22] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows, to appear*, 2005.

[23] R. Müller. *Event-oriented dynamic adaptation of workflows*. PhD thesis, University of Leipzig, 2002.

[24] R. Müller, U. Greiner, and E. Rahm. AgentWork: A workflow system supporting event-oriented workflow adaptation. Technical report, Univ. of Leipzig, 2003.

[25] N. A. Naeem, S. Raymond, A. Poupon, M. Cygler, and B. Kemme. Exp-DB: Fast development of information systems for experiment tracking. In *Int. Conf. on Adv. Inf. Systems Engineering (Short Paper)*, 2003.

[26] OpenJMS Project Homepage. http://openjms. sourceforge.net/.

[27] A. Pajon, J. Ionides, J. Diprose, J. Fillon, R. Fogh, A. Ashton, H. Berman, W. Boucher, M. Cygler, E. Deleury, R. Esnouf, J. Janin, R. Kim, I. Krimm, C. Lawson, E. Oeuillet, A. Poupon, S. Raymond, T. Stevens, H. van Tilbeurgh, J. Westbrook, P. Wood, E. Ulrich, W. Vranken, X. Li, E. Laue, D. Stuart, and K. Henrick. Design of a data model for developing laboratory information management and analysis systems for protein production. *Proteins*, 58(2):278–284, 2005.

[28] J. B. J. Patterns. http://java.sun.com/blueprints/patterns/mvc-detailed.html.

[29] PostgreSQL. http://www.postgresql.org.

[30] W. van der Aalst, L. Aldred, M. Dumas, and A. ter Hofstede. Design and implementation of the YAWL system. In *Int. Conf. on Adv. Information Systems Engineering*, 2004.