

# MANAGING LABORATORY WORKFLOW WITH LABBASE\*

Lincoln Stein, Steve Rozen, Nathan Goodman  
*Whitehead Institute/MIT Center for Genome Research*  
*Building 300, One Kendall Square*  
*Cambridge, MA 02139, U.S.A*  
*{lstein,steve,nat}@genome.wi.mit.edu*

<http://jura.wi.mit.edu/rozen>

## ABSTRACT

We have designed and implemented a system for managing workflow in large semi-automated laboratory projects. This system sits on top of LabBase, a database management system specialized for representing complex biological data types and experimental steps. The workflow manager uses a simple state/transition model to represent laboratory protocols and offers a perl-based API for interaction with LabBase. In order to accommodate the need to modify the protocols frequently, the workflow management has been decoupled from data management, allowing the database schema to change without affecting the workflow protocols and vice versa.

The system has proven to be of great usefulness over a period of seven months of heavy usage and multiple laboratory protocol changes. Enhancements are planned that will further increase its robustness and utility.

LabBase, its API, and the workflow manager are publicly available under a license that permits free redistribution of source and object code, though currently a license for the commercial ObjectStore DBMS is also required.

*Keywords:* Laboratory Workflow Management, Laboratory Process Management, Laboratory Databases, Genome Informatics, Laboratory Informatics, Laboratory Information Systems, Molecular Biology Databases, Perl, LabBase.

## 1. Introduction

Biological laboratory research projects present a number of unique challenges for the designers of information systems. The experimental protocols are complex and involve many steps, typically with multiple branch points and cycles. There are many different reagents and many different types of experiment to track. The data types themselves are complex and are frequently interrelated in unusual ways. Superimposed on this complexity is a rapid rate of change. As old techniques are refined and new techniques introduced, laboratory protocols change, sometimes very rapidly.

The Whitehead Institute/MIT Center for Genome Research is typical of a large genome laboratory. The Center is engaged in several large-scale mapping projects, which together require the completion of several million individual experimental

---

\*This work was supported by funds from the National Institutes of Health, National Center for Human Genome Research, grant number P50 HG00098.

steps. Each project is composed of a number of interlocking laboratory protocols that range in size from five to 30 experimental steps. Reagents conceptually “flow through” the protocols as one experiment after another is performed. A reagent that fails a particular experimental step may be discarded, or may be reintroduced into the protocol in order to repeat one or more steps. At any point in time there are hundreds to thousands of reagents at intermediate stages of processing through the various laboratory protocols. Rapid protocol evolution is common. Minor protocol changes occur on a weekly basis, while a major protocol change (defined as the addition or deletion of one or more experimental steps) occurs almost monthly.

In previous papers [4, 11, 5] we described the design and implementation of the LabBase database management system (DBMS), a system tailored to the needs of laboratory information systems. LabBase provides support for:

1. data access from programs written in multiple languages and running on diverse platforms in a client/server manner,
2. the representation of unusual data types frequently encountered in biological labs, such as nucleotide sequences,
3. viewing data from both static and historical perspectives,
4. the ability to accommodate frequent schema changes.

In this paper we describe the “workflow manager”, an application programmer interface (API) layered on top of LabBase that allows laboratory protocols to be defined, monitored, and modified.

### *1.1. LabBase Background Information*

There are two key notions in the LabBase data model:

- Materials represent laboratory entities such as genetic markers, STSs, and microtitre trays.
- Steps represent operations that are performed on one or more materials and that generate experimental results, for example DNA sequencing steps, polymerase chain reaction (PCR) steps, microtitre tray setup steps and data analysis steps such as map construction and error checking.

Steps add information to materials by associating them with tag/value pairs. For example, a DNA sequencing step can associate a sequence tag and the DNA sequence data itself with a genetic marker material. Materials roughly correspond to the “object types” of semantic data models, while the tags correspond to attributes [6]. The main difference is that the tags are, strictly speaking, attached to the steps themselves rather than to the materials. In order to accommodate the fact that the laboratory may repeat the same experiment multiple times, we allow

any given step to be performed repeatedly on the same set of materials. When materials are queried for the values of their associated tags, by default the query processor returns the tag's most recent value. This is mostly commonly used to obtain the "current" value of an associated tag. It provides a static view into the data. The data can also be viewed historically as a "step history" consisting of a series of temporally related experimental steps. In this case the changing values of the tags can be examined and compared. In addition to materials, steps and tags, LabBase provides value sets, providing storage for sets of arbitrary values, including material references. The query language provides support for inserting and deleting materials from value sets, as well as for querying a value set's membership.

The LabBase query language [9, 10] uses a declarative datalog syntax similar to Prolog's [2]. For example, the query `STS(S)` will cause LabBase to successively bind the variable `S` to each STS material it finds in the database. The query `STS(S),DNA_sequence(S,Seq)`, instructs LabBase to bind variable `S` to a series of STSs and then to bind the DNA sequence of that STS to the variable `Seq` (in this case, `DNA_sequence` is a tag added to the STS material by a previous `Sequence_step`).

### *1.2. Example Laboratory Protocol*

A simplified laboratory protocol is shown in Figure [1]. Its steps are as follows:

1. Schedule an STS (a short DNA fragment that can be detected by PCR) for characterization on whole human DNA.
2. Perform the characterization experiment.
3. Interpret the characterization results according to certain criteria. For example, STSs that amplify multiple different DNA fragments from whole human DNA are probably part of a repetitive sequence and should be discarded (step 8).
4. Schedule successful STSs for screening against members of a yeast artificial chromosome (YAC) library (Each YAC contains a single large human DNA fragments; by determining the STSs contained within each of the YACs, the overlap relationships and ultimately the relative orders of the YACS can be determined).
5. Perform the STS/YAC screening experiment and score the results.
6. Interpret the results. If the experiment failed, schedule it to be redone once. Otherwise discard the STS (steps 9 and 10).

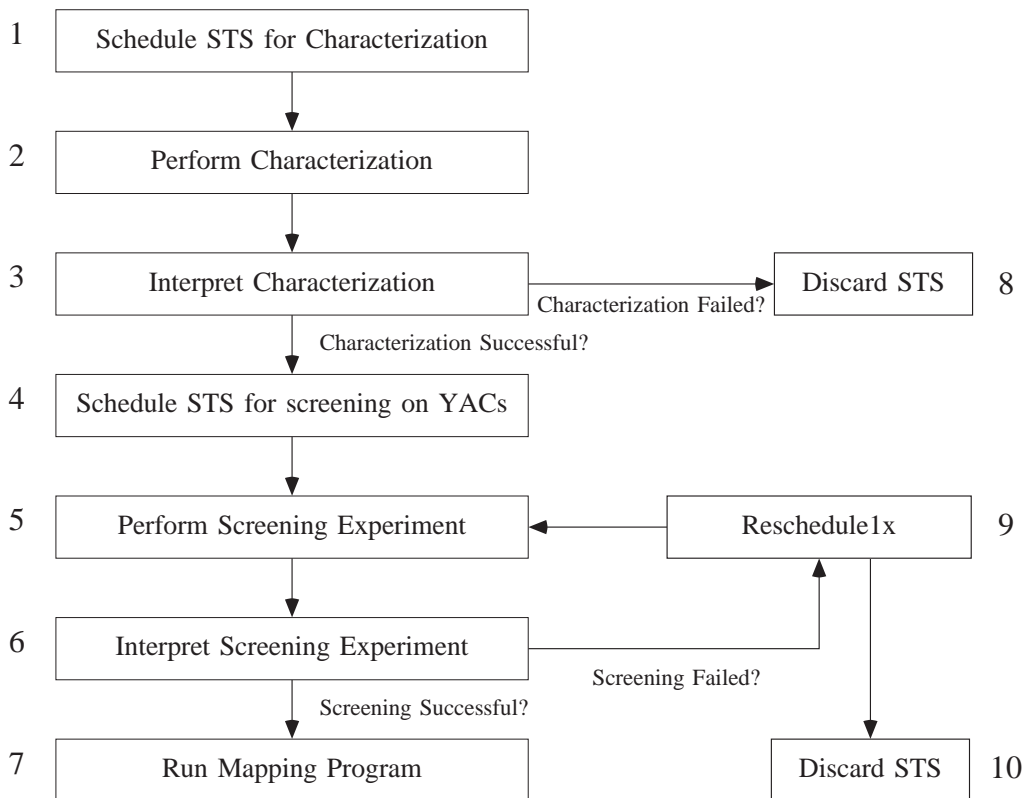


Figure 1: Example Laboratory Workflow.

7. Pass the results of successful screening experiments to a map assembly program.

⋮  
 (A number of other steps)  
 ⋮

Note that the rescheduling step (step 9) presents a potential problem for workflow management. Unless provisions are made, it is possible that this step can be repeated endlessly as an STS is screened and failed many times.

The LabBase schema for this protocol defines each of the experimental steps involved. For example, it defines `YAC_screening_step` and `screening_interpretation_step` to hold the results of the YAC library screening experiments and their interpretations respectively. However, there are no constraints built into the LabBase schema to specify that each interpretation step be preceded by a screening

step. The next sections describe how the LabBase workflow manager functions as a layer on top of LabBase to constrain and manage workflow-related transactions.

## 2. Workflow Manager

Our design of the LabBase workflow manager was strongly influenced by our style of laboratory software design [11]. For each step in the laboratory workflow there is a corresponding small program responsible for handling the database transactions and data analysis requirements of that step. These programs take the form of short perl [13] scripts, a language chosen for its ease of use, its support for shell-like system integration, and its powerful string matching and manipulation functions. To support the workflow shown in Figure [1] we have one perl script to handle STS scheduling, another to record the STS characterization results, another to interpret the results of the characterization, and so on. Some of these scripts are run as a result of user intervention. For example, the script that captures the results of STS screening experiments is run when a laboratory scientist presses a macro button on a Microsoft Excel worksheet on his Macintosh. Other scripts are run at regular intervals by the Unix cron facility, an example of which is the script that interprets the results of screening experiments.

To accommodate this style, the workflow manager was designed as a library of perl routines that are loaded by each data processing script at execution time. The workflow routines are part of an API to the LabBase DBMS.

### 2.1. Workflow Description Tables

We maintain descriptions of each of the laboratory protocols in external flat files called protocol description files. Each protocol description file contains one or several workflows, each of which involves a sequence of laboratory steps affecting a single LabBase material. The workflows are described using a state/transition model. Once a material enters a workflow, it is constrained to be in one of a limited number of states, and the legal transitions between states are explicitly enumerated. The protocol description file for the example STS characterization workflow has the contents shown in Figure [2].

The protocol description files use a tabular format that is easily understood by novice programmers. Each workflow is introduced by a short table that declares the LabBase material involved in the workflow, and a LabBase variable that, by convention, will be bound to this material during all queries involving the protocol. This declaration is followed by two more tables. The states table names all the valid states for the material. We tend to use long descriptive names for the states and for the purposes of brevity define a symbolic variable names for each state (in perl scalar variables are always preceded by a dollar sign, list variables are preceded by the “@” symbol, and subroutine calls are preceded by an ampersand.). The states table is followed by the transitions table, a list of the valid transitions and

```

WORKFLOW-MATERIAL      BINDING
-----
STS                      S

STATES                                VARIABLE-ABBREV.
-----
waiting_for_char_scheduling           $WFCHARSCHEDULE
waiting_for_characterization           $WFCHAR
waiting_for_char_interpretation       $WFCHARINTERP
characterization_failed                $CHARFAILED
waiting_for_screening_scheduling       $WFSCREENSCHEDULE
waiting_for_screening                  $WFSCREEN
waiting_for_screen_interpretation      $WFSCREENINTERP
waiting_for_screening_redo             $WFSCREENREDO
waiting_for_screening_redo_interp      $WFSCREENREDOINTERP
screening_failed                       $SCREENFAILED
waiting_for_mapping                    $WFMAPPING
mapped                                  $MAPPED

TRANSITIONS      FROM      TO
-----
enter_STS                $WFCHARSCHEDULE
schedule_charact        $WFCHARSCHEDULE      $WFCHAR
enter_charact            $WFCHAR                $WFCHARINTERP
accept_charact           $WFCHARINTERP         $WFSCREENSCHEDULE
fail_charact             $WFCHARINTERP         $CHARFAILED
schedule_screening       $WFSCREENSCHEDULE     $WFSCREEN
enter_screening          $WFSCREEN              $WFSCREENINTERP
accept_screening         $WFSCREENINTERP       $WFMAPPING
fail_screening           $WFSCREENINTERP       $WFSCREENREDO
reenter_screening        $WFSCREENREDO         $WFSCREENREDOINTERP
fail_screening_again     $WFSCREENREDOINTERP  $SCREENFAILED
accept_screening_again   $WFSCREENREDOINTERP  $WFMAPPING
map_STS                  $WFMAPPING            $MAPPED

```

Figure 2: STS Characterization Protocol Description File.

the source and destination state for each. Note that the problem of terminating the cycle in the workflow has been solved by “unrolling the loop”. The workflow defines distinct `waiting_for_screening` and `waiting_for_screening_redo` states, each with a distinct set of transitions. Another approach to this problem would have been to have the program responsible for this part of the workflow generate an *ad hoc* LabBase query to determine how many times the STS had been through the cycle and then make the appropriate transition.

The workflow manager has no built in provisions for modeling hierarchical trees of protocols and sub-protocols. All protocols are flat.

## 2.2. Workflow Manager Perl API

When a perl script responsible for handling a laboratory workflow step is executed, it must first load the perl LabBase library. It must then load the protocol for which it is responsible in a single call to `load_workflow()`. This call opens, parses and compiles the appropriate protocol description file, creating a series of perl function calls each corresponding to one of the states and transitions named in the protocol’s workflow tables. Each function call emits a LabBase query string that can be passed to LabBase for evaluation.

In the STS characterization protocol given above, loading of the protocol description file would create 12 perl function calls for LabBase state queries, and 12 function calls to generate queries to perform the corresponding state transitions. The perl script can then execute the queries necessary to accomplish its task.

- State queries:

```
waiting_for_char_scheduling()
waiting_for_characterization()
waiting_for_char_interpretation()
characterization_failed()
waiting_for_screening_scheduling()
waiting_for_screening()
waiting_for_screen_interpretation()
waiting_for_screening_redo()
waiting_for_screening_redo_interp()
screening_failed()
waiting_for_mapping()
mapped()
```

- Transition queries:

```
enter_STS()
schedule_charact()
enter_charact()
```

```

accept_charact()
fail_charact()
schedule_screening()
enter_screening()
accept_screening()
fail_screening()
reenter_screening()
fail_screening_again()
accept_screening_again()
map_STS()

```

It is important to note that none of the workflow functions actually queries or modifies the database. The functions return a LabBase query that can be used as a component within a more complex query at the discretion of the application programmer. A simple example of the API shows how it can fetch the names of all STSs in the `waiting_for_mapping` state:

```
@pending = &send_query($db,&waiting_for_mapping());
```

In this code fragment, the query returned by the `waiting_for_mapping()` function call is immediately sent to a LabBase database using another library routine, `send_query()` (this example assumes that the database has previously been opened using another function call and that the database identifier is stored in the perl variable `$db`). The list of STSs waiting for mapping is returned in the perl array variable `@pending`. A more complex example shows how the results of a workflow function call can be combined with other query fragments.

```
$query = &waiting_for_mapping() . ",YAC_hits(S,Hits)";
@hits = &send_query($db,$query);
```

In this fragment, the result of the `waiting_for_mapping()` function is combined via a string concatenation operation (“.”) with a query fragment that returns the list of YACs that were found to be positive in a previous screening experiment. Recall from above that the protocol description file declared that the LabBase variable `S` would be used as the standard binding for all materials of type STS within the STS characterization workflow. Similarly, here is one way to construct a query to determine if an STS named `D10S203` is in the `waiting_for_mapping` state :

```
$query = "STS_id(S,'D10S203')," . &waiting_for_mapping();
$in_state = &send_query($db,$query);
```

Transitions are handled in a similar manner. The following sequence of statements moves an STS named `D10S203` from the `waiting_for_interpretation` state to the `waiting_for_mapping` state:

```
$ok = &send_query($db,&accept_screening('D10S203'));
```

The workflow manager function `accept_screening()` emits LabBase code that checks that the STS is in the appropriate state. If so it performs the transition (i.e. moves D10S203 to the `waiting_for_mapping` state) and returns a true value in the perl variable `$ok`.

It is possible to add additional LabBase constraints to the transitions listed in the protocol description file. As a concrete example, it would be undesirable to schedule an STS for characterization unless its DNA sequence had been entered. The workflow manager allows arbitrary LabBase query fragments to be attached to transitions by entering this query as an optional third field of the transitions table. The movement of the material through the transition will only be performed if the constraint expressed by the query fragment returns a true value. In order to incorporate a check that the STS has a DNA sequence, the `schedule_charact` transition line could be rewritten in the protocol description file as:

```
⋮  
schedule_charact $WFCHARSCHEDULE $WFCHAR DNA_sequence(S,D)  
⋮
```

The LabBase query `DNA_sequence(S,D)` will return true if and only if the STS material bound to the variable `S` has a `DNA_sequence` tag. Constraint queries can be arbitrarily complex.

The workflow manager also contains numerous perl function calls for querying LabBase to determine what state a material is in, for moving materials between protocols, and for iterating over all the states contained within in a particular protocol. When combined with LabBase's native support for step history queries, these routines make it easy to determine a material's state and the route it followed to get there, as well as to report the status of a protocol as a whole. Generic report tools built on top of these routines allow the progress of a laboratory protocol to be followed so as to identify bottlenecks in the protocol and steps at which unacceptable failure rates are encountered.

### *2.3. Implementation*

The implementation of the workflow manager is straightforward. Internally, the states are represented in LabBase as value sets, unordered lists of materials. The function calls that correspond to state queries evaluate to a query of this form

```
value_set(state,variable)
```

where *state* is the name of a LabBase value set and *variable* is the LabBase variable name to be bound to the relevant material. The workflow manager makes the value set name unique for each protocol by prefixing the protocol name to the state

name. As an example, the perl function `waiting_for_mapping()` will evaluate to the LabBase query:

```
value_set('STS_WAITING_FOR_MAPPING',S)
```

This query binds variable `S` to each material in the value set `STS_WAITING_FOR_MAPPING`, listing the contents of the value set. The perl function calls that correspond to transitions evaluate to a more complex query of the form:

```
insist(material_id(variable, 'ID')),
      insist(value_set(state1, variable)),
      insist(constraints),
      delete(value_set(state1, variable)),
      insert(value_set(state2, variable)).
```

where

- *material\_id* is replaced by the appropriate LabBase predicate to derive the material from its string identifier,
- `ID` is the database ID for this material,
- *constraints*, are any optional constraints entered into the protocol description file,
- *state1* and *state2* are the starting and ending states of the transition, and
- *variable* is the LabBase variable to be bound to the material.

In the LabBase query language, the built-in predicate `insist()` returns an error if the expression within it evaluates to false.

As a concrete example, a call to the `schedule_charact('D10S203')` function would evaluate to the following LabBase query:

```
insist(STS_ID(S, 'D10S203')),
      insist(value_set('STS_WAITING_FOR_CHAR_SCHEDULING', S)),
      delete(value_set('STS_WAITING_FOR_CHAR_SCHEDULING', S)),
      insert(value_set('STS_WAITING_FOR_CHARACTERIZATION', S)).
```

If the constraint that the STS must have a DNA sequence were added to the `schedule_charact` transition in the manner described at the end of the previous section, the function would evaluate in this way:

```
insist(STS_ID(S, 'D10S203')),
insist(DNA_sequence(S, D)),
      insist(value_set('STS_WAITING_FOR_CHAR_SCHEDULING', S)),
      delete(value_set('STS_WAITING_FOR_CHAR_SCHEDULING', S)),
      insert(value_set('STS_WAITING_FOR_CHARACTERIZATION', S)).
```

The implementation of the states/transition workflow model is a very concrete incarnation of the way in which the biologists in the laboratory (and the programmers who support them) think about the workflow. The states are buckets, and markers move from bucket to bucket as they pass through the laboratory protocol.

### 3. Discussion

The design of the LabBase workflow manager involved a number of compromises and tradeoffs. The first decision was to keep workflow information separate from the LabBase schema. This separation occurs at two levels. At a trivial level, the workflow and the schema are decoupled because they are physically kept in different places. The schema is maintained within the LabBase database itself, while the workflow information is kept in flat files in a separate location. However, this design decision was made only as a matter of convenience. There is no fundamental reason that the states and transitions tables couldn't be stored within LabBase, and a future implementation of the workflow manager may indeed store its tables in this way.

At a more fundamental level the database schema, with its knowledge of materials and experimental steps, is almost completely decoupled from the workflow manager. Although it is common for there to be one-to-one correspondences between the experimental steps defined in the LabBase schema and the workflow transitions defined in the protocol description files, this is only a convention followed by the authors of the data management scripts used by the laboratory.

Our rationale for adopting this decoupled approach arises from our experience with the predecessor to LabBase, an object-oriented database for the management of genetic mapping projects [4] called MapBase. In MapBase, the status of a laboratory material within the genetic mapping protocol was derived when needed by examining all the experimental steps that had been performed on that material and determining the outcomes for each of those steps. MapBase thus tightly coupled the laboratory data with the workflow.

What we discovered is that this scheme did not offer the flexibility necessary to deal with the ever-changing nature of laboratory projects. As old techniques were improved and new ones introduced, experimental steps were modified, split up or merged. Each change in the laboratory protocol forced a database schema change, and each schema change in turn would force us to undertake a time-consuming re-evaluation of the workflow management. Often even a seemingly minor change such as exchanging the order of two experimental steps would unpredictably alter the status of materials that were in intermediate states of processing. In a few cases the addition of a new experimental step had the unforeseen side effect of reactivating hundreds of laboratory materials that had already been completely processed. Because these old entities were lacking the data related to the new experimental step, it was reasonable for the workflow management logic to infer that they weren't finished yet.

By decoupling the data gathering aspects of the laboratory protocol from workflow management, the LabBase workflow manager is able to cope with a rapidly changing laboratory environment. They are usually accommodated by making a small alteration to a perl script. Indeed, the workflow manager is immune to the schema-altering changes that occur when attributes associated with a particular experimental step are added or deleted.

However, many laboratory protocol changes, such as the addition or removal of a step, do require the workflow description to be changed because they correspond to a conceptual change in the workflow. When the workflow has to be modified it can be done by changing the protocol description file with a text editor and modifying one or two perl scripts. Because LabBase maintains the state of materials by storing them in sets, protocol modifications that require us to change the status of in-process materials can be handled easily by *ad hoc* LabBase queries that move materials from one set to another.

The workflow management system is also general enough for us to create generic workflow query tools, and to write status and accounting reports that work across diverse laboratory protocols. It should also be feasible to design graphical browsers to inspect the contents of laboratory protocols, although we have not done so yet.

The approach of decoupling the workflow management from the data could work with other data managers. For example, in the *C. elegans* database ACeDB [3] there is no built-in special-purpose workflow management support, but the status of a laboratory material could be stored in a “workflow-state” attribute, and step histories could be stored in a multivalued attribute. From a material’s state and implicit knowledge of the laboratory protocols, external programs could determine what experiments to perform next, just as they do when operating on LabBase data. Efforts are underway to provide explicit support for workflow management tasks by adding workflow steps to ACeDB schemas [8], and by adding client/server support to the ACeDB software [12].

A contrasting, tightly-coupled approach is taken by the Object-Protocol Model (OPM) [1], in which knowledge of the laboratory protocol (workflow) is an integral part of the database schema. In this object-oriented modeling language, each laboratory protocol is a database object with explicitly defined inputs and outputs. The inputs and outputs are database objects that correspond to laboratory entities of various sorts: a protocol has the effect of transforming one set of entities into another. OPM allows protocols to be composed of sub-protocols and sub-sub-protocols, and provides a full-fledged semantic data model in addition to its workflow-modeling facilities.

Our approach has disadvantages. The largest one is that it is up to the application programmer to maintain the semantic link between the experimental results that are entered into LabBase and the workflow transitions that are traversed in concert with those results. It is possible for the programmer to neglect to call a transition-generating function, or to make a transition under the wrong circumstances. In practice we have found that these errors occur rarely, and when they do it is easy to correct the problem using *ad hoc* LabBase queries. Even if a workflow

were to become completely scrambled it would still be possible to rebuild it using a well chosen set of queries on the step histories of the affected materials. The ability to put constraints on workflow transitions using arbitrary LabBase queries partially alleviates the risk that a buggy program will invalidate a workflow. However for constraints to be effective they must carefully anticipate potential problems, something that is not always easily accomplished.

Concurrency control is another area of potential concern. Although LabBase provides concurrency control among multiple clients at the query level, there is nothing to prevent two simultaneously-connected clients from attempting to call different transition functions for the same material. Only the first transition will have an effect; the workflow manager will reject the second because the material will not then be in an appropriate initial state. In order to avoid this potential problem, we have adopted a style in which each transition and its associated experimental step(s) are the responsibility of a single program. This works well in practice. (Of course a program can be responsible for several related transitions).

Another weakness in our approach is that workflow steps are entered into LabBase without giving any explanation of what transition corresponds to that step. Because the protocol description file resides outside of LabBase, it is inconvenient to annotate experimental steps with a record of the workflow transitions they were associated with. Particularly when a laboratory protocol has changed multiple times, it is not always obvious from looking at a material's step history why particular decisions were made at forks in the workflow. Finally, the LabBase workflow manager's reliance on perl is a potential limitation. It has not been a problem at the Genome Center because perl is essentially the only language that we use for system integration. However, it could become troublesome if we were to find the need to incorporate workflow management functions into data analysis programs written in C++ or Tcl [7]. We are confident that we could produce a C++ library with the same functionality.

#### 4. Conclusions and Future Directions

The workflow manager complements LabBase's built-in experimental step history facilities by adding a simple state/transition model to describe the steps of a laboratory protocol. Although the model is simple, in practice it has been more than adequate for all the tasks that we have encountered. The workflow manager has weathered remarkably well a period of 7 months of intense usage and multiple changes in the various protocols in use at the MIT Genome Center. It continues to be an essential component of our laboratory informatics system. There are still a number of deficiencies in the workflow manager in the areas of robustness, concurrency control, and loose coupling with the LabBase step history facilities. Our plans for future enhancements include:

1. Tighter integration with LabBase step histories. We intend to move the external protocol descriptions into LabBase itself and provide a mechanism in which

each experimental step is optionally associated with a pointer to the protocol in use and the workflow transition associated with that step. Ultimately we may make protocol descriptions part of the formal LabBase schema.

2. Better error checking and concurrency control.
3. Better support for workflow cycles and for nested workflows.
4. Extension of the workflow manager API to other languages such as Tcl and C++.
5. Creation of additional workflow management tools, such as graphical browsers.

LabBase, the workflow manager, and the LabBase API are in production at the Genome Center and are freely available.

## 5. Acknowledgements

Rob Nahf, Andre Marquis and Richard Resnick, the main application level programmers for the Genome Center, contributed many invaluable suggestions for the development of the workflow management system.

## 6. References

1. I-Min A. Chen and Victor M. Markowitz. The Object-Protocol Model, version 2.3. Technical Report LBL-32738, Lawrence Berkeley Laboratory, 1 Cyclotron Road, Berkeley, CA, 94720, USA, May 1994. This document and others on OPM available at [ftp://gizmo.lbl.gov/pub/DM\\_TOOLS/OPM/opm.html](ftp://gizmo.lbl.gov/pub/DM_TOOLS/OPM/opm.html).
2. William F. Clocksin and Christopher S. Mellish. *Programming in Prolog*. Springer-Verlag, 1987.
3. Richard Durbin and Jean Thierry-Mieg. A *C. elegans* database, 1991. Documentation, code and data available from anonymous ftp servers at <lirmm.lirmm.fr>, <cele.mrc-lmb.cam.ac.uk> and <ncbi.nlm.nih.gov>.
4. Nathan Goodman. An object oriented DBMS war story: Developing a genome mapping database in C++. In Won Kim, editor, *Modern Database Management: Object-Oriented and Multidatabase Technologies*. ACM Press, 1994.
5. Nathan Goodman, Steve Rozen, and Lincoln Stein. Building a laboratory information system around a C++-based object-oriented dbms. In *Proceedings of the 20th International Conference on Very Large Data Bases*, September 1994. Available at <ftp://genome.wi.mit.edu/pub/papers/Y1994/building.ps.Z>.

6. Richard Hull and Roger King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19:201–260, September 1987.
7. John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company, 1994.
8. Otto Ritter. Integrated genome database schema, 1994. Deutsches Krebsforschungszentrum, Heidelberg, This and other IGD-related documents available at <http://genome.dkfz-heidelberg.de/igd-docs/>.
9. Steve Rozen, Lincoln Stein, and Nathan Goodman. *LabBase User Manual*. Available at <ftp://genome.wi.mit.edu/pub/papers/Y1994/labbase-manual.ps>.
10. Steve Rozen, Lincoln Stein, and Nathan Goodman. Constructing a domain-specific DBMS using a persistent object system. In *Sixth International Workshop on Persistent Object Systems*, September 1994. In press. Available at <ftp://genome.wi.mit.edu/pub/papers/Y1994/labbase-design.ps.Z>.
11. Lincoln Stein, Andre Marquis, Ert Dredge, Mary Pat Reeve, Mark Daly, Steve Rozen, and Nathan Goodman. Splicing UNIX into a genome mapping laboratory. In *USENIX Summer 1994 Technical Conference*, pages 221–229, June 1994. Available at [ftp://genome.wi.mit.edu/pub/papers/Y1994/Summer94\\_Usenix.ps.Z](ftp://genome.wi.mit.edu/pub/papers/Y1994/Summer94_Usenix.ps.Z).
12. Jean Thierry-Mieg, October 1994. Personal communication.
13. Larry Wall and Randal L. Schwartz. *Programming perl*. O'Reilly & Associates, Inc., 1990.