Modeling and Storing Scientific Protocols

Natalia Kwasnikowska[1], Yi Chen[2], and Zoé Lacroix[2]

[1] Hasselt University and Transnational University of Limburg, Belgium
natalia.kwasnikowska@uhasselt.be
[2] Arizona State University, Tempe, AZ, USA
{yi,zoe.lacroix}@asu.edu
http://bioinformatics.eas.asu.edu

Abstract. We propose an abstract model for scientific protocols, where several atomic operators are proposed for protocol composition. We distinguish two different layers associated with scientific protocols: design and implementation, and discuss the mapping between them. We illustrate our approach with a representative example and describe ProtocolDB, a scientific protocol repository currently in development. Our approach benefits scientists by allowing the archiving of scientific protocols with the collected data sets to constitute a scientific portfolio for the laboratory to query, compare and revise protocols.

1 Introduction

Scientific discovery relies on the adequate expression, execution, and analysis of scientific protocols. Although data sets are properly stored, the protocols themselves are often recorded only on paper or remain in a digital form developed to implement them. Once the scientist who has implemented the scientific protocol leaves the laboratory, the record of the scientific protocol may be lost. Collected data sets without the description of the process that produced them may become meaningless. Moreover, to support scientific discovery, anyone should be able to reproduce the experiment. Therefore, a detailed description of the protocol is necessary, together with the collected data sets.

A scientific protocol is the process that describes the experimental component of scientific reasoning. Scientific reasoning follows a hypothetico-deductive pattern and is composed of the succession of the expression of a *causal question*, a *hypothesis*, the *predicted results*, the design of an *experiment*, the actual *results* of the experiment, the comparison of the predicted and the experimental results, and the *conclusion*, supportive or not of the hypothesis [1]. Scientific protocols (also called data-analysis pipelines, workflows or dataflows) are complex procedural processes composed of a succession of tasks expressing the way the experiment is conducted. They usually involve a data-gathering stage, that may be followed by an analysis stage. A scientific protocol thus describes how the experiment is conducted and records all necessary information to reproduce the experiment. In bioinformatics, the importance of identifying protocol tasks has been addressed by Stevens et al. [2] and Bartlett et al [3], while Tröger [4] has proposed a language for expressing *in silico* protocols that approximates research method used for *in vitro* experiments.

We propose a high-level abstract model for scientific protocols representing two different layers associated with scientific protocols: design and implementation, and discuss the mapping between them. Our approach benefits scientists by allowing the archiving of scientific protocols with data sets to constitute a scientific portfolio for the laboratory to query, compare and revise protocols.

2 Related Work

Several approaches integrate scientific protocol models with a database system, but provide little support for the actual design phase of a protocol, do not distinguish between design and implementation, and provide limited support for querying and versioning of protocols. They include the Object Protocol Model [5] and Zoo [6] that both use the object-oriented data model. More recent efforts propose an integration of protocols and relational databases. Shankar et al. [7] propose a language for modeling protocols that is tightly integrated with SQL.

On the other hand, several systems focus on the design issues of protocols, sometimes combined with the (distributed) execution of protocols, but without fully leveraging the storage and query capability of databases. They include Taverna [8], with a vast integration of bioinformatics resources, and Kepler [9, 10], based on the Ptolemy II system. WOODSS [11] emphasizes the support of several abstraction levels of protocol design and facilitates protocol composition and reuse. Several researchers have agreed on the separation of the design of a protocol from its implementation [10, 12–14]. For instance, Ludäscher et al. [10] propose a distinction between abstract and concrete protocols and use database mediation techniques for an abstract-to-concrete translation. Zhao et al. [14] propose an XML-based virtual data language for a typed and compositional protocol specification, with mapping descriptors between the design and implementation. A formal graphical language for hierarchical modeling of protocols has also been proposed in Hidders et al. [15], and combines Petri nets with operators and typing system from nested relational calculus.

Compared with previous work, we focus here on how to define a formal, abstract model for defining scientific protocols that is as high-level as possible, so as to be suitable to general applications as well as for storage of protocols in a database system. We distinguish the design from possible implementations of protocols and define the mapping between them.

3 Modeling Scientific Protocols

The abstract protocol definition language introduced in this section aims at representing the structure of scientific protocols, and is, by design, unbiased towards any specific data model or query language. We aim at modeling *in vivo*, *in vitro*, as well as *in silico* experiments.

Each step of a scientific protocol can be represented by a *task* [2, 3]. To model a protocol, we distinguish its *design*, that captures its scientific aim, from its *implementation*, that specifies resources selected to execute the tasks. This distinction allows for comparison between different choices of resources, allowing the scientist to select the implementation best meeting the protocol's needs. Therefore we decompose each scientific protocol into two components: *protocol design* and *protocol implementation*. Both components consist of coordinated *tasks*, but at different abstraction levels. As we present a syntactical model, we will specify the data flow by identifying its *conceptual type* and *format*.

Each task of the protocol design is defined by its *task name*, *conceptual input type*, and *conceptual output type*. When an ontology is available to describe the scientific objects and tasks involved, the input and output of each protocol design task may be defined by their respective concept classes. The protocol design task itself may appear in the ontology, as a relationship defined between the input concept class and output concept class.

A task of the protocol implementation describes the resource selected to implement a protocol design task. Each protocol implementation task is defined by its *application name*, *input format*, and *output format*. The input/output format is a possible representation for the conceptual input/output type of the corresponding design task. The name of a protocol implementation task denotes a resource, an application or a service, implementing the corresponding protocol design task, together with its annotation (e.g., parameters, url etc).

3.1 Protocol Design Model

A scientific protocol can be defined inductively from tasks, or basic protocols, and four connectors. Formally, the protocol design model is defined as follows.

Definition 1. *Let $\mathcal{T}$ be a set of task names. Let $\mathcal{C}$ be a set of conceptual type names, over which an operator $\oplus$ is defined and a sub-typing relation "$\preceq$". A protocol design task* $\mathrm{T_D}$ *is a triple (i, n, o) with $i, o \in \mathcal{C}$ and $n \in \mathcal{T}$. The set $\mathbb{T}_{\mathcal{T},\mathcal{C}} = \mathcal{C} \times \mathcal{T} \times \mathcal{C}$ is the set of protocol design tasks defined from $\mathcal{T}$ and $\mathcal{C}$. Now we define recursively the set $\mathbb{P}_{\mathcal{T},\mathcal{C}}$ of protocol designs defined from $\mathcal{T}$ and $\mathcal{C}$, and for each protocol design* D *we impose requirements on its input type* $\mathrm{In(D)}$ *and its output type* $\mathrm{Out(D)}$*:*

- *if* $\mathrm{D} = (i, n, o)$ *then* $\mathrm{D} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$*, with* $\mathrm{In(D)} = i$ *and* $\mathrm{Out(D)} = o$*;*
- *if* $\mathrm{D_1} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$ *and* $\mathrm{D_2} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$ *and* $\mathrm{Out(D_1)} \preceq \mathrm{In(D_2)}$ *then* $\mathrm{D_1} \centerdot \mathrm{D_2} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$*, with* $\mathrm{In(D_1 \centerdot D_2)} \preceq \mathrm{In(D_1)}$ *and* $\mathrm{Out(D_2)} \preceq \mathrm{Out(D_1 \centerdot D_2)}$*;*
- *if* $\mathrm{D_1} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$ *and* $\mathrm{D_2} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$ *then* $\mathrm{D_1} \oplus \mathrm{D_2} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$*, with* $\mathrm{In(D_1 \oplus D_2)} \preceq \mathrm{In(D_1)} \oplus \mathrm{In(D_2)}$ *and* $\mathrm{Out(D_1)} \oplus \mathrm{Out(D_2)} \preceq \mathrm{Out(D_1 \oplus D_2)}$*;*
- *if* $\mathrm{D} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$ *and* $\mathrm{Out(D)} \preceq \mathrm{In(D)}$ *and k is an integer then* $\mathrm{D}^k \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$*, with* $\mathrm{In(D}^k) \preceq \mathrm{In(D)}$ *and* $\mathrm{Out(D)} \preceq \mathrm{Out(D}^k)$*;*
- *if* $\mathrm{D} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$ *and* $\mathrm{Out(D)} \preceq \mathrm{In(D)}$ *then* $\mathrm{D}^* \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$*, with* $\mathrm{In(D}^*) \preceq \mathrm{In(D)}$ *and* $\mathrm{Out(D)} \preceq \mathrm{Out(D}^*)$.

In the above definition, the operator "$\centerdot$" denotes the *successor* connector, i.e., the serial composition. The operator "$\oplus$" denotes the *split-merge* connector, i.e., the parallel composition. The operators k and $*$ denote *k-recursion*, and *star-recursion*, respectively. Relation "$\preceq$" denotes sub-typing between conceptual type names, provided by chosen ontology or type system. Type $i_1 \oplus i_2$, with $i_1, i_2 \in \mathcal{C}$, denotes a collection type, whose precise semantics depends on the semantics of the split-merge connector.

We emphasize that Def. 1 only captures the syntax of a protocol design, where the data flow is only described in terms of conceptual type names. Nevertheless, based on the interaction with our collaborators, we claim that our definition of protocol design is sufficient to faithfully model scientific protocols used in practice, once suitable semantics are provided for the operators.

Definition 2. *Let $(i, n, o) \in \mathcal{C} \times \mathcal{T} \times \mathcal{C}$. We define recursively the set of types Types*(D) *and the set of Tasks*(D) *of a protocol design* D *as follows:*

– *if* $\mathrm{D} = (i, n, o)$ *then Types*$(\mathrm{D}) = \{i, o\}$ *and Tasks*$(\mathrm{D}) = \{(i, n, o)\}$,
– *if* $\mathrm{D} = \mathrm{D}_1 \centerdot \mathrm{D}_2$ *or* $\mathrm{D} = \mathrm{D}_1 \oplus \mathrm{D}_2$, *then Types*$(\mathrm{D}) =$ *Types*$(\mathrm{D}_1) \cup$ *Types*(D_2) *and Tasks*$(\mathrm{D}) =$ *Tasks*$(\mathrm{D}_1) \cup$ *Tasks*(D_2),
– *if* $\mathrm{D} = \mathrm{D}_1^k$ *or* $\mathrm{D} = \mathrm{D}_1^*$, *then Types*$(\mathrm{D}) =$ *Types*(D_1) *and Tasks*$(\mathrm{D}) =$ *Tasks*(D_1).

We say that a protocol design D is *composed of* the tasks in *Tasks*(D). If a protocol design D is of the form $\mathrm{D}_1 \centerdot \mathrm{D}_2$ or $\mathrm{D}_1 \oplus \mathrm{D}_2$, with $\mathrm{D}_1, \mathrm{D}_2 \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$, then D is *directly composed of* D_1 and D_2. Similarly, if a protocol design D is of the form D_1^k or D_1^*, then D is *directly composed of* D_1. If a protocol design D is composed of D_1, then we also call D_1 a *sub-protocol* of D.

3.2 Protocol Implementation Model

Once the design of a protocol is defined, its specification in terms of resources used to execute it may be defined. Each design step may be implemented by specifying the input format, an application name, and the corresponding output format. Although sometimes a design step can be implemented by a single implementation step, it is common that a design step needs to be mapped to a complex process, involving multiple biological resources, thus to a sub-protocol rather than a single task of the implementation protocol. The need for a sub-protocol to implement a single design task may occur to include adapters to translate the output format from the previous implementation step into the input format of the selected implementation resource, or for specifying alternative implementations.

The protocol implementation model is similar to the protocol design model. Specifically, rather then a set of task names $\mathcal{T}$, we now have a set of application names $\mathcal{A}$. Rather then a set of conceptual type names $\mathcal{C}$, we now have a set of format names $\mathcal{F}$, over which an operator $\oplus$ is defined. The set of protocol implementation tasks $\mathbb{T}_{\mathcal{A},\mathcal{F}}$ and the set of protocol implementations $\mathbb{P}_{\mathcal{A},\mathcal{F}}$ are defined similar to Def. 1, except that for the sake of concreteness, we replace sub-typing "$\preceq$" on conceptual type names by equality "=" on format names.

The set *Formats*(I) of format names and the set *Resources*(I) of application names of an protocol implementation I, i.e., $\mathrm{I} \in \mathbb{P}_{\mathcal{A},\mathcal{F}}$, have a definition similar to Def. 2. It is worth noting that *Formats*(I) and *Resources*(I) provide basic provenance information for the data collected by executing protocol I.

3.3 Mapping Design to Implementation

Each design task of the design protocol may be mapped to one or more implementation tasks or protocols.

Definition 3. *A* conceptual type mapping *is a partial function* $\varphi_{\mathcal{C}} \colon \mathcal{C} \to \mathcal{F}$. *A* protocol design task mapping *is a partial function* $\varphi_{\mathcal{T}} \colon \mathbb{T}_{\mathcal{T},\mathcal{C}} \to \mathbb{P}_{\mathcal{A},\mathcal{F}}$. *A protocol design task mapping* $\varphi_{\mathcal{T}}$ *is said to be* consistent *with a conceptual type mapping* $\varphi_{\mathcal{C}}$ *if for every protocol design task* $\mathrm{T_D} \in \mathbb{T}_{\mathcal{T},\mathcal{C}}$ *it holds that if* $\varphi_{\mathcal{T}}(\mathrm{T_D}) = \mathrm{I}$ *then* $\mathrm{In}(\mathrm{I}) = \varphi_{\mathcal{C}}(\mathrm{In}(\mathrm{T_D}))$ *and* $\mathrm{Out}(\mathrm{I}) = \varphi_{\mathcal{C}}(\mathrm{Out}(\mathrm{T_D}))$. *If* $\varphi_{\mathcal{T}}(\mathrm{T_D}) = \mathrm{I}$ *then we call* I *an implementation of protocol design task* $\mathrm{T_D}$ *under* $\varphi_{\mathcal{T}}$.

Definition 4. *Let* $\mathrm{D}, \mathrm{D}_1, \mathrm{D}_2 \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$. *Given a protocol design task mapping* $\varphi_{\mathcal{T}}$ *we define its* generalization $\hat{\varphi}_{\mathcal{T}} \colon \mathbb{P}_{\mathcal{T},\mathcal{C}} \to \mathbb{P}_{\mathcal{A},\mathcal{F}}$ *such that* $\hat{\varphi}_{\mathcal{T}}$ *corresponds to* $\varphi_{\mathcal{T}}$ *on* $\mathbb{T}_{\mathcal{T},\mathcal{C}}$ *and:*

- $\hat{\varphi}_{\mathcal{T}}(\mathrm{D}_1 \centerdot \mathrm{D}_2) = \hat{\varphi}_{\mathcal{T}}(\mathrm{D}_1) \centerdot \hat{\varphi}_{\mathcal{T}}(\mathrm{D}_2)$,
- $\hat{\varphi}_{\mathcal{T}}(\mathrm{D}_1 \oplus \mathrm{D}_2) = \hat{\varphi}_{\mathcal{T}}(\mathrm{D}_1) \oplus \hat{\varphi}_{\mathcal{T}}(\mathrm{D}_2)$,
- $\hat{\varphi}_{\mathcal{T}}(\mathrm{D}^k) = \hat{\varphi}_{\mathcal{T}}(\mathrm{D})^k$ *and*
- $\hat{\varphi}_{\mathcal{T}}(\mathrm{D}^*) = \hat{\varphi}_{\mathcal{T}}(\mathrm{D})^*$.

We call $\hat{\varphi}_{\mathcal{T}}$ *a* protocol design mapping. *If* $\mathrm{D} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$ *and* $\mathrm{I} = \hat{\varphi}_{\mathcal{T}}(\mathrm{D})$ *then* I *is an implementation of protocol design* D *under* $\hat{\varphi}_{\mathcal{T}}$.

Definition 5. *Let* $\mathrm{D} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$. *We define the set* $\Phi(\mathrm{D})$ *of all possible implementations of* D *and its associated mappings as the set of all tuples* $(\varphi_{\mathcal{C}}, \varphi_{\mathcal{T}}, \mathrm{I})$ *where:*

- $\varphi_{\mathcal{C}}$ *is a conceptual type mapping with* $dom(\varphi_{\mathcal{C}}) = Types(\mathrm{D})$,
- $\varphi_{\mathcal{T}}$ *is a protocol design task mapping with* $dom(\varphi_{\mathcal{T}}) = Tasks(\mathrm{D})$ *and consistent with the conceptual type mapping* $\varphi_{\mathcal{C}}$,
- I *is a protocol implementation of* D *under* $\hat{\varphi}_{\mathcal{T}}$.

Finally we define the protocol itself, composed of a protocol design and a set of protocol implementations.

Definition 6. *We define a* protocol $\mathrm{P} = (\mathrm{D}, \mathrm{Imp}(\mathrm{D}))$ *as a pair of protocol design* $\mathrm{D} \in \mathbb{P}_{\mathcal{T},\mathcal{C}}$ *and* $\mathrm{Imp}(\mathrm{D})$ *being a finite subset of* $\Phi(\mathrm{D})$.

4 Example of a Scientific Protocol

We present a representative example of scientific protocol: *Study of germination of Lesquerella seeds.*[3] Lesquerella species are a promising oil crop with potential for industrial applications. Different members of that species possess different traits with regard to oil content, oil quality and yield. Current breeding programs aim to produce a variety suitable for commercial cultivation. One of the prerequisites to achieve this aim, is prolonged storage of seeds. The following protocol (restricted here to the *in silico* part for space reasons) was developed to determine base and optimal temperatures for germination of different Lesquerella seeds.

4.1 Statistical Analysis of Lesquerella Germination Data

The data obtained from the *in vitro* part of the experiment and stored in `Observations.xls`, was analyzed using SAS programs.

1. Determination of maximum germination was performed by succession of two SAS programs: max and first obs.sas used `Observations.xls` as input and produced `max percentages.xls` as output. That file was used as input to merge maxmin.sas, which produced the file `maxmin germshoots.xls`.
2. Germination proportions were analyzed using program genmod.sas, with the file `maxmin germshoots.xls` as input and two files as output: `maxshoots diffs.xls` and `maxshoots lsmeans.xls`.
3. Preprocessing of data necessary for determination of base and optimal temperatures for germination was achieved in two sub-steps. `Observations.xls` was used as input to sample numbers for DAPest.sas resulting in file `DAPest sample numbers.xls`, and was subsequently used as input to DAPest.sas which produced file `DAPestData.xls`. Also, `Observations.xls` was used as input to graphing to print.sas, which produced five bitmaps.
4. Base temperature (TB) for germination was determined by two separate methods, but only one of the methods was suitable for determining optimal temperatures (TO).
   (a) TB by regression analysis — reg.sas was run 4 times with `DAPestData.xls` as input and producing an Excel file each time. Those four files were subsequently merged by merge datasets.sas into a single Excel file. That file was analyzed with proc mixed.sas producing two output files `TbG50mns.xls` and `TbG50mndiffs.xls`.
   (b) TB and TO by 2-phase linear regression, broken model — the following analysis was repeated 13 times, for each kind of seed. `DAPestData.xls` was used as input to pho341.sas, producing an intermediate file. That file served as input to pho342.sas, producing another intermediate file which was used as input to broken3.sas. The latter produced two Excel files: `SeedID (limits).xls` and `SeedID (limits2).xls`.

[3] This protocol was collected at the U.S. Arid-Land Agricultural Research Center, Maricopa, AZ, courtesy of Jeff White and Neal Adam, the author of the protocol.

4.2 Analysis of the Structural Features

Analyzing scientific protocols, we frequently observe that a single protocol step includes multiple tasks. Step 1 for determining maximum germination of seeds includes two sub-steps, each consisting of the execution of a SAS program.

The enumeration of steps does not always reflect the order of tasks. In step 4, step 4a for computing base temperature and step 4b for computing base and optimal temperatures, can be executed in parallel, although they are stated in sequential order in the example. Some steps introduce a loop, e.g., step 4b is performed for every kind of seed. This is a particular kind of loop, that can be expressed by an iteration over the "collection" of seeds.

We see that the main structure of the protocol is mostly linear (step 1 and 2), or parallel (step 4a and 4b) or introduces a loop (step 4b). We also observe that the description of the protocol mixes the design with implementation. The implementation itself can be diverse. Most steps are implemented by using applications, but sometimes manual interaction may be necessary.

4.3 Example Protocol Model

The protocol presented in Sect. 4.1 can be modeled with the definitions of Sect. 3 as follows. First, we define the set of type names $\mathcal{C}$ as $\{\mathbf{SeedData}\}$ and the set of design task names $\mathcal{T}$ as $\{\mathrm{MaxGermination}, \mathrm{Proportions}, \mathrm{Preprocessing}, \mathrm{BaseTemp}, \mathrm{BaseAndOptTemp}\}$. We define the following protocol design tasks:

$\mathrm{T}_{\mathrm{D}1}$: $(\mathbf{SeedData}, \mathrm{MaxGermination}, \mathbf{SeedData})$,
$\mathrm{T}_{\mathrm{D}2}$: $(\mathbf{SeedData}, \mathrm{Proportions}, \mathbf{SeedData})$,
$\mathrm{T}_{\mathrm{D}3}$: $(\mathbf{SeedData}, \mathrm{Preprocessing}, \mathbf{SeedData})$,
$\mathrm{T}_{\mathrm{D}4}$: $(\mathbf{SeedData}, \mathrm{BaseTemp}, \mathbf{SeedData})$,
$\mathrm{T}_{\mathrm{D}5}$: $(\mathbf{SeedData}, \mathrm{BaseAndOptTemp}, \mathbf{SeedData})$.

We define now protocol design D with input $\mathrm{In(D)} = \mathbf{SeedData}$ and output $\mathrm{Out(D)} = \mathbf{SeedData}$ as $\mathrm{D} = \mathrm{D}_1 \oplus \mathrm{D}_2$, with $\mathrm{D}_1 = \mathrm{T}_{\mathrm{D}1} \centerdot \mathrm{T}_{\mathrm{D}2}$, $\mathrm{D}_2 = \mathrm{T}_{\mathrm{D}3} \centerdot \mathrm{D}_3$, $\mathrm{D}_3 = \mathrm{T}_{\mathrm{D}4} \oplus \mathrm{D}_4$ and $\mathrm{D}_4 = \mathrm{T}_{\mathrm{D}5}$[13]. Note that D_1 corresponds to steps 1 and 2 in our example, D_2 to steps 3 and 4, D_3 to step 4 and D_4 to step 4b. Last but not least, D represents the design of the whole protocol (left-hand side of Fig. 1).

Because the description presented in Sect. 4.1 is an actual implementation, our protocol implementation follows it closely, with following simplifications. Whenever multiple Excel files where generated, we assume they could have been equally merged into one file with multiple tabs. If the multiple outputs have different format names, additional converters are introduced. We plan to address the issue of multiple outputs in the future when we define operator semantics.

We define the set of format names $\mathcal{F}$ as $\{\texttt{Excel}, \texttt{Bitmap}\}$ and we simply use the set of program names as $\mathcal{A}$. The protocol implementation tasks are:

$\mathrm{T}_{\mathrm{I}1}$: $(\texttt{Excel}, \textsf{max and first obs.sas}, \texttt{Excel})$,
$\mathrm{T}_{\mathrm{I}2}$: $(\texttt{Excel}, \textsf{merge maxmin.sas}, \texttt{Excel})$,
$\mathrm{T}_{\mathrm{I}3}$: $(\texttt{Excel}, \textsf{genmod.sas}, \texttt{Excel})$,
$\mathrm{T}_{\mathrm{I}4}$: $(\texttt{Excel}, \textsf{sample numbers for DAPest.sas}, \texttt{Excel})$,

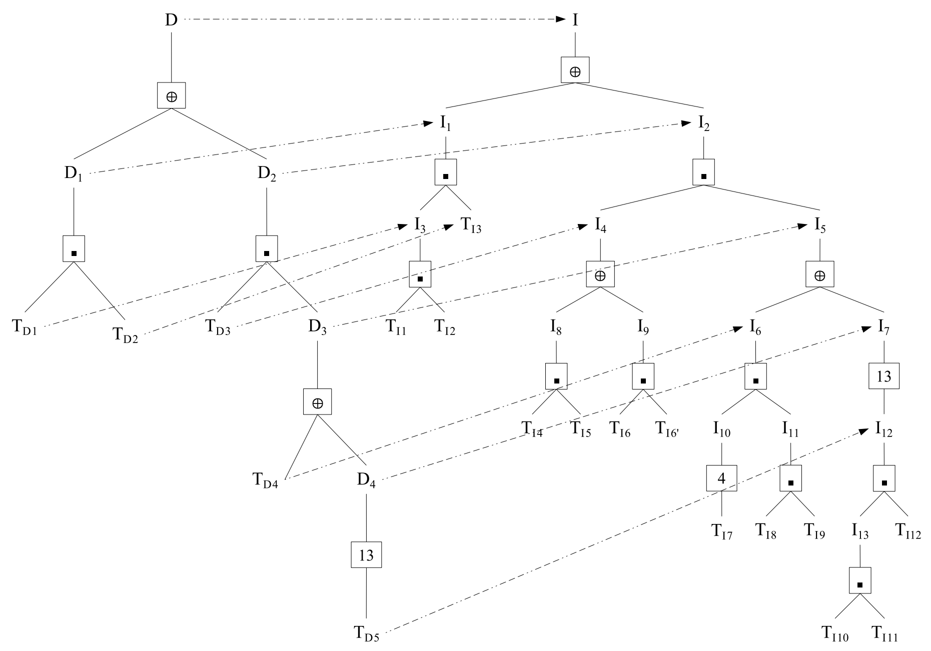

Fig. 1. Design (left) and implementation (right) of example presented in Sect. 4.1

T_{I5}	:	(`Excel`, DAPest.sas, `Excel`),
T_{I6}	:	(`Excel`, graphing to print.sas, `Bitmap`),
$T_{I6'}$	:	(`Bitmap`, convert2excel.exe, `Excel`),
T_{I7}	:	(`Excel`, reg.sas, `Excel`),
T_{I8}	:	(`Excel`, merge datasets.sas, `Excel`),
T_{I9}	:	(`Excel`, proc mixed.sas, `Excel`),
T_{I10}	:	(`Excel`, pho341.sas, `Excel`),
T_{I11}	:	(`Excel`, pho342.sas, `Excel`),
T_{I12}	:	(`Excel`, broken3.sas, `Excel`).

We define now protocol implementation I with input In(I) = `Excel` and output Out(I) = `Excel` as $I = I_1 \oplus I_2$, with $I_1 = I_3 \centerdot T_{I3}$, $I_2 = I_4 \centerdot I_5$, $I_5 = I_6 \oplus I_7$, $I_3 = T_{I1} \centerdot T_{I2}$, $I_4 = I_8 \oplus I_9$, $I_8 = T_{I4} \centerdot T_{I5}$, $I_9 = T_{I6} \centerdot T_{I6'}$, $I_6 = I_{10} \centerdot I_{11}$, $I_{10} = {T_{I7}}^4$, $I_{11} = T_{I8} \centerdot T_{I9}$, $I_7 = I_{12}^{13}$, $I_{12} = I_{13} \centerdot T_{I12}$ and $I_{13} = T_{I10} \centerdot T_{I11}$. The protocol implementation is illustrated in the right-hand side of Fig. 1.

Now we are ready to define the mapping between protocol design D and protocol implementation I. We define $\varphi_{\mathcal{C}}$ as a mapping on $\mathcal{C}$ with $\varphi_{\mathcal{C}}(\textbf{SeedData}) =$ `Excel`. The protocol design mapping is represented in the picture by dashed lines. Finally, the whole protocol is defined as $(D, \{\varphi_{\mathcal{C}}, \varphi_{\mathcal{T}}\}, I)$.

5 ProtocolDB

ProtocolDB is composed of an Access database as the back-end repository on the server-side. The server side components also include the Microsoft Internet Information Server (Version 5) and the Apache Tomcat Web Server (Version 5.5.16) to handle user requests. The user interface interacts with these sub-systems to provide the necessary storage and retrieval functionalities. HTTP connectors and JDBC-ODBC connectivity functionality are used to communicate with the database to perform the operations on the repository.

The schema is composed of five tables. The table `registered_users` stores the information related to the users of the system. The primary key for this table is the attribute `nickname`. Once a scientist is registered, each entry or modification of a protocol will be recorded with the scientist's information. Information pertaining to the protocol as a whole, including its name, scientific aim, date and time of last edition are stored in `protocol_info`. The attributes `DateSaved`, `TimeSaved` and `ProtocolNickname` form the composite primary key for this relation. Design steps of protocols are stored in the table `Design_Steps`. The attributes `StepNumber`, `DateSaved`, `TimeSaved`, and `ProtocolNickname` form the composite primary key for this relation. Similarly, implementation steps are stored in the table `Implementation_Steps`. The structure of the protocol consisting of the successor connector "$\bullet$" and the split-merge connector "$\oplus$" is recorded in table `connection_between_steps`. The attributes `from_step`, `to_step`, `ProtocolNickname`, `DateSaved`, and `TimeSaved` form the composite primary key for this relation. The foreign key of each of the relation `Design_Steps`, `Implementation_Steps`, and `connection _between_steps` is linked to the primary key (composite primary key) of the `protocol_info` relation to link the information corresponding to a particular protocol distributed among the different relations in the database.

6 Conclusion

The model for scientific protocols proposed in the paper aims at representing the general structure of scientific protocols, with explicit distinction between design and implementation. It allows for comparison of alternative implementations and resources, an approach that is compatible with path-based guiding systems [16]. On-going and future work includes the extension of the model to allow the storage of collected data sets, for support of cross protocol-data queries and reasoning on data provenance.

This model is used to develop ProtocolDB, a repository of scientific protocols where scientists can store, retrieve, compare, and re-use scientific protocols. The system is currently under development and will be available shortly at:
`http://bioinformatics.eas.asu.edu/protocoleDatabase.htm`.

Acknowledgments This research was partially supported by the National Science Foundation grants IIS 0612273, IIS 02230042 and IIS 0222847, and the NIH National Library of Medicine grant R03 LM008046-01. ProtocolDB was implemented by Phanindra Dev Deepthimahanthi. We thank Jeff White, U.S. Arid-Land Agricultural Research Center (ALARC), Mike Berens, Anna Joy, and Dominique Hoelzinger, Translational Genomics Research Institute, Jan Van den Bussche, Hasselt University, and Jan Hidders, University of Antwerp, for their valuable input.

References

1. Lawson, A.: Studying for Biology. Addison-Wesley Educational Publishers (1995)
2. Stevens, R., Goble, C.A., Baker, P.G., Brass, A.: A classification of tasks in bioinformatics. Bioinformatics **17**(1) (2001) 180–188
3. Bartlett, J.C., Toms, E.G.: Developing a Protocol for Bioinformatics Analysis: An Integrated Information Behavior and Task Analysis Approach. Journal of the American Society for Information Science and Technology **56**(5) (2005) 469–482
4. Tröger, A., Fernandes, A.: A language for comprehensively supporting the in vitro experimental process in silico. In: Fourth IEEE Symposium on Bioinformatics and Bioengineering (BIBE 2004). (2004) 47–56
5. Chen, I.M.A., Markowitz, V.M.: An overview of the object protocol model (opm) and the opm data management tools. Inf. Syst. **20**(5) (1995) 393–418
6. Ailamaki, A., Ioannidis, Y.E., Livny, M.: Scientific workflow management by database management. In: SSDBM. (1998) 190–199
7. Shankar, S., Kini, A., DeWitt, D.J., Naughton, J.: Integrating databases and workflow systems. SIGMOD Rec. **34**(3) (2005) 5–11
8. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics Journal **20**(17) (2004) 3045–3054
9. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific Workflow Management and the Kepler System. Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows (2005) (to appear).
10. Ludäscher, B., Altintas, I., Gupta, A.: Compiling abstract scientific workflows into web service workflows. In: SSDBM, IEEE Computer Society (2003) 251–254
11. Medeiros, C., Alcazar, J., Digiampietri, L., Pastorello, G., Santanche, A., Torres, R., Madeira, E., E, B.: WOODSS and the Web: annotating and reusing scientific workflows. SIGMOD Record **34**(3) (2005) 18–23
12. Hashmi, N., Lee, S., Cummings, M.: Abstracting Workflows: Unifying Bioinformatics Task Conceptualization and Specification Through Semantic Web Services. In: W3C Semantic Web for Life Sciences position paper. (2004)
13. Foster, I., Voeckler, J., M.Wilde, Y.Zhao: Chimera: a virtual data system for representing, querying and automating data derivation. In: SSDBM. (2002) 37
14. Zhao, Y., Dobson, J., Foster, I., Moreau, L., Wilde, M.: A notation and system for expressing and executing cleanly typed workflows on messy scientific data. ACM SIGMOD Record **34** (2005) 37–43
15. Hidders, J., Kwasnikowska, N., Sroka, J., Tyszkiewicz, J., Van den Bussche, J.: Petri net + nested relational calculus = dataflow. Technical Report TR UA 2006-04, University of Antwerp, Belgium (2006)
16. Cohen-Boulakia, S., Davidson, S., Froidevaux, C., Lacroix, Z., Vidal, M.E.: Path-based systems to guide scientists in the maze of biological data sources. Journal of Bioinformatics and Computational Biology (2006) (to appear).