A Language for Comprehensively Supporting the *In Vitro* Experimental Process *In Silico*

Ane Tröger and Alvaro A.A. Fernandes
Department of Computer Science
University of Manchester
{a.troger|a.fernandes}@cs.man.ac.uk

Abstract

One of the challenges for bioinformaticians is to approximate, in silico*, tried and tested research methods used* in vitro*. One of the problems standing in their way is the lack of a concrete framework for designing and expressing* in silico *experiments that aims at being isomorphic to* in vitro *experiments. This paper introduces such a framework in the form of a specification language called* ***ISXL****. ISXL projects to biologists a model of* in silico *experiments that approximates the research method they are most familiar with, as follows. An ISXL-specified experiment (1) conforms to a conceptual model that explicitly captures the basic constituents of experiments in the empirical sciences; (2) may be defined in relation to explicit hypothesis formulation and validation rather than simply taking the form of an evidence gathering process as in alternative approaches; (3) may be long-lived and evolve over time, in the sense that there is built-in support for denoting past versions of specifications, past results, past hypotheses, past validation criteria; (4) may denote other experiments and their constituent parts, thereby reflecting the interrelatedness of scientific processes. Features (1)-(4) above are made possible by endowing ISXL with certain characteristics of a* persistent workflow environment*. This allows ISXL experiments to be rich in metadata without imposing too great a burden on the biologist. The metadata in turn open the way for ISXL experiments to be capable of introspection and reflection. This paper focuses on describing of ISXL conceptually and syntactically, and indicates how ISXL experiments are given a formal semantics.*

1. Introduction

There is great awareness that data integration is, and is likely to remain for the foreseeable future, a central concern in bioinformatics. Much progress has been made in addressing that concern [4, 1]. More recently, some researchers have moved on to tackle the problem of process integration as well [2, 13]. This later development benefits greatly from advances in both workflow technology and service orchestration. The combined effect of these advances is to provide biologists with the conceptual means to cast part of their experimental protocols in computational form. In this form, such protocols are referred to as ***in silico* experiments**. This paper postulates that referring to such constructs as *in silico* experiments is an overstatement insofar as the computational form they assume is far from isomorphic to the form a comparable *in vitro* activity would take. They are, at best, a computational model of the evidence-gathering stage of the *in vitro* experimental process, and hence, only partially isomorphic to the latter. The linguistic framework contributed in this paper addresses this concern by providing explicit support not just for evidence gathering (based on data *and* process integration) but also for linking that evidence gathering with hypothesis formulation and validation. Moreover, the framework is conceptualized as a persistent environment. This means that an experiment is long-lived and interrelated with other experiments. Thus, it can refer to its previous specifications and the results obtained from those as well as to those of other experiments. This means that experiments specified in the proposed language, called **ISXL**, are not merely self-centred, transient computational entities, rather they can evolve in response to their own history and that of other experiments it relates itself to.

The remainder of the paper is structured as follows. Section 2 explains in more detail the main motivations underlying this paper, the challenges arising from them, and the contributions reported in the form of specific responses to the challenges identified. Section 3 introduces a motivating scenario based on distant homology detection. Section 4 describes how ISXL experiments provide the working biologist with support for binding conceptually and concretely the core evidence-gathering specification with the hypothesis for which the evidence is sought and with the validation process that concludes as to the level of support that the gathered evidence lends the hypothesis. Section 5 describes the mechanisms behind the support for long-lived, evolving, interrelated *in silico* experiments, exemplify their use in the language and suggests the benefits that then accrue. Section 6 provides, within the constraints of the space available, a description of the most salient syntactic and semantic aspects of ISXL. Section 7 situates the contributions of the paper against the background of related work. Section 8 concludes by discussing the potential impact of the contributions presented.

2. Motivation, Challenges, Contributions

The main **motivation** for the contributions reported in this paper stems from the broad aim of replicating *in silico* the experimental method carried out *in vitro*, as closely as it is possible, and useful, to do so. The goal is to provide biologists with computational mechanisms that allow for *in silico* experiments to be sufficiently isomorphic to *in vitro* ones that the transition from one experimental context to the other is not unduly burdensome. This is a step in the direction of fulfilling some of the crucial promises implied by the use of phrases such as *e-science* to denote the growth in *in silico* experimental practices.

Consider the following observations: most bioinformatics tools for process co-ordination provide only (1) a computational model of the evidence-gathering stage of the experimental process; and (2) the means to carry out experiments that are isolated and atemporal. However, scientific practice involves: (a) not just evidence gathering but relating the latter to an explicitly formulated hypothesis (for which the evidence is presumably collected) and an explicitly specified validation process through which one can conclude which level of support the gathered evidence lends to the hypothesis; and (b) relating the current expression of the experiment with previous ones, the current results with previous results, the current formulation of the hypothesis (and its associated validation) with previous ones, both of this very same experiment as it evolves over time as well of other experiments with which it is, or should be, explicitly related. Therefore, experimentalists current lack expressive support for *(a)* and *(b)* above. Granted that technologies such as lab information management systems (LIMSs) do offer some support in this respect, but they also often suffer from lack of integration with the computational artifiacts by means of which evidence is gathered. The contributions reported in this paper aim at improving matters for biologists with respect to *(a)* and *(b)* above in a more integrated manner.

The motivation above gives rise to **challenges** that can be stated as follows: (C.1) how to conceptualize, from the viewpoint of *in vitro* experimental practice, what current computational models of *in silico* practice lack? (C.2) how to support, concretely, the extended *in silico* computational models of *in vitro* practice that such a conceptualization task identifies?

The classical components of an *evidence-gathering* activity in the empirical sciences are *materials* (e.g., a probe s and a biological sequence database D), *methods* (e.g., sequence alignment) and a *protocol* (e.g., 'apply the latter to the former'). The specific materials, methods and protocol used can only be decided upon in light of a specific *hypothesis* (e.g., s is involved in pathology p) in support of which the evidence is sought. The degree of support provided by the evidence gathered is calculated by some *validation* procedure (e.g., support is *high* if the sequence similarity score of s is above some threshold e against sequences $s_1, \ldots, s_n \in D$ known to be involved in p for some threshold value n).

For a claim of isomorphism between *in silico* and *in vitro* practice to be credible, a computational framework is required that can: (R.1) express explicitly the hypothesis as an operational procedure; (R.2) specify precisely the evidence-gathering process to be used in obtaining the data needed to associate some degree of support to the hypothesis; (R.3) specify precisely how the data obtained in the evidence-gathering stage map to a specific degree of support for the hypothesis; and (R.4) satisfy *(R.1)* to *(R.3)* under a model of scientific practice that recognizes that experiments are neither isolated nor atemporal, i.e., they refer to past experiments and the past states and results of those experiments.

The **contributions** reported in this paper constitute a specific, effective, well-integrated response to challenges *(C.1)* and *(C.2)* above, as follows:

1. An *in silico* experiment language, ISXL, is proposed whose underlying conceptual model encompasses not just evidence gathering but also the explicit statement of the hypothesis the evidence is gathered for, and of the procedure used to validate that hypothesis by computing the degree of support that the evidence gathered confers to the latter. This satisfies requirements *(R.1)* to *(R.3)* above.

2. Implicit persistence mechanisms for ISXL-specified experiments are provided that enable the language to offer explicit constructs which denote (and hence allow a biologist to refer to) past versions of an experiment, other experiments (and possibly past versions of these), and past results obtained by enacting any of the above. This satisfies requirement *(R.4)* above.

These contributions are described and discussed in detail in Section 4 and Section 5, respectively. More detail on the more salient syntactic and semantic aspects of ISXL is provided in Section 6. The consideration of the benefits that accrue from them is best done in the context of a motivating example that Section 3 now introduces and describes.

3. Motivating Example

As a motivating example, consider the problem of hypothesizing protein function (**PF**) from biological sequence analysis. **PF** remains an important problem in spite of the intense recent activity in bioinformatics. It is also a fairly complex problem in two senses at least. Firstly, from a purely biological point of view, the basic scientific knowledge required is not yet established in the minute detail needed. In view of this and other difficulties, bioinformaticians have devoted a significant amount of effort to developing tools for analysis whose combined use, it is believed, increases the tractability of the problem. However, here too the complexity is daunting, insofar as, from a purely computational point of view, the integration of data and process at the grain required is a complex problem. This is compounded by the fact that the topology of the computation (i.e., the flow graphs over both data and process nodes) is quite complex too.

Consider, for simplicity, a sub-problem of **PF**, viz., that of detecting distant homology (**DH**). The following is one possible **DH** protocol: (1) given a probe, perform a search over a sequence database to retrieve matches; (2) perform a filtering step; (3) if more than one match survived the filtering, then perform multiple sequence alignment, otherwise, perform pairwise alignment; (4) perform profile generation over the resulting alignment; (5) using the profile as probe, perform a search over a sequence database to retrieve matches; (6) if no new matches were found, then the profile and the sequences in the previous pass are the solution, otherwise perform multiple sequence alignment over the profile and the new matches; (7) go back to performing profile generation over the resulting alignment.

In the **DH** problem, implementing the protocol means the automation of the gathering of evidence but this does not imply support for the explicit validation of an explicitly formulated hypothesis that supposedly explains the evidence generated by the protocol. Sequence similarity is evidence for homology but does not imply it. This means that the evidence gathered by performing the **DH** protocol above must be tied to some explicitly formulated hypothesis that is meant to explain that evidence. This hypothesis must then be subject to explicit validation by some well-founded, principled process. Currently, bioinformatics offers adequate automated support for evidence gathering while the validation of process of the hypothesis that is meant to explain the evidence gathered is largely done off-line, e.g., using statistical tools (and one is back to a dependence on expert software engineers for composition and front-ending). Note, however, that the magnitude of the need (and the difficulty of managing it by hand at this scale) is compounded many times when one moves from each sub-problem to the full **PF** problem. Moreover, automating the protocol automates the execution of an experiment but this does not imply support for the dynamics of the scientific process (which requires records of lineage and provenance for all of methods, materials and protocols to be explicitly kept and cross-referenced). The need for addressing the support for the scientific process can be seen in the **DH** problem in the fact that different methods (e.g., different approaches to alignment) may produce different results, and it is an essential part of the scientific process that different protocol formulations are attempted and that the trail of such attempts is kept in mind all the time for the best scientific knowledge to be attained. Here too, bioinformatics offers little support, thereby relegating biologists to the use of time-consuming (and methodologically brittle) off-line or decoupled methods (e.g., from lab books and personal file stores to fully-fledged LIMSs).

In this case, which is, in the relevant respects, typical of many, a biologist would only be partially supported in her goal of approximating *in silico* the experimental practices she is used to *in vitro*. This is because most *in silico* frameworks only offer partial isomorphism to *in vitro* practices. In contrast, the conceptual model that underpins ISXL allows for the specification of *in silico* experiments, in which the constituent parts of the experimental process can be explicit stated and are logically bound. Moreover, unless the specification, (possibly) the values consumed and produced in each enactment, and the sequence of protocol steps taken (i.e., the trace of each enactment) are kept in a structured, denotable manner, she will have to generate from the *in silico* activity a significant amount of metadata by hand (e.g., for lab book entries, for provenance records, etc.). However, those metadata are but a by-product of the *in silico* process, and hence are derivable from it in automated fashion.

ISXL is a persistent workflow language. It can, therefore, both collect, and, later, resolve references to such metadata as are associated with, or derivable from, the enactment of experiment specifications. Also, ISXL does such metadata management using versioning, and hence supports reference to past events in the experimental process. Finally, since persistence is supported by default, reference by one experiment to any other experiment in the ISXL persistent repository is supported too.

The benefits of the contributions reported in this paper can thus be seen to be associated with the greater degree of isomorphism to *in vitro* practices that is achievable by ISXL-specified *in silico* experiments. Scientific experimentation is a collaborative activity of significant duration. The knowledge discovery process that underlies scientific activity can become laborious, especially in regard of the need to keep information about how hypotheses acquire or lose support. The associated metadata are both voluminous *and* functional, in the sense that scientists often reason with their records of the experimental process, not just with the evidence that it produces. How and when an experiment was conducted, with which materials and methods, what was the protocol followed and the results achieved, are all important for the understanding of options open and closed.

Sections 4 and 5 provide more detail as to how ISXL supports biologists in the ways suggested in this section via the motivating example just introduced.

4. Contextualizing Evidence Gathering

It seems that the idea of using workflows in bioinformatics is gaining credence in the community [2, 13]. The reason seems to be that workflow technologies lie at a convenient abstract level for the representation of (and a suitable enactment environment for) data integration and process coordination. ISXL follows this workflow approach: it is, therefore, essentially a workflow specification language with an underlying target enactment environment. The workflow approach, in and of itself, goes a long way towards capturing the constituent parts of *in vitro* experiments as *in silico* processes.

Basically, a workflow models a process as a composition of smaller units of work (i.e., individual processing steps typically referred to as *tasks*). Whenever there is a data flow connection between two tasks (e.g., alignments being passed to a profile generator), a data dependency arises to which one can also attach constraints (e.g., to wait, or not, until the data stream is entirely produced before it begins to be consumed). Task coordination is specified by control flow connections (e.g., start two tasks concurrently). Here, again, a control dependency arises which, again, can be constrained (e.g., do not start until handles on the necessary ports have been secured). The data and flow dependencies define a topology for the process. Runtime circumstances determine which data and control flow constraints are satisfied and when. From the runtime circumstances data and control traces (i.e., paths through the data and control flow graphs) arise [12, 7, 6].

Bioinformaticians stand to benefit from using workflow technology as a model of the *in silico* experimental process. The constituent part of *in vitro* practices that a workflow models most visibly is the evidence gathering stage of that process, as follows. **Evidence gathering** is described by a **protocol** that relates the **materials** used in the experiment with **methods** that somehow process those materials and produce new ones as results. The relationships established by the specified protocol are captured in a workflow by the data and control flow dependencies and constraints. Each method is defined as a processing unit, whose signature comprises the work to be done (as a binding to an internal or external service) by consuming the content of in-trays (i.e., metaphorically, the container for input materials) and producing content for out-trays (ditto, for output materials).

Figure 1 shows an excerpt of the ISXL specification of the motivating example in Section 3 where the syntax for the evidence-gathering stage is applied to the example from the previous section. Materials and methods are declared in their respective sections of the specification. The protocol declares the **coordination rules** for the computation in two groups: one for control flow and another for data flow. The operational semantics of ISXL is defined in terms of state-transition diagrams for methods and trays, as described in Section 6. Briefly, Figure 1shows part of the ***evidence_gathering*** block in ISXL. The definition of methods `M1` and `M2` (the first two steps in the motivating example) is shown. Each method has its associated ***task*** (in this case, one example is of a remote service, the other uses an ***eval*** call[1]), as well as ***inTrays*** and ***outTrays***.

The ***protocol*** is expressed as a set of coordination rules. In this example, `M1 -> M2` stipulates that the completion of the `M1` method enables the `M2` method to start executing. That `M1` also has a data dependency with `M2` is indicated by the data flow rule `M1.probe_results -> M2.probe_results`, indicating that once the `probe_results` out-tray of `M1` is full, its contents can flow into the `probe_results` in-tray of `M2`.

Now, the evidence gathering stage needs to be logically bound to a **hypothesis** in whose support the evidence is gathered and the hypothesis, in turn, is logically bound to some **validation** procedure, i.e., an assessment of the degree to which the evidence gathered does lend support to the hypothesis. To the best of our knowledge no other workflow approach to bionformatics has accounted for this logical binding of the evidence gathered with the other constituents.

This is surprising because the validation procedure is, by definition, representable as a process. Moreover, a hypothesis can (and should) be given an operational form (typically, as the specification of a relationship) and hence, can also be expressed as a process (essentially one that ascertains whether the relationship is true of the entities involved). What distinguishes these two processes from the one that models the evidence gathering stage is their methodological status: to be properly contextualized, the evidence gathering process must be logically bound to the corresponding hypothesis and validation procedure. If so, it would be expected that proposals for workflow languages in bioinformatics would also express hypotheses and validation procedures as processes, especially because this allows the language to enforce the logical binding between the constituent parts of an experiment rather than overburdening the biologist with the administrative task of ensuring that this methodological requirement is satisfied. This is, therefore, one of the contributions of ISXL, viz., it confers the hy-

1 ***eval*** calls are motivated in Section 6.

pothesis and validation procedure that correspond to the evidence gathering stage a methodological status that the underlying mechanisms can then take upon themselves to enforce.

```
evidence_gathering
 materials
  probe
 end
 methods
  M1 ( # given a probe, perform a search
   task: FASTA@http://www.ebi.ac.uk/fasta33/
    inTrays: probe
   outTrays: probe_results
  ),
  M2 ( # filtering step for G+C content
   task: eval(python,"filterForGCContent(60)")

    inTrays: probe_results
   outTrays: results
  ),
#... other methods
end
protocol
 control flow
  :   -> M1;
  : M1 -> M2;
  if eval(python,"cardinality(results) > 1")
  : M2 -> M.3.1;
  #... other control flow rules
 end
 data flow
  probe -> M1.probe;
  M1.probe_results -> M2.probe_results;
  #... other data flow rules
  end
 end
end
```

Figure 1. Evidence-Gathering in ISXL

One could argue that the same functionality could be achieved in other workflow languages by simple compositional mechanisms. However, the logical relationship would still have to be specified in the language if it is to be enforced. This relationship being, at the appropriate abstraction level, the same for all experiments, it would be wasteful and error prone to specify it on a per-experiment basis: its proper locus is the language itself, i.e., the compilation process should ensure that an executable mechanism is generated that enforces the relationship. Although the syntactic cues for the compiler are simple, the beneficial consequences are quite significant. Thus, ISXL conforms to a conceptual model in which the logical relationship between constituent parts of an experiment is explicitly stated (and hence can be automatically enforced by the compiler, just as type relationships routinely are). Figure 2 depicts the conceptual model underlying the language:

The advantage of such a conceptual model is that the semantics of the integrated specification of experiments as workflows are enforced without requiring the user to implement them, as would be needed in general-purpose programming languages or other workflow languages. The import of the conceptual model in Figure 2 is, therefore, that in order to bind the constituents logically, all that is needed is to relate them syntactically.

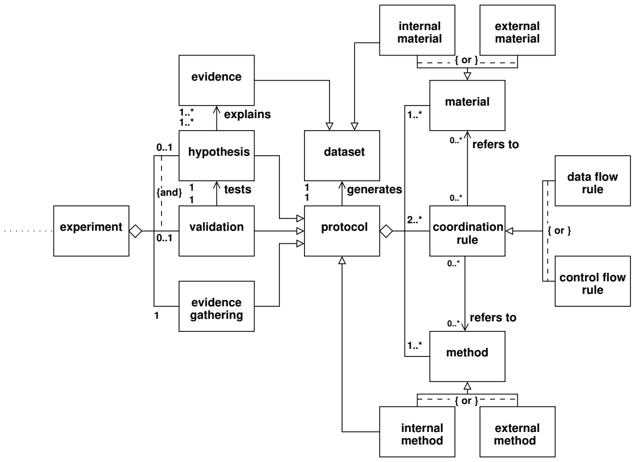

Figure 2. ISXL Conceptual Model

Figure 3 shows another excerpt of the ISXL specification of the motivating example in Section 3. The syntactic cues represented by the explicit declaration of the materials involved, together with the explicit statement of an operational hypothesis and a validation procedure imply the need for the ISXL compiler to generate the mechanisms through which the logical relationships expressed in the conceptual model are enforced at runtime.

The hypothesis is an executable specification that wraps the evidence and conjectures that some property of the evidence gathered holds. In the example, the conjecture might be that "*no profile with G+C content above 60% is longer than 15 letters*". The validation is an executable specification that wraps the hypothesis and assesses its validity, typically over a sample (for example, one might count the true and false predictions over a given sample).

Upon processing the hypothesis specification, the ISXL compiler can generate a partially instantiated procedure (e.g., using programming techniques such as currying). This partially instantiated procedure can then be invoked in the validation specification by the mere mention of the ***hypothesis*** keyword. The compiler output for the validation procedure makes sure that the context provides the necessary arguments to completely determine the curried function, so that it can be called and return a truth value. In the Python code in Figure 3, a list comprehension expression is used to traverse a sample and identify the true predictions. This can then be used to count the number of cases in which the hypothesis evaluates to true and false. Notice the way in which the ISXL compiler has enough handles (as well as expressive means) to enforce the logical relationship between hy-

```
experiment DH_filtered_for_G+C_content
  inTrays: probe
 outTrays: profile, results
evidence_gathering
 #... omitted, see Figure 1
end
hypothesis
 # profiles are no more than 15 letters long
 eval(python,
  "length(DH_for_highG+C(probe).profile <= 15")
end
validation
 # count true and false predictions in a sam-
ple
 eval(python,
  "X = [hypothesis for probe in sample];
  True = X.count;
  False = sample.count - True")
 end
end
```

Figure 3. Binding Evidence and Hypothesis

potheses, the evidence associated with them, and the validation procedures that are used to ascertain the degree of support the hypotheses enjoy.

This section has shown how ISXL allows *in silico* experiments to enjoy a greater degree of isomorphism to *in vitro* ones than other workflow approaches to date have allowed. The benefits accruing manifest themselves in the lessening of the administrative burden on biologists coupled with assurance of methodological rigor. Section 5 describes how the design of ISXL has taken a further, important step from the definition of the conceptual model depicted in Figure 2 by instantiating it as (part of a) schema for a persistent repository. This opens the way for ISXL to have persistent process language aspects with benefits that Section 5 describes in detail.

5. Evolving, Interrelated ISXL Experiments

Referring to the constituent parts of past experiments and the evidence they produced is a common way to understand what has been achieved, and how, before one sets out to seek new knowledge. It is part and parcel of the experimental method to seek to improve upon past experiments by deriving, from their specifications and from the results obtained through them, new experiments that, in this sense, are versions of their predecessors, given that this approach structures experiments into lineages.

In *in vitro* experiments, there is normally a lab book where scientists write down the specification of experiments (comprising all of materials, methods and protocol, as well as the hypothesis and the validation results, albeit without a formal, regular structure), the results obtained each time the experiment was run, and, in crucial cases, provenance data and logs describing how the experiment proceeded. If, as a result of insights obtained, a new experiment suggests itself as the next step of what has now become an extended, multi-step investigation, then all this information in the lab book is available to inform the specification of the new experiment.

While it is true that great effort has been poured into automating the management of the metadata that lab books traditionally collect, integrating the metadata collected by such systems with an *in silico* experiment has not, to the best of the authors' knowledge, been attempted by many researchers (one exception being the ^{my}Grid project). The challenge in integrating collected metadata with an experiment is that of providing the means for the specification of the latter to refer to the former. At a technical level, this presupposes that the language engine (i.e., the execution environment) keeps state so as to provide constructs for referring to entities, say, in a persistent repository that the language engine itself is aware of, and hence knows how to access and use (as opposed to a generic, standard file system accessible to any process running in the file system host machine). Languages whose engines possess this ability are referred to as *persistent languages*. ISXL is a domain-specific persistent process modelling language for *in silico* experiments in biology. Therefore, it systematically and automatically collects and stores metadata not only about the results of experiments but also about past versions of such experiments.

The formal structure of the ISXL repository is given by the conceptual schema in Figure 2, augmented as indicated in Figure 4. It shows that ISXL embraces the idea that experiments can have many specifications (through what can be thought of as a versioning process) and that each of these specifications can be run many times, as a result of which metadata are kept (e.g., the materials used, the data and control traces, etc.).

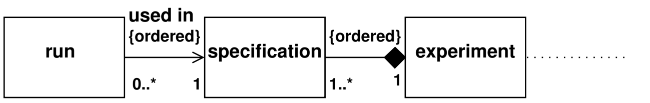

Figure 4. Extended ISXL Conceptual Schema

In silico experiments whose metadata are systematically kept can be referred to and can also evolve. This means that experiments can be derived to give rise to new (perhaps improved, perhaps alternative) versions. They can also be invoked as subcomponents of new specifications (for instance, the evidence-gathering part of one experiment can be reused with a different validation procedure for the same hypothesis, or perhaps for a hypothesis that was differently expressed in a previous version of the same, or another, experiment). The effect of these design decisions is the support for evolving, interrelated *in silico* experiments.

If reference to such metadata is integrated well into the expressive capabilities of the language, questions like the following can be asked:

1. What was the hypothesis in the first specification of this family of experiments?
2. How was the evidence gathering specified then and how was it obtained, as revealed in the control and data traces the last time it was run?
3. What were the results of run i of this experiment?
4. What other experiments involving the same materials and methods produced better degree of support for their hypotheses and what were these hypotheses?
5. Has the same evidence been gathered in support of one particular hypothesis but with different validation procedures?
6. Which one of those resulted in a degree of support above a certain threshold?
7. What is the complete lineage that has led to this experiment specification?

These questions can be converted into ISXL syntax that, when evaluated, yields an answer that can influence the outcome of the experiment in whose specification they occur. This capability is not easily replicated by recourse to non-persistent languages, because one would have to code not only the data management functions but, more challengingly, the retrieval and use of the stored metadata into the body of the experiment. In non-persistent languages, in order to write a program one is expected to have determined, precisely and in advance, the (type of) data the program will receive as input and produce as output. So, if biologists are specifying an experiment as a program in Perl, or in Java, or in any other non-persistent language, they must plan in advance all data and metadata that are to be *denotable* (otherwise, by default, it will not be), and then design and implement the appropriate repository. Only then, would experiments be capable of referring back to previous versions, thereby giving rise to a lineage relationship. This is by no means a trivial task even for skilled programmers, because it presupposes the possession of the software engineering skills necessary for the development of as sophisticated an infrastructure as ISXL provides out-of-the-box. The second contribution of ISXL described in this paper is, therefore, the support for long-lived, evolving experimental processes through the provision of linguistic constructs (and corresponding infrastructure) for cross-reference between experiments.

Figure5 shows an excerpt of an ISXL specification (not related to the motivating example in Section 3) where its support for cross-reference is exemplified. The example shows a data flow rule that will send data to `someMethod.inTray`. If `run01` of specification version `specVers1` (by default, of this experiment) has collected more sequences than `run01` of specification `specVers2` (ditto) in their respective evidence-gathering stages, then `specVers1` provides `someMethod.inTray` with those sequences, otherwise not.

```
data flow
 [if (length(specVers1.run01.evidence)) >
   length(specVers2.run01.evidence))] :
   specVers1.run01.evidence ->
      someMethod.inTray;
 #... other data flow rules
end
```

Figure 5. Example Cross-Reference in ISXL

The syntax to access elements of experiments is similar to file system addressing and can be translated to XPath expressions (as indicated more precisely in Section 6).

Figure 6 contrasts the abstract architecture of a classical compiler with that of ISXL. The top half shows that non-persistent programming languages must take specific steps to achieve persistence. In contrast, the bottom half shows the extent to which ISXL can be said to be persistent, viz., both the compilation and the enactment processes leave traces in a repository whose contents language constructs can denote. This can be achieved by any of many persistence bindings available (associated either to a general-purpose programming language, e.g., as in JDBC or JDO for Java, or to a data model, e.g., as in the ODMG standard for object-oriented databases).

ISXL source code is first mapped to a canonical representation (that gives that source a regular structure). Then, if it constitutes a new version of an existing experiment, that canonical form is added to the lineage of the experiment that it versions and that resides in a repository concretely instantiating the ISXL conceptual model (in Figures 2 and 4). The canonical representation is then compiled into the workflow language whose enactment engine is used to endow ISXL specifications with an operational semantics. The compiled version of the canonical form of an ISXL specification is also stored in the repository. Finally, metadata generated by each run of a specification are also kept. This comprises the data and control traces of the run, i.e., precisely which materials and methods were used and which control and data flow transitions took place. Any results (i.e., evidence and degree of support) are also kept[2]. If ISXL specifications are enacted more than once, the run metadata are stored in association with the version of the

2 Clearly, this is not to be understood as the keeping of replicas of datasets but rather of references to them. So, ISXL experiments can be rerun but not, strictly speaking, necessarily replicated.

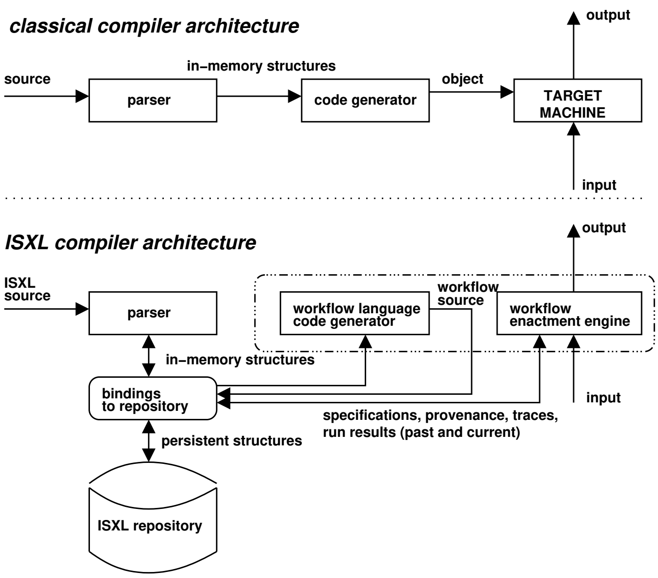

Figure 6. ISXL Compilation Strategy

specification that was enacted. Run metadata, therefore, accumulate (rather than overwrite the most recently stored), which allows previous results to be referred to in the language, and hence influence both the course of the evidence gathering (or of the hypothesis validation) or the functional design of future versions, for example.

The combined effect of the facilities arising from the persistence model adopted in the design of ISXL is to support the formulation of questions (1) to (7) above as ISXL expressions that can be evaluated at run time. These facilities for persistence with introspection allow ISXL-specified *in silico* experiments to mirror more closely the way in which, *in vitro*, scientists proceed tentatively towards their goal of generating new scientific knowledge.

As pointed out above, the ISXL repository organizes experiments into lineages that represent, in effect, the network of relationships between an experiment and its evolving history, as well as between one experiment and many others. Through lineages that share specification components in the repository, the connectedness of scientific investigations over (networked) space and over time is more faithfully captured. This is, therefore, another way in which it can be said that ISXL-specified *in silico* experiments come closer to the ideal of being isomorphic to *in vitro* ones.

6. Syntactic and Semantic Aspects of ISXL

This section provides additional information on some salient syntactic and semantic aspects of ISXL with a view towards indicating how any concrete implementation of the language builds upon an array of techniques and technologies. In particular, the section indicates those that the authors are deploying to build a proof-of-concept implementation of the language.

Syntactically, an ISXL specification of an *in silico* experiment must contain an evidence gathering section, and may contain, as a pair, a hypothesis and the specification of how to validate it. The evidence gathering section contains declaration of materials (e.g., data services) and methods (e.g., computational services), and a specification of the protocol to be followed. Methods map to stateful tasks, whose inputs and outputs are also stateful. The latter are referred to as in-trays and out-trays, respectively.

The protocol takes the form of a set of control flow rules and a set of data flow rules. A control flow rule has in the antecedent a guard, i.e., a Boolean-valued expression over task and tray states (and tray values), and in the consequent, an action that effects transitions in task and tray states (and tray values). A data flow rule is only different in that its action typically effects the transport of data between trays.

In Figure 1, in the control flow specification, given methods M and M', the expression $M \rightarrow M'$ is syntactic sugar for M `= finished:` M' `:= enabled`, meaning that M' can execute once, and only once, M has finished executing. In Figure 1, `M1 -> M2` is an example of the type of rule just described. Analogously, in the data flow specification, given trays T and T', the expression $M.T \rightarrow M'.T'$ is syntactic sugar for $M.T$ `= full:` $M.T$ `->` $M'.T'$, meaning that once, and only once, the out-tray T in M is full, its content can be transferred to the in-tray T' in M'. In Figure 1, `M1.probe_results -> M2.probe_results` is an example of the type of rule just described. The above examples suggest how flows of control and data occur (and how enactment progresses) as a result of transitions in the state of tasks and trays, respectively. In other words, the abstract semantics of ISXL is specified with respect to state transition diagrams associated with tasks and trays. This engine-independent semantics is based on the evaluation of the control and data flow rules until a quiescent state is reached[3]. Thus, the consequent of control and data flow rules affects the state of tasks and trays as prescribed by the separate state-transition diagrams depicted in Figure 7.

However, as is the case with most sufficiently expressive languages, the semantics of ISXL is more informatively formulated as being relative to an execution environment. This concrete semantics of an ISXL specification is given by translation to a process modelling language with an associated enactment engine. ISXL, therefore, does not come

3 If a quiescent state is reached at all, that is, because ISXL specifications are much too expressive for termination and confluence to be guaranteed. Undoubtedly, the development of static analysis tools would be useful future work to identify, whenever feasible, that a specification might lead to non-termination.

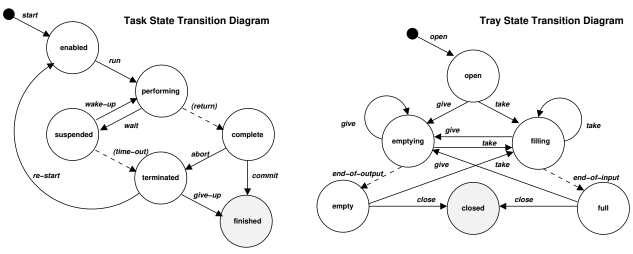

Figure 7. State Transition Diagrams

equipped with its own enactment engine and is not, in principle, tied to any specific engine.

In the current proof-of-concept implementation, ISXL compiles into the PML process modelling language as enacted by a PWI (for Process Wise Integrator) engine [5, 8]. The PML/PWI environment is a modern, object-oriented industrial-strength and state-of-the-art process modelling and enactment platform whose most distinctive (and distinctly useful) feature is its seamless support for process evolution via orthogonal persistence mechanisms. Given the requirement to support long-lived, evolvable *in silico* experiments, the PML/PWI environment is particularly suitable as a compilation/ enactment target for ISXL.

The ISXL design refrains from implementing within-language support for any data types, and hence the language does not come with any operations to side-effect state, other than via coordination rules (which are constrained to operate on task and tray states). This is consistent with the intention of coordinating *coarse-grained* processes that map to fully-fledged, highly-aggregated services. Nevertheless, ISXL provides syntax for source code Σ in some general-purpose language Λ, to be evaluated. This is done by means of tasks that rather than being defined via a reference to a service are defined by an `eval(Λ, Σ)` statement. It is part of the ISXL compilation strategy to pair up the compiler with both one target process enactment engine (or more) and one general-purpose language (or more). With respect to the latter, of course, an interpreted language makes this binding easier, but this is not mandatory: one might expose a compilation/execution server as a service for most modern non-interpreted languages. For instance, in the running example, Python is used, so the filtering step `M2` might map to a call to a function that processes a list of sequences and returns that subset of it whose elements all satisfy the filtering property.

As described in Section 5, ISXL support for long-lived, evolvable *in silico* experiments is founded on a persistent repository. The current proof-of-concept implementation of the repository uses Xindice, the Apache Software Foundation native XML database. Since there are many convenient approaches to map XML in-memory structures to persistent ones, the ISXL compiler can cleanly generate the code for metadata management. This option for an XML database also means that the concrete schema is written in XML Schema, that references to metadata map to XPath expressions, transformations to XSLT ones and side-effects to the repository to XUpdate commands.

7. Related Work

Many different motivations, raising intersecting (but not identical) issues to those stated in Section 2, have resulted in many proposals with many points of contact with the work described here and touching aspects of the language. To the best of our knowledge, there is no proposal that offers alternative approaches to all the functionalities exhibited by ISXL.

Bio* Open Sources [9] aims to support open source programming in bioinformatics. The approach is to specify programs as *in silico* experiments, using fine-grain abstractions for the bioinformatics domain. Many libraries have been made available for this purpose, e.g., BioPerl, BioJava and BioPython, offering greater fitness-for-purpose than their underlying languages. BioPerl, for example, captures concepts of the biological domain (e.g., data formats for sequences, Bio::Seq) and provides the computational abstractions for application developers to code methods that can be used as building blocks in bioinformatics process models (e.g., parsing analysis reports with Bio::SearchIO module) including process coordination (e.g., pipelining data with BioPipeline module). In contrast, ISXL aims to support the expression of *in silico* experiments as specifications that approximate the *in vitro* model with abstractions that are semantically closer to those that are familiar to experimental scientists. Bio* projects do not follow any conceptual model of the experimental process, as ISXL does, therefore they do not offer built-in support for *in silico* experiments as such.

With regard to long-lived, evolving, interrelated experiments, Bio* offers no built-in mechanisms to keep track of the necessary metadata: programmers themselves must design and implement the associated functionalities. For biologists to take on such tasks, they would need to be proficient software engineers, to the extent necessary for them to tackle the associated complexity, especially with regard to the infrastructure for the repository.

Some other proposals that bear similarity with the contributions reported here fall under the category of workbenches. A workbench aims to coordinate components that are more abstractly defined and bundle more aggregated, added-value functionality than simple querying interfaces and programming languages. The most prominent example is ^{my}Grid [3, 10, 11]. ^{my}Grid focuses on providing bioinformatics tool developers with a convenient platform with which to orchestrate services. There is no attempt, as of the time of writing, to approach in myGrid the degree of isomorphism achieved by ISXL. myGrid also keeps track of provenance data and provides a repository for all information needed for evolution, but does not provide explicit constructs to derive experiments from past experiments.

8. Conclusions

This paper has described some of the contributions of ISXL, centering, in particular, on its goal to allow the expression of *in silico* experiments that approximate better the structure and practices observed *in vitro*.

An ISXL-specified experiment (1) conforms to a conceptual model that explicitly captures the basic constituents of experiments in the empirical sciences; (2) may be defined in relation to explicit hypothesis formulation and validation rather than simply taking the form of an evidence gathering process as in alternative approaches; (3) may be long-lived and evolve over time, in the sense that there is built-in support for denoting past versions of specifications, past results, past hypotheses, past validation criteria; (4) may denote other experiments and their constituent parts, thereby reflecting the interrelatedness of scientific processes.

Some of the problems that such contributions help ameliorate include the following. For *in silico* experiments to be the backbone of e-science, they are required to be truer representations of scientific practice. Ideally, the way the experimental process is conducted should remain the same irrespective of the environment in which it is enacted, modulo the appropriate adjustments. Therefore, one should be entitled to expect to easily find the constituent parts of experiments logically bound in one single conceptual object, but only data and process integration have been so bound. The diversity of data sources and tools, as well as the amount of data generated by experiments, place a heavy burden on scientists. This takes the form of great costs in managing the data and metadata their experiments produce and of great loss in their ability to use this metadata in the specification of new experiments. For example, there are significant impediments at present for scientists to look back on the evolving nature of their investigations and to look forward to derive new versions from past experiments, when the process is being conducted *in silico*. This compromises their throughput in generating and assimilating new knowledge, whereas one would expect the opposite effect from the deployment of such powerful tools as modern computers are. So far, only provenance data has been kept, but scientists need to manage their data and metadata on their own because there is no support for experiments that are aware of their own history.

This paper describes the extent in which ISXL fulfils its goal of providing biologists with computational mechanisms that allow for *in silico* experiments to be sufficiently isomorphic to *in vitro* ones that the transition from one experimental context to the other is not unduly burdensome. In this respect, ISXL seems to fare much better in addressing (or not incurring) the problems above. Thus, ISXL is a step in the direction of fulfilling some of the crucial promises that have led to the growth in *in silico* experimental practices in biology.

Acknowledgement: The authors would like to thank Phillip Lord for useful discussions on the topics addressed in this paper.

References

[1] S.-Y. Chung and L. Wong. Kleisli: A New Tool for Data Integration in Biology. *Trends in Biotechnology*, 17(9):351–355, 1999.

[2] T. Oinn et al. *Taverna*. http://taverna.sourceforge.net/.

[3] A. A. A. Fernandes. myGrid: Grid Services for Data-Intensive Bioinformatics. In *Proc. Objects in Bio- & Chem-Informatics Conference*, 2002.

[4] C. A. Goble, R. Stevens, G. Ng, S. Bechhofer, N. W. Paton, P. G. Baker, M. Peim, and A. Brass. Transparent Access to Multiple Bioinformatics Information Sources. *IBM Systems Journal*, 40(2):532–551, 2001.

[5] M. Greenwood, D. Balasubramaniam, G. Kirby, K. Mayes, R. Morrison, W. Seet, B. Warboys, and E. Zirintsis. Reflection and Reification in Process System Evolution: Experience and Opportunity. In *Proc. 8th European Workshop Software Process Technology*, 2001.

[6] IBM. *WSFL: Web Services Flow Language*, 2002. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[7] IBM et al. *BPEL Specification: Business Process Execution Language for Web Services Version 1.1*. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

[8] ICL. *ProcessWise Integrator PML Reference: Version 4.1*, April 1996. ICL/PW/635/01 http://processweb.cs.man.ac.uk/doc/pmlRefPdf/.

[9] Open Bioinformatics Foundation. *Bio* Open Sources*. http://www.open-bio.org.

[10] R.Stevens, A. Robinson, and C.A. Goble. myGrid: Personalised Bioinformatics on the Information Grid. *Bioinformatics*, 19(Suppl. 1):302–304, 2003.

[11] R. Stevens, P. Lord, T. Oinn, and P. Li. Performing In Silico Experiments on the Grid Using myGrid. In *Proc. ISMB*, 2003.

[12] Wil van der Aalst and Kees van Hee. *Workflow Management - Models, Methods, and Systems*. MIT Press, 2002.

[13] M.D. Wilkinson and M. Links. BioMOBY: An Open-Source Biological Web Services Proposal. *Briefings In Bioinformatics*, 3(4):331–341, 2002.