Specifying *In Silico* Experiments as Coordinations of Coarse-Grained Processes

Ane Tröger and Alvaro A. A. Fernandes
Department of Computer Science
University of Manchester
Oxford Road, Manchester M13 9PL, UK
{a.troger|a.fernandes}@cs.man.ac.uk

Abstract A paradigm shift has occurred from centralized to distributed computing as the baseline upon which information systems are designed and implemented. This has raised the prospect of cooperation by coordination of computational, rather than physical, processes becoming the prevailing mode for large-scale systems. Just as, in business, e-business, soon, in science, e-science. The analogue of a business process in scientific contexts is an *in vitro* experiment, insofar as a major output of scientific practice is the knowledge obtained by the experimental method. Hence the concept of *in silico* experiments as the core component of a computational model of scientific activity now being referred to as *e-science*. Surprisingly, though, while the challenges presented by the ambition to coordinate e-business processes at very-coarse grains (often entire supply chains) are being seriously addressed, no comparable activity levels are yet discernible in e-science. This paper describes a platform for *in silico* experiments that is centred around a process coordination language called ISXL. ISXL projects to working scientists a model of *in silico* experiments that approximates more closely their *in vitro* practices than comparable efforts. An ISXL-specified experiment is (1) cast in a conceptual model that helps enforce the essence of the research method as applied to the empirical sciences; (2) may include explicit hypothesis formulation and validation (rather than simply the core functionality of evidence gathering); and (3) is long-lived, in that there is built-in support for denoting experiment specifications that evolve, as well as the results of past runs, which are, in turn, annotated and kept in an underlying persistent store. Here the focus is on (1) and (2), (3) is discussed in [7].

1 Introduction

In biology, diversity is often seen as a blessing, but most would agree that in bioinformatics the blessing is a mixed one, to put it mildly. A great deal of effort has been poured into removing impediments related to the heterogeneous, distributed, autonomous nature of data stocks in bioinformatics. This effort has yielded significant advances in data access and integration. In comparison, progress on coordinating the associated computational processes has been much more modest. An exception is query and retrieval functions, which are easier to make accessible in an integrated fashion because they are, or can without great effort be, projected out to users from the very software components that manage the data stocks. In other words, their closeness to the data simplifies the access and integration problem. Moreover, one could also argue that this limited success falls out from the fact that query languages can be seen as a declarative coordination languages, albeit with purposely constrained expressive power. When one broadens the scope to computational tools and systems that go beyond data access and integration to analysis, simulation and modelling tools, the landscape is noticeably barer. Success in data access and integration must, we claim, be

accompanied by advances in process coordination if the match between *in silico* and *in vitro* experimental practice is to be as significant as to justify the great hopes pinned on the e-science vision.

The contribution of this paper is the proposal of a platform for *in silico* experiments centred around a coarse-grained process coordination language called ISXL. The paper describes ISXL conceptually, syntactically, and semantically. It also indicates how we envisage it should be practically implemented to deliver the pragmatic benefits its design aspires to. The remainder of the paper is structured as follows. Section 2 introduces a motivating example that underpins the subsequent presentation and discussion. Section 3 briefly describes work that is related to one or more of the many facets of ISXL. Section 4 contains the main contributions of the paper. Finally, Section 5 concludes.

2 Motivating Example

As a motivating example, consider the problem of hypothesizing protein function from biological sequence analysis (hereafter, for the sake of brevity, referred to as *the **PF** problem)*. **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 subproblem of **PF**, viz., that of detecting distant homology (hereafter, for the sake of brevity, referred to as *the **DH** problem*). 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.

Although this protocol seems innocuous enough, closer analysis reveals that it poses many *specific challenges*. Firstly, it requires combining (and hence, interoperating across) very distinct data resources, e.g., protein sequence databases and protein family databases. Secondly, it requires combining (and hence, interoperating across) very distinct computational resources, e.g., homology filters and profile generators. Thirdly, note that the latter gives rise to a process model in the technical sense, i.e., a computation with a complex topology, composed of many sequential steps, involving branch selection, requiring iteration, etc. Fourthly, and more importantly, note that, even if only for usability by non-bioinformaticians, such process models are better expressed as *a coordination of components*, at a particularly coarse granularity, so that the components (i.e.,

the data and computational resources) are appropriately abstracted away from implementation detail.

Admittedly, given precise, complete and stable requirements, and a non-negligible amount of time (e.g., of the order of days, if not weeks), an expert software engineer could hope to capture the protocol *in silico* by writing an application using some general-purpose language (e.g., Java, or Perl, or Python), especially if she can count on domain-specific libraries (e.g., BioJava, or BioPerl, or BioPython, from the Open Bioinformatics Foundation (`http://open-bio.org/`)). However, this route is closed to biologists who are not expert software engineers: for them, this level of detail is inappropriate in the sense defined above.

One widely-used approach to counteract this impediment has been to front-end expertly-developed protocols with interfaces for non-bioinformaticians. However, by doing so one loses compositionality and closure, insofar as, short of the costly, tedious and error-prone approach of cutting and pasting results from interface to interface, such protocols do not (at the level of non-bioinformaticians) components make. In particular, since **DH** is but one subproblem of **PF**, a non-bioinformatician would naturally expect the automation of many such subproblems to be composable up to a complete protocol for **PF**. However, this is far from being the case without, again, precise, complete and stable requirements, and, again, a non-negligible amount of time from an expert software engineer to carefully craft the *in silico* solution of **PF** from the *in silico* solution of its subproblems, such as **DH**.

Now, suppose that such carefully crafted composition is, indeed, developed and front-ended for the benefit of non-bioinformaticians. Still, this approach to *in silico* biology fails to be isomorphic to *in vitro* biology on at least the following grounds: (a) automating the protocol automates 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; (b) 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).

In the **DH** problem, the need for addressing (a) above can be seen in the fact that 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 excellent 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 subproblem to the full **PF** problem. The need for addressing (b) above 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., lab books, personal file stores, etc.).

3 Related Work

Work that is related to the contributions of this paper can be categorized into the following kinds: data integration, fine-grained process integration, fine-grained process composition, fine-grained process coordination, and coarse-grained process coordination. Proposals that fall under the **data integration** category have aimed primarily at reducing impediments stemming from the semantic diversity of data stocks in biology. Their contribution has been to layer on top of semantically diverse data sources an integrated view of the data in the form of a query language (e.g., BioKleisli [1]), or a query interface front-ending a query language (e.g., TAMBIS [2]). Proposals that fall under the **fine-grained process integration** category have aimed primarily at reducing impediments stemming from the semantic diversity of analysis tools in biology. Their contribution has been to layer on top of semantically diverse analysis tools sources an integrated view of the data in the form of libraries (e.g., EMBOSS [5]) that are crafted for consistency. Proposals that fall under the **fine-grained process composition** category have aimed primarily at building on the contributions of the previous two categories. Their goal has been to facilitate the composition of data access and data analysis into simple, largely-linear topologies, i.e., steering clear of coordination proper, for which concurrency mechanisms are typically presupposed. Their contribution has been to layer on top of data and process integration suites a query interface with which a user can (with different degrees of effort) form a chain of evidence gathering steps (e.g., SRS [9]). Proposals that fall under the **fine-grained process coordination** category differ from the latter kind in that they support concurrency and hence allow more complex, nonlinear topologies to be specified that are best seen as computations based on component coordination. Their contribution has been to allow data and analysis tools to be seen as interacting components (e.g., IntelliGEN [4]). A more recent initiative is Biopipe [3]. The design of ISXL assumes that all the above proposals are meritorious in their goal. It therefore takes their achievements for granted and aims to build on top of such initiatives by viewing them as encapsulated components that an ISXL experiment can simply invoke. In contrast, proposals that fall under the **coarse-grained process coordination** category share with ISXL the view that the components they coordinate are more abstractly defined and bundle more aggregated, added-value functionality than simple querying and fine-grained analysis. All the proposals in this category are work in progress at the time of writing. Characteristic examples are BioMOBY [8] and myGrid [6], both of which adopt a service-oriented architecture, in which data sources and analysis tools are exposed as services that can be invoked through messaging protocols such as SOAP. While BioMOBY focusses on the registration and discovery aspects of service-based architectures, myGrid focusses on orchestration and on supporting end-users. ISXL also aims to express coarse-grained process coordination. It is a tool for building applications of the kind also supported by BioMOBY and myGrid, but unlike them it is language,not an application in itself. It can be compiled into many target orchestration languages at different grains. So, while BioMOBY offers facilities for service registration and discovery, ISXL presupposes them (e.g., as supplied by BioMOBY); while myGrid can be seen as an e-scientist workbench, ISXL is a tool that that workbench might include. ISXL adds capabilities to both BioMOBY and myGrid in the following respects.

It covers not just evidence gathering but also result interpretation, and supports long-lived, evolving experiments whose specification and results can be referred to within and across experiments. Thus, an ISXL-expressed *in silico* experiment comes closer to the ideal of isomorphism with the equivalent *in vitro*.

4 ISXL: From Concept to Implementation

ISXL*, Conceptually* The conceptual model that underpins the design of ISXL has the following facets. From the viewpoint of a biologist, ISXL provides constructs to model *in silico* experiments that comprise not just evidence gathering but also the explicit specification of the hypothesis that is supposed to explain the evidence produced by the protocol as well as the validation of that supposition by an explicitly specified method. Moreover, just like the evidence gathering core, the hypothesis and the validation process are specifiable as protocols too. Protocols constitute the core of an ISXL specification and coordinate methods that make use of materials. The core of the protocol specification comprises coordination rules that specify the conditions under which control flows from method to method and, separately, data flows, i.e., of the output of one method to the input of another. The UML-based diagram in Figure 1 depicts the conceptual relationships between ISXL constructs from an external and static viewpoint. It shows more precisely the sense in which an ISXL-expressed *in silico* experiment comes closer to the ideal of isomorphism with the equivalent *in vitro* one. In particular, it shows that, at a suitably-coarse grain, data and functions can be orchestrated with compliance to the appropriate methodological constraints.

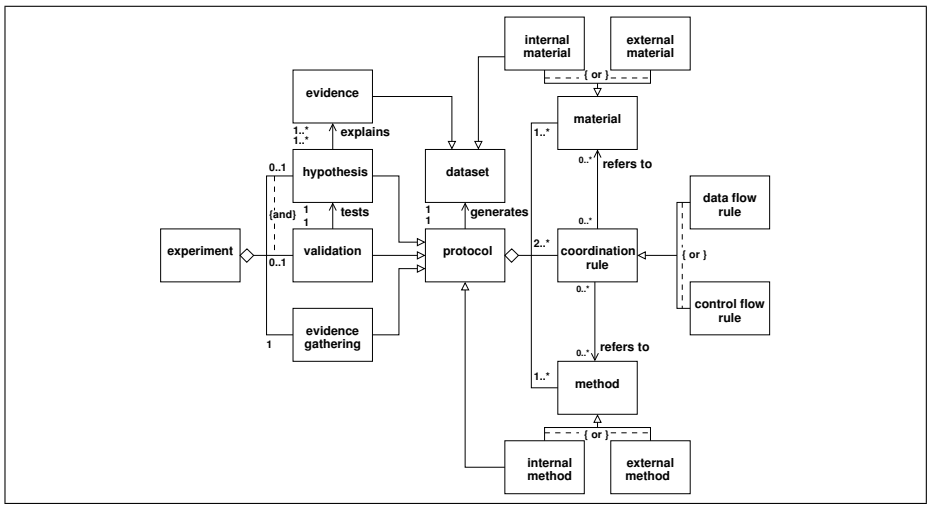

Figure 1. ISXL: Externally and Statically

Figure 1 is a static depiction of what an ISXL-specified *in silico* experiment is. An ISXL experiment also has a dynamic aspect, in that the language supports long-lived, evolving investigations as discussed in a companion paper [7]. Figure 1 depicts ISXL from the viewpoint of a biologist. Figure 2 extends (and completes) the ISXL conceptual model by showing how the isomorphism between the external and internal views is obtained. Thus, methods map to tasks and materials

map to tray contents in tasks, where a tray is an abstraction of a task port out of, and into, which data flows, and a task is an abstraction of a computational process (e.g., an interpretable piece of code, a web service, etc.).

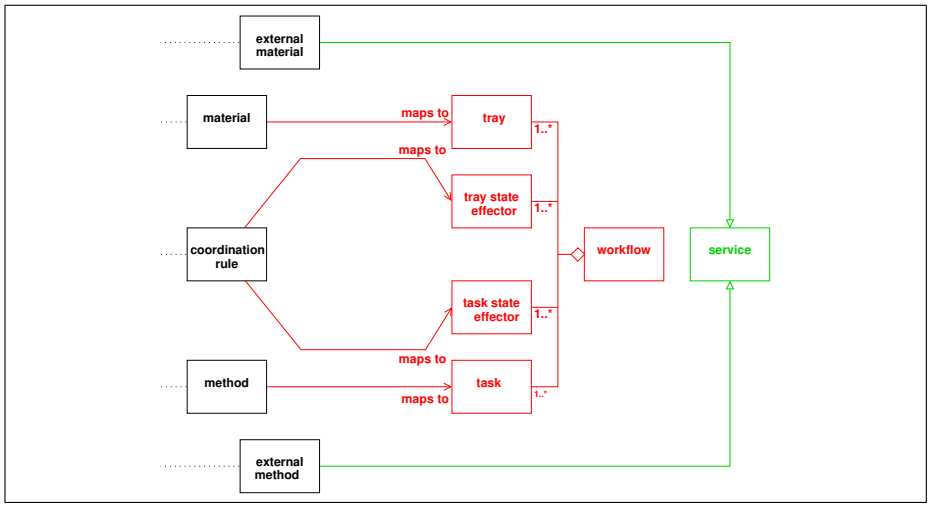

Figure 2. ISXL: Internally

ISXL, *Syntactically* The ISXL specification of an *in silico* experiment must contain an evidence gathering section, and may contain, as a pair, a hypothesis and a specification of how the latter is to be validated. 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. Materials map to trays (wherefrom, and to, the inputs and outputs of tasks flow) that are also stateful. The latter are referred to as in-trays and out-trays, respectively, A protocol is composed of a set of control flow rules and a set of data flow rules. A control flow rule has in the antecedent a condition, 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 differs in that its action affects trays only, e.g., by commanding the transport of data between trays. The admissible task and tray states (and hence the lexicon upon which the condition and action parts of ISXL control and data flow rules are founded) can be seen in Figure 5. The abstract operational semantics of a protocol is given in terms of the forward-chaining rule engine in Figure 6, as explained below. The hypothesis is an executable specification that wraps the evidence and conjectures that some property of the gathered evidence holds. In the example, the conjecture might be that no profile with G+C content above 60% will be longer than 15 letters. The validation is an executable specification that wraps the hypothesis and either assesses its validity, typically over a sample or against thresholds, or provides the information needed for doing so. For example, one might count the true and false predictions over a given sample to assess its accuracy on the materials used.

The ISXL source for a case of the **DH** protocol described in Section 2 is given in Figures 3 and 4. A control flow rule such as `M.3.1 = finished or M.3.2`

```
experiment DH_filtered_for_G+C_content

           inTrays: probe, sample
          outTrays: profile, results, valuation

  hypothesis # profiles will be no more than 15 letters long
    eval(Λ," len(profile) <= 15 ")
  end # hypothesis

  validation # count correct and incorrect predictions in a sample
    eval(Λ," X = [ $DH_filtered_for_G+C_content for probe in $sample ];
             Y = [ $hypothesis for profile in X ];
             correct = Y.count;
             incorrect = X.count - Y.count
             $valuation = (correct, incorrect) ")
  end # validation

  evidence gathering

    materials
      probe
    end # materials

    methods
      M1   (    # given a probe, perform a search
            task: FASTA @ http://www.ebi.ac.uk/fasta33/
         inTrays: probe
        outTrays: probe_results
      ),
      M2   (    # filter out if G+C content below 60%
            task: eval(Λ,"filterForGCContent(60)")
         inTrays: probe_results
        outTrays: results
      ),
      M3.1 (    # perform multiple sequence alignment
            task: Clustal-W @ http://www.ebi.ac.uk/clustalw/
         inTrays: probe, profile, results
        outTrays: alignment
      ),
      M3.2 (    # perform pairwise alignment
            task: WU_BLAST2 @ http://dove.embl-heidelberg.de/Blast2/
         inTrays: probe, results
        outTrays: alignment
      ),
      M4   (    # perform profile generation
            task: HMMER_2.2 @ http://hmmer.wustl.edu/
         inTrays: alignment
        outTrays: profile
      ),
      M5   (    # using the profile as probe, perform a search
            task: PHI-BLAST @ http://www.ncbi.nlm.nih.gov/blast/
         inTrays: profile
        outTrays: results
      )
    end # methods                                # continues in Part II
```

Figure 3. An ISXL Specification of DH (Part I)

```
    protocol                                    # continuation of Part I
      control flow
        M1 = START
          :  M1 := enabled;
        M1 = finished
          :  M2 := enabled;
        M2 = finished and eval(Λ,"cardinality(M2.results) > 1"
          :  M.3.1 := enabled;
        M2 = finished and eval(Λ,"cardinality(M2.results) = 1"
          :  M.3.2 := enabled;
        M.3.1 = finished or M.3.2 = finished
          :  M.4 := enabled;
        M4 = finished
          :  M5 := enabled;
        M5 = finished and eval(Λ,"cardinality(M5.results) > 0"
          :  M.3.1 := enabled;
        M5 = finished and eval(Λ,"cardinality(M5.results) = 0"
          :  STOP;
      end # control flow

      data flow
        probe = full
          :  probe -> M1.probe;
        M1.probe_results = full
          :  M1.probe_results -> M2.probe_results;
        M2.results = full and eval(Λ,"cardinality(M2.results) > 1"
          :  M2.results -> M3.1.results
          ,  probe -> M3.1.probe;
          ,  M3.1.profile := empty
        M2.results = full and eval(Λ,"cardinality(M2.results) = 1"
          :  M2.results -> M3.2.results
          ,  probe -> M3.2.probe;
        M3.1.results = full and eval(Λ,"cardinality(M3.1.results) > 0"
          :  results := empty
          ,  M3.1.results -> results;
        M3.1.profile = full and
        M3.1.results = full and eval(Λ,"cardinality(M3.1.results) > 0"
          :  profile := empty
          ,  M3.1.profile -> profile;
        M3.1.alignment = full
          :  M3.1.alignment -> M4.alignment;
        M3.2.alignment = full
          :  M3.2.alignment -> M4.alignment;
        M4.profile = full
          :  M4.profile -> M5.profile;
          ,  M4.profile -> M3.1.profile;
        M5.results = full and eval(Λ,"cardinality(M5.results) > 0"
          :  M5.results -> M3.1.results
      end # data flow

    end # protocol
  end # evidence gathering
end # experiment
```

Figure 4. An ISXL Specification of DH (Part II)

`= finished : M.4 := enabled` transfers control to task `M.4` (i.e., enables it to start executing) if either of the tasks `M.3.1` or `M.3.2` has finished running. A data flow rule such as `M3.1.alignment = full : M3.1.alignment -> M4.alignment` transfers the content of the `alignment` out-tray of task `M.3.1` to the `alignment` in-tray of task `M.4` if the former is full. ISXL control flow rules can express workflows involving iteration (e.g., of the `M.3.1` to `M5` section)), splits (e.g., at `M2` and `M5`), joins (e.g., at `M.3.1` and `M4`). Notice how control and data flow rules allow a method (e.g., `M.3.1`, which is being instantiated to ClustalW) to be used with different parameters (i.e., a probe sequence and some search results in the first pass versus a profile and some search results in the others).

For simplicity, it is being assumed that all the tasks denoted in the example are exposed as web services, e.g., Soaplab (`http://industry.ebi.ac.uk/soaplab/`) ones. Note that ISXL has none of the built-ins one would expect in a programming language. True to the goal of merely coordinating existing coarse-grained processes, in an ISXL computation, every atomic procedural abstraction is an invocation of an external process (which we view as services). Such invocations also encapsulate data operations on trays (i.e., services may be purely computational, or they may be data services).

ISXL does provide 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 references to services, are defined by `eval(`Λ`,` Σ`)` statements. It is part of the intended ISXL compilation strategy to pair the compiler up with both one target process enactment engine (or more, as explained below) and one general-purpose language platform (or more). With respect to the latter, of course, interpreted languages make this binding easier, but this is not mandatory: one might expose a compilation/execution server as a service for most modern non-interpreted languages. In the running example, 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 can be seen in Figure 3, both the specification of the hypothesis and of the validation process are done, in this example, as calls to `eval`. As Figure 1 shows, the hypothesis and validation are (fully-fledged) protocols. In Figure 3, we assume sufficient syntactic sugaring to allow a protocol consisting of a single `eval` task to collapse to that `eval` expression. We also assume that `eval` expressions that only occur in rule antecedents need not be declared as methods, insofar as they need not keep state. We further assume that a binding mechanism is available (invoked by the presence of a `$`-initiated identifier within a string in Λ) by means of which ISXL values can be made available in the space of the interpreter of Λ, and vice versa. Finally, in Figures 3 and 4, Λ is Python (or Jython), i.e., under the assumptions above, the double-quoted strings are well-formed Python fragments.

Note that even though all the computational resources used are well-known analysis tools and hence comparatively fine-grained process, ISXL methods are best seen as denoting fully-fledged services, aggregating much more functionality than suggested by this running example. Even without appealing to this capability, neither data integration, nor fine-grained process integration, nor fine-grained process composition approaches can emulate ISXL's functionality in this example, because, e.g., they lack the potential to enact either control or data flows (let alone both, independently) in parallel. A fine-grained process coordination approach could emulate ISXL functionalities for the **DH** problem, but composing from the level of **DH**, as one subproblem, to **PF** may be nontrivial because such

systems are often tightly-coupled to back-end data and computational resources, and scalability can become a problem in such settings due to the need to negotiate semantic heterogeneities. ISXL is capable of (decoupled) coarse-grained process coordination as are BioMOBY and myGrid, but these systems lack explicit means for hypothesis formulation and validation. They also lack a view of the life-cycle of an experiment, which ISXL supports by default (see [7]).

ISXL*, Semantically* ISXL has an abstract (i.e., enactment-engine-independent) semantics based on the evaluation of the control and data flow rules until a quiescent state is reached. (If it is reached at all, that is, because ISXL specifications are much too expressive for termination and confluence to be guaranteed. Indeed, the development of static analysis tools that can help ensure that an ISXL specification fails to terminate or to be confluent is a research goal worthwhile pursuing.) The consequents of control and data flow rules affect the state of tasks and trays according to the state-transition diagrams in Figure 5.

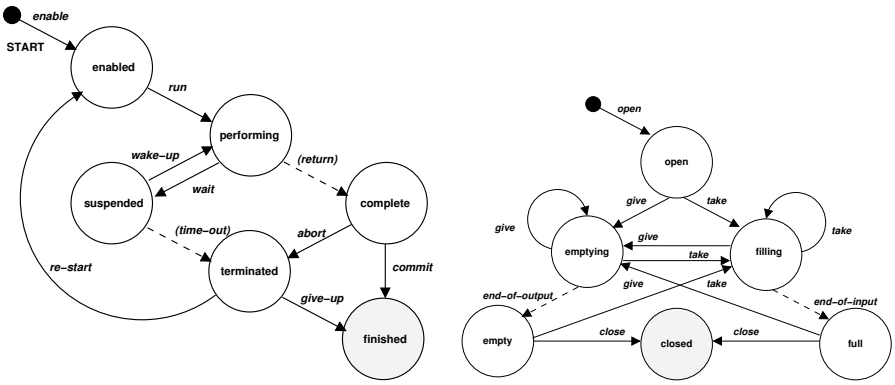

Figure 5. ISXL Task- (left) and Tray- (right) State-Transition Diagrams

The abstract semantics of an ISXL experiment is the abstract semantics of its protocol section. In turn, the abstract semantics of the protocol section is defined by an algorithm that evaluates a set of coordination rules. Many concrete algorithms could be used for this purpose, leading to different run-time properties of ISXL experiments. In our work, we have tended to ground our thinking on this matter on forward-chaining algorithms (of which RETE is one of the best-known examples, because it is relatively efficient and efficiently implementable).

Figure 6 presents the pseudocode for one such rule engine. The rule engine can be parameterized by an ordering (which is used for conflict resolution when more than one rule can be fired) and by the number of rules that can be fired in one iteration. In Figure 6, these two parameters default to textual occurrence and to a single rule, respectively. Some definitions and notational conventions now follow. Given a protocol (i.e., a list of distinct rules) $R = [r_1, \ldots, r_n]$ and a total order $\prec$ on R, let $slice(R, i, j)$ denote the sub-list $[r_i, \ldots, r_j]$ of R, let $R' = sort(R, \prec)$ denote the enumeration R' of R under $\prec$, and let ++ denote list append. If r is the rule $c : a$, then $condition(r)$ denotes c and $action(r)$ denotes a. If e is an expression, then $[\![e]\!]$ denotes its value. If r is the rule $c : a$, then note that $[\![condition(r)]\!]$ denotes the Boolean value resulting from evaluating c

```
rule_engine (R, ≺ = textual_occurrence, from = 1, to = 1)
  while TRUE :
    R′ = []
    for r in R :
      if [[condition(r)]] = TRUE :
        R′ ++ [r]
    if R′ = [] :
      return
    else :
      R″ = sort(R′, ≺)
      R‴ = slice(R″,from,to)
      for r in R‴ :
        if action(r) = STOP :
          return
        else :
          [[action(r)]]
```

Figure 6. A Rule Engine for ISXL Protocol Evaluation

against the set of task- and tray-state transition diagrams for the protocol to which r belongs. Likewise, $[\![action(r)]\!]$ denotes the outcome of executing a (and hence effecting the appropriate transitions) on the set of task- and tray-state transition diagrams for the protocol to which r belongs. The concrete, low-level semantics of each state and transition in either of the diagrams in Figure 5 is, of course, implementation-specific.

The concrete semantics of an ISXL specification is given by translation of the abstract semantics just sketched to a process modelling language over the latter's concrete enactment engine. This means that ISXL does not come equipped with its own enactment engine and is not tied to any. For example, myGrid workflows are written in Scufl, a language whose workflows are enacted by Freefluo (`http://freefluo.sourceforge.net/`). As is the case with most sufficiently expressive languages, the semantics of ISXL is more informatively formulated as being relative to an execution environment. We have specified in detail a compiler of ISXL into the PML process modelling language as enacted by a PWI (for Process Wise Integrator) engine (`http://processweb.cs.man.ac.uk/doc/pmlRefPdf/`).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 an ISXL compilation target.

In addition, we plan to compile an ISXL experiment into a BPEL4WS (`ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf`) orchestration that can be enacted over the BPWS4J (`http://www.alphaworks.ibm.com/tech/bpws4j`) engine. This is still ongoing work but the basic compilation strategy can be briefly sketched, as follows. Each ISXL experiment compiles into a BPEL4WS process whose main body implements the saturation algorithm in Figure 6 by which task states and tray states both progress through the transition diagrams in Figure 5. Each task compiles into a subprocess inside which a set of BPEL4WS containers are defined, viz., one to record the task state, as many to record tray states as there are in- and out-trays, and, again, as many

to record tray contents as there are trays. The container corresponding to the task state is managed so as to behave according to the ISXL task state transition semantics. Likewise, the containers corresponding to tray states. Finally, and most crucially, the subprocess acts as a wrapper for the required web service (e.g., a FASTA service), insofar as its main responsibility is to prepare the call (e.g., a SOAP RPC request) that invokes the required web service as a result of the task it models entering the *perform* state. Thus, a change in task state (as a result of some action in some rule consequent) is the consequence of a message having been received by the subprocess that represents the task. In other words, an action in a rule consequent compiles to a message-send to the subprocess that implements the task or tray referred in the action. Upon receipt, the subprocess updates the appropriate containers, and, in consequence, the task (or tray) state undergoes a transition, and, in turn, the required web service may be invoked, data may start to be sent, or be requested, etc. Data exchange, as standard in SOAP-based interactions, is document-based and is kept in containers that model tray contents and, from there, are subsequently passed around and shared by message exchange within the component activities of the root process that models the experiment as a whole.

5 Conclusions

This paper has introduced ISXL, a language for *in silico* experiments that uses a coarse-grained process coordination approach. An ISXL-specified experiment is (a) cast in a conceptual model that shadows the research method as applied to the empirical sciences; (b) may include explicit hypothesis formulation and validation rather than simply the core functionality of evidence gathering; and (c) is long-lived, in that there is built-in support for denoting experiment specifications that evolve, as well as the results of past runs, which are, in turn, annotated and kept in an underlying persistent store. In this respect, ISXL is a contribution towards the vision of *collaboratories* which are for e-science what *virtual organizations* are for e-business.

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. 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.
3. S. Hoon, K. K. Ratnapu, J. ming Chia, B. Kumarasamy, X. Juguang, M. Clamp, A. Stabenau, S. Potter, L. Clarke, and E. Stupka. Biopipe: A Flexible Framework for Protocol-Based Bioinformatics Analysis. *Genome Research*, 13:1904–1915, 2003.
4. K. Kochut, J. Arnold, A. P. Sheth, J. A. Miller, E. Kraemer, B. Arpinar, and J. Cardoso. IntelliGEN: A Distributed Workflow System for Discovering Protein-Protein Interactions. *Distributed and Parallel Databases*, 13(1):43–72, 2003.
5. P. Rice, I. Longden, and A. Bleasby. EMBOSS: The European Molecular Biology Open Software Suite. *Trends in Genetics*, 16(6):276–277, 2000.
6. R.Stevens, A. Robinson, and C. Goble. myGrid: Personalised Bioinformatics on the Information Grid. *Bioinformatics*, 19(Suppl. 1):302–304, 2003.
7. A. Tröger and A. A. A. Fernandes. A Language for Comprehensively Supporting the *In Vitro* Experimental Process *In Silico*. In *Proc. IEEE Fourth Symposium on Bioinformatics and Bioengineering (BIBE2004)*, pages 47–56, 2004.
8. M. D. Wilkinson and M. Links. BioMOBY: An Open-Source Biological Web Services Proposal. *Briefings In Bioinformatics*, 3(4):331–341, 2002.
9. E. M. Zdobnov, R. Lopez, R. Apweiler, and T. Etzold. The EBI SRS Server: New Features. *Bioinformatics*, 18(8):1140–1150, 2002.