Scientific Workflows: More e-Science Mileage from Cyberinfrastructure*

Bertram Ludäscher[†] [‡] ludaesch@ucdavis.edu
Shawn Bowers[‡] sbowers@ucdavis.edu
Timothy McPhillips[‡] tmcphillips@ucdavis.edu
Norbert Podhorszki[†] npodhorszki@ucdavis.edu

Abstract

*We view scientific workflows as the domain scientist's way to harness cyberinfrastructure for e-Science. Domain scientists are often interested in "end-to-end" frameworks which include data acquisition, transformation, analysis, visualization, and other steps. While there is no lack of technologies and standards to choose from, a simple, unified framework combining data modeling and process-oriented modeling and design of scientific workflows has yet to emerge. Towards this end, we introduce a number of concepts such as models of computation and provenance, actor-oriented modeling, adapters, hybrid types, and higher-order components, and then outline a particular composition of some of these concepts, yielding a promising new synthesis for describing scientific workflows, i.e., Collection-Oriented Modeling and Design (*CoMaD*).*

1 Cyberinfrastructure for e-Science

Traditional methods for conducting science, *i.e.*, by observing natural processes *in vivo* and by controlled wetlab experiments *in vitro*, are increasingly complemented (and at times replaced) by *in silico* experiments and simulations. Computational scientists, for example, have been exploiting high-end computing resources for simulation science[1] for some time now. In recent years, many other scientific disciplines have embraced and even driven IT development and computer science research to improve current practice or invent new ways of doing science through "computational thinking" [1]. In the UK, *"e-Science is about global collaboration in key areas of science and the next generation of infrastructure that will enable it."*[2] *"[Another] feature of such collaborative scientific enterprises is that they will require access to very large data collections, very large scale computing resources and high performance visualisation back to the individual user scientists."*[3]

*Work supported by NSF/SEEK (DBI-0533368), DOE SciDAC/SDM (DE-FC02-01ER25486) and CPES.

[†]Dept. of Computer Science, University of California, Davis
[‡]Genome Center, University of California, Davis

[1] `http://www.csm.ornl.gov/newSS.html`

[2] John Taylor, Director General of Research Councils, UK

[3] `http://www.rcuk.ac.uk/escience/`

In the US, this "next generation of infrastructure" is also called *cyberinfrastructure* [3], emphasizing the need for advanced software tools and high-end computing platforms that allow scientists to effectively and efficiently conduct scientific data management and analysis, often collaboratively. While the use of cyberinfrastructure for e-Science obviously cannot replace a scientist's insight, creativity, or ingenuity to come up with new (possibly interdisciplinary) theories and discoveries, there is hope that cybertools can accelerate scientific knowledge discovery and even facilitate new experiments and science not possible before.

Cyberinfrastructure can be seen as a layered architecture: Instruments ranging from microscopes to telescopes, from microarrays, mass spectrometers, and fMRI scanners to *in situ* and remote sensing devices for environmental studies, *etc.*, all provide the raw data for subsequent analyses. Similarly, computational scientists run large-scale simulation codes on supercomputers, generating vast amounts of data, *e.g.*, from atmospheric and/or ocean models, models of supernova explosions, protein folding models, or particle simulations in fusion plasmas. Assimilation models inform simulations via observational data, *i.e.*, they integrate, *e.g.*, Doppler radar measurements into a cloud model.

On top of data producers, a Grid middleware layer can be employed to facilitate transparent access to distributed data and computational resources. Grid or web services then provide distributed data access, authentication, resource allocation, scheduling services, remote execution, *etc.*

Finally, scientists may use this cyberinfrastructure in a number of ways. For example, some users may rely on web-based tools or custom portals through which they can execute predefined services or data analysis pipelines. Similarly, a web-accessible "workbench" environment may allow users to create simple ad-hoc analysis workflows. When trying to implement more sophisticated use cases, specifically advanced analysis pipelines from the available resources, however, scientists are often left to their own devices. This can mean, *e.g.*, that a scientist has to use copy-paste to move data from one tool or webpage to another. Alternatively, some scientists simply stay within a single tool (say a spreadsheet application) or otherwise confine themselves to predefined use cases provided through a web portal. For more sophisticated analyses, scientists may need to develop their own specialized programs or scripts to "glue" together various applications. In order to fill the obvious

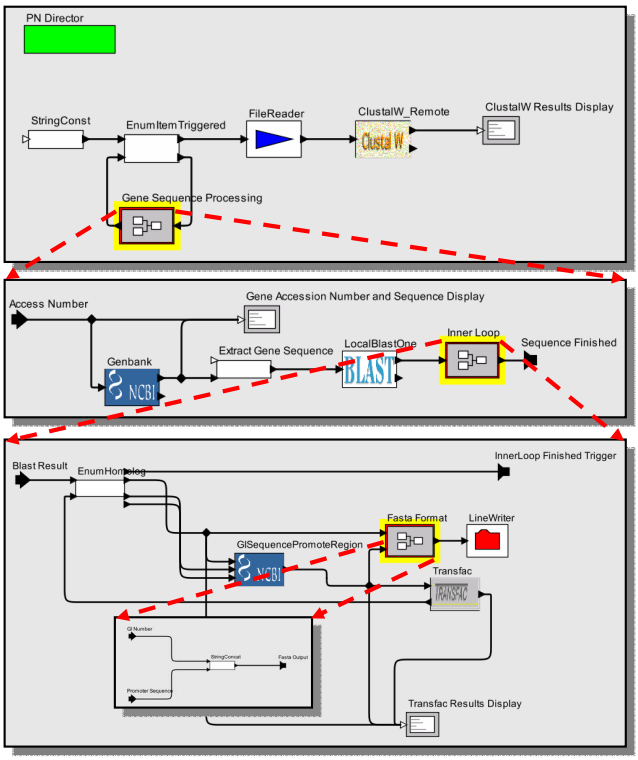

Figure 1. Promoter Identification Workflow (PIW)

need at the top-level of cyberinfrastructure, *i.e.*, where scientists build and run their virtual experiments and analysis pipelines, *scientific workflows* are emerging as a new modeling and execution paradigm [12, 22, 29].

2 Scientific Workflows

The term 'workflow' has traditionally been used mainly in the context of business applications. The Workflow Management Coalition,[4] *e.g.*, defines workflow as "*the computerized facilitation or automation of a business process, in whole or part.*" In this sense, a scientific workflow facilitates or automates a "science process", *e.g.*, the execution of an experiment in the wetlab, followed by analytical post-processing of the data, and an interpretation of the results. Specifically, we can view a scientific workflow as a description of the processes that a scientist needs to execute in order to create a "science product" (*e.g.*, data analysis results, visualizations, solved protein structures, publications, *etc.*)

Promoter Identification Workflow. As a concrete example, consider the scientific workflow depicted in Figure 1. This so-called *Promoter Identification Workflow* (PIW) [27] has been implemented in the KEPLER system [21]. The workflow chains together genomic techniques (here: from microarray experiments) with bioinformatics tools (here, *e.g.*, BLAST, ClustalW, and Transfac). Starting from microarray data, cluster analysis algorithms are used to identify genes that share similar patterns of gene expression profiles that are then predicted to be co-regulated as part of an interactive biochemical pathway. Given the gene-ids, gene sequences are retrieved from a remote database (*e.g.*, GenBank) and fed to a sequence alignment tool (*e.g.*, BLAST) that finds similar sequences. In subsequent steps, transcription factor binding sites and promoters are identified to create a promoter model that can be iteratively refined.[5]

Terminology and Basic Concepts. The workflow description in Figure 1 illustrates a number of basic concepts: We consider a workflow as a network of independent components called *actors*. Actors can be *atomic* (representing, *e.g.*, calls to local tools or remote services) or *composite*: the figure shows three composite actors together with the subworkflows they contain. KEPLER inherits many features from the underlying PTOLEMY II system [2], *e.g.*, the capability of nesting workflows via composite actors, and the ability to separate the concern of workflow *description* from that of workflow *orchestration*. The former is given by the workflow graph (and its nested subgraphs), while the latter is given by a special orchestration and scheduling component called a *director*. The green box, labeled PN-Director in Figure 1 is such a component. It dictates that the workflow description be executed as a *process network* (PN).

PN is a basic model of computation (MoC), based on the abstract Kahn process network model [16, 19]. In this MoC, actors execute as concurrent asynchronous processes which communicate by sending streams of data tokens over FIFO *channels* (the directed edges in the workflow graph).

Several models of computation can be seen as special cases of PN: *e.g.*, an SDF-Director[6] assumes that each actor declares at compile-time its "firing rate", *i.e.*, the number of tokens consumed (for each input port), and the number of tokens produced (for each output port) per invocation of the actor. While the PN MoC does not have such restrictions, these SDF restrictions allow certain static analyses of workflows and yield guarantees that are not available for PN. For example, in SDF it is decidable at compile-time whether a workflow can execute within bounded memory (all message queues between actors need to hold only a fixed number of tokens), or whether a deadlock can occur. In Grid workflows further restrictions are often applied, *e.g.*, the DAG (directed acyclic graph) MoC can be seen as a restriction of SDF, excluding cycles and stream processing. Thus, a DAG schedule is simply a topological sort of a workflow graph.[7]

[4] `http://www.wfmc.org`

[5] Promoters are subsequences that enable gene transcription.

[6] for Synchronous Data-Flow

[7] Many other MoCs exist in PTOLEMY II and thus in KEPLER: *e.g.*,

3 Actor-Oriented Modeling and Design

We call workflows which are designed with a dataflow MoC (*e.g.*, PN or SDF) in mind a *dataflow process network*. From a user's point of view, actors are the main modeling construct in such workflows (with the director handling important details, *e.g.*, workflow scheduling and orchestration). Thus, we also speak of *actor-oriented modeling and design* (AMAD) [18, 5]. The AMAD approach includes hierarchical modeling (via composite actors) as mentioned above, the use of actor *parameters* to change their behavior, and *semantic types* (expressions from an ontology language) [5, 6] to further describe data beyond the conventional structural data types. For the above PIW workflow, a semantic data type could involve concepts such as DNA_sequence or Promoter, while a structural type might be $[\,(\texttt{int}, \texttt{string})\,]$, *i.e.*, a list of pairs from $\texttt{int} \times \texttt{string}$.

The actors at the core of AMAD can be thought of as parameterized functions, *i.e.*, an actor A with parameters $\bar{\mathsf{p}} = p_1, \ldots, p_k$, and n inputs and m outputs gives rise to a parameterized function $\mathsf{f}^{\mathsf{A}}_{\bar{\mathsf{p}}}$:

$$\mathsf{f}^{\mathsf{A}}_{\bar{\mathsf{p}}} :: (\alpha_1, \ldots, \alpha_n) \rightarrow (\beta_1, \ldots, \beta_m)$$

where α_i and β_j are type expressions for the structural types of the corresponding input and output ports of A. If $n = 0$ then A is called a *source*, else if $m = 0$ then A is a *sink*. A stateful actor A can be modeled via a function f^{A} having additional (usually hidden) input and output arguments to pass the original state in and the updated state out.[8]

Let τ be a structural type (schema) describing an actor's input (or output) ports. A *hybrid data type* is a pair $(q_\tau, \sigma_\mathcal{O})$ consisting of (i) a query $q_\tau(x_1, \ldots, x_k)$, selecting k-tuples $(x_1, \ldots, x_k)$ of data items, and (ii) a semantic annotation of the form $\sigma_\mathcal{O} = (x_1{:}\mathsf{C}_1, \ldots, x_k{:}\mathsf{C}_k)$, "tagging" instances x_i with concepts C_i from an ontology $\mathcal{O}$.

Both types (*i.e.*, τ and $\sigma_\mathcal{O}$) can be treated as orthogonal, allowing the workflow system to warn the user about structural as well as semantic type violations independently. The approach can be extended to allow logic constraints between both type systems. For example, the logic constraint

$$\forall x \forall y : \underbrace{\mathtt{R}(x, y)}_{\tau\text{-expression}} \longrightarrow \underbrace{\mathsf{Promoter}(x) \wedge \mathsf{DNA\_seq}(y)}_{\mathcal{O}\text{-expression}}$$

can be seen as an annotation of the record type $\mathtt{R}(x, y)$ with the corresponding concepts from an ontology $\mathcal{O}$. For more complex annotation constraints and their use see [6].

to model a physical process via ordinary differential equations and solve them numerically, a CT (continuous time) director can be used.

[8] In a dataflow process network, the state output port of A would be wired back to the state input port of A via an intermediate delay element to avoid a deadlock in the feedback loop.

Shimology: Got Adapters? The AMAD model can be refined further by distinguishing *user actors* (those that are critical for the scientist) from *adapters* ("shim" actors [15]). In a bioinformatics workflow, *e.g.*, a sequence alignment step or a clustering step would be considered crucial from the user's perspective (*e.g.*, they may be mentioned in the methods section of a scientific publication), hence we classify the corresponding actors as user actors. On the other hand, actors which deal with file format conversions, data movement, the control of loop-constructs, unit conversions (*e.g.*, from Fahrenheit to Celsius), *etc.* can be considered (generalized) adapters. Adapters and shims can be further classified according to the type of mismatch they overcome. Some adapter categories are: *converters* translate information into alternative forms (*e.g.*, a *file converter* to/from FASTA format; a *unit converter* to/from SI units); *selectors* access parts of a data structure for subsequent processing; *transformers* reorganize or transform a data structure (*e.g.*, via an XQuery), *i.e.*, they implement schema mappings; and finally, *iterators* "switch gears", *e.g.*, by introducing a loop around an actor $\mathsf{f}^{\mathsf{A}} :: \alpha \rightarrow \beta$ so that the new compound can accept inputs of type $[\alpha]$ instead of just α (*cf.* Section 4).

The Different Faces of Scientific Workflows

As mentioned before, domain scientists are typically less interested in adapters and more in the right choice of user actors (*e.g.*, which analysis methods should be used) and the overall workflow design. A workflow engineer, on the other hand, might be charged with lower-level design issues in a scientific workflow, or with deploying a workflow in a particular execution environment. Finally, a computer scientist's role may be to ensure that workflows can be statically analyzed or efficiently scheduled. Below, we elaborate on these different perspectives on scientific workflows.

Domain Scientist's View. From an end user's point of view, *automation* is a major benefit of the scientific workflow approach. For example, a scientist may want to execute the same data analysis pipeline over and over again with many different input datasets and/or using different parameter settings (*e.g.*, via parameter sweeps [9]). The system should also be able to record (and later replay) user interactions made during the workflow (interactive actors may require user input at runtime, *e.g.*, asking the scientist to select or dismiss data items for further analysis, to choose among different analysis methods, *etc.*)

Workflow runs should generate analysis reports, including sufficient *provenance* information to facilitate result interpretation and "debugging" [7]. For example, the user may be interested in the specific input datasets, intermediate results, parameter settings *etc.* that contributed to a particular (possibly surprising) result. Workflow descriptions

together with *execution traces* can thus serve as evidence or explanations for analysis results. In the future, certain types of scientific publications may include workflow descriptions and provenance metadata as corroborating evidence for the published results (similar to the current practice of some bioinformatics journals to require deposition of data products in public repositories such as PDB).

Other common requirements from a scientist's perspective include: Workflow designs should be intuitive and lend themselves to *reuse* and *repurposing*. Moreover, workflow *reconfiguration* should be seamless. For long-running workflows (*e.g.*, launching simulations on remote supercomputers [17]), it is often necessary to *monitor* workflow execution and if necessary suspend or abort remote jobs and/or dynamically change parameter settings of jobs.

Scientific workflows are artifacts (stored *e.g.* in XML) that can be *shared* and discussed by the community, and further evolved as needed. For this to be effective, paradigms and languages should encourage *comprehensible designs* of workflows, based on suitable *modeling constructs* and abstractions. This may be achieved, *e.g.*, via actor-oriented modeling and design and its extensions, *i.e.*, by bringing together flow-oriented and hierarchical modeling (nested workflows), flexible type systems, high-level *design patterns* (*e.g.*, based on higher-order functions), *etc.*

Workflow Engineer's View. Domain scientists are the end users (and increasingly the co-designers) of scientific workflows. Workflow engineers play a role similar to software engineers. In the absence of high-level design support for end users, a workflow engineer may need to design and implement scientific workflows. In particular, subsets of the many existing tools in a given discipline often have to be made available as actors, *i.e.*, "wrapped" into components that can be invoked by the scientific workflow system. Many tools, *e.g.*, can be invoked as a command from an interactive shell; a corresponding *shell-actor* can then be used to quickly wrap existing tools into executable actors.

Another currently very popular approach is to export functionality from existing tools by deploying corresponding *web services*. The use of web services can be useful for loosely coupled workflows, but in itself does little to address the underlying challenges of workflow design, orchestration, and service composition: How are multiple components supposed to work together when they were not designed to fit together? What should web service signatures look like to facilitate reuse? Modeling and design frameworks (*e.g.*, AMAD and COMAD) begin to answer such questions, based on general techniques and principles know from programming languages and software engineering, *i.e.*, types with parametric polymorphism, polymorphic methods, abstract data types, models of behavioral polymorphism [18], and other reusability paradigms *e.g.*, based on monads and arrows [14]. While computer scientists might develop or extend these sophisticated methods, most underlying details could be hidden from the workflow engineer, who instead could focus on capturing the scientist's ideas in a high-level workflow modeling language.

Another concern for workflow engineers is performance: Given a workflow description, how can the workflow be executed efficiently in a cluster or Grid environment, taking advantage of task-, pipeline-, and data-parallelism? This optimization challenge can include a variety of problems, *e.g.*, job-shop scheduling, load balancing, workflow rewriting, and result caching (*e.g.*, "smart re-runs" of workflows can avoid unnecessary computations by taking into account dataflow dependencies and data and parameter updates).

Computer Scientist's View. The borderline between the roles and views of workflow engineers and computer scientists is not always clear cut. For example, scientific workflow modeling and design methods may be based on a blend of techniques from software engineering, database theory, query optimization, stream processing, functional programming, and type theory. In particular, when devising a new workflow modeling paradigm, a computer scientist might be interested in formalisms that allow *static analysis* of workflows to guarantee certain properties such as (structural and/or semantic) type safety, freedom from deadlocks, boundedness of memory, *etc.* A formal model of computation (MoC) can help computer scientists to derive strategies for scheduling and optimization of workflows. Query rewriting, functional program transformations, and stream optimization techniques may be applicable for some MoCs but not for others. Some workflow scheduling problems are provably hard (*e.g.*, NP-complete) and will require heuristics to guarantee efficient plan generation. Computer scientists may also be charged to devise inference algorithms to exploit structural and semantic type information during workflow design, *e.g.*, to search for suitable components and data sets, or to generate schema mappings for structural adapters [4].

Another computer science challenge is the design and efficient implementation of *provenance management systems* for scientific workflows [25]. Consider a workflow description W together with some input data $\mathbf{x}$ and parameter settings $\bar{\mathsf{p}}$. A model of computation M prescribes how to compute the workflow output $\mathbf{y} = M(W_{\bar{\mathsf{p}}}(\mathbf{x}))$. With every workflow execution we associate a data structure $\mathbf{r}$, called a *run*. We can think of a run $\mathbf{r}$ as a complete capture of the execution of W on $\mathbf{x}$, *in terms of* the MoC M. Depending on the user requirements, a *model of provenance* (MoP) M' will in general ignore some aspects $\mathbf{i}$ of M that are not relevant for the user, while at the same time capturing some information that might not be part of M (*e.g.*, the workflow owner, execution timestamps, *etc.*) With slight abuse of no-

tation, we may state that $\mathbf{t} = \mathbf{r} - \mathbf{i} + \mathbf{m}$, *i.e.*, a *provenance trace* $\mathbf{t}$ is a trimmed workflow run $\mathbf{r}$ that ignores some aspects $\mathbf{i}$ of the given MoC model, but models some additional (user-defined) properties $\mathbf{m}$ [7].

Bioinformaticians and Other New Species. The above discussion of the different perspectives on and faces of scientific workflows was based on a rough classification of roles, *i.e.*, domain scientist, workflow engineer, and computer scientist. In concrete projects, the borderlines may be more or less blurred than described above and additional roles may exist. The establishment of new fields from interdisciplinary grass roots as well as the recent emergence of e-Science will probably also breed new species of scientists who are at home in multiple fields [11] and thus are poised for scientific discoveries in previously unexplored areas. A prominent example are bioinformaticians which can act in all of the above-mentioned roles, and thus can be most effective in advancing the state-of-the-art at the intersection of disciplines, *e.g.*, biology and computer science.

4 Function-Oriented Modeling and Design

In order to illustrate how scientific workflows give rise to interesting technical challenges and opportunities for computer scientists, consider again the promoter identification workflow (PIW) in Figure 1. This early design exhibits a number of workflow modeling problems [20]. First and foremost, this process network design is overly complex: *e.g.*, there are several complex loop constructions involving backward arcs. Moreover, there are several "special purpose" actors such as EnumHomolog that are designed to fit precisely in the spot which they occupy, resulting in a very brittle and non-reuseable workflow. Another problem is that the design is unnecessarily "serial", *i.e.*, it does not expose much of the task- and pipeline parallelism inherent in PIW.

Figure 4 shows a schematic depiction of several different workflow modeling paradigms. In the basic process network model of Figure 4(a), all actors are considered the same, *i.e.*, the model does not distinguish between user actors and adapters. In particular, the "auxiliary" actors in Figure 4, dealing with control-flow issues such as loop control, distract from the main user actors and add to the overall workflow complexity. Note that there are several dataflow directors (DF) that can implement different dataflow MoCs such as PN and SDF.

One idea to improve the basic process network model for scientific workflow design is to include ideas from functional programming, specifically higher-order constructs such as map and fold [20]. Consider the workflow graph in Figure 2: This graph represents a dataflow network with modeling constructs from functional programming (in Figure 4(b), this is indicated by a FP-DF director). The workflow graph in Figure 2 corresponds to the following functional program:

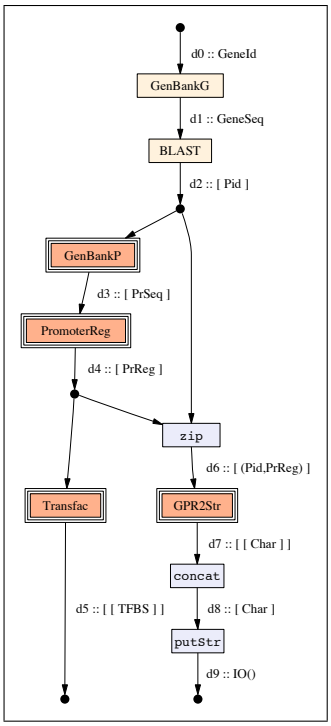

Figure 2. Functional process network for PIW; double outlined boxed are wrapped by a map functor.

```
d0 = $gid
d1 = GenBankG d0
d2 = BLAST d1
d3 = map GenBankP d2
d4 = map PromoterRegion d3
d5 = map Transfac d4
d6 = zip d2 d4
d7 = map GPR2str d6
d8 = concat d7
d9 = putStr d8
```

Note that both the graph as well as the functional program have a much simpler structure than the workflow design in Figure 1. In this modeling paradigm, actors have an associated function signature which can be used to type-check a workflow, or propagate the known types through the workflow to infer the unknown types. For example, the type of the first actor might be:

$$\mathsf{GenBankG} \;::\; \mathsf{GeneId} \rightarrow \mathsf{GeneSeq}$$

where GenBankG takes as input some GeneId and returns a sequence GeneSeq. Similarly, we might have the following additional actor signatures:

$$\begin{array}{lcl} \mathsf{BLAST} & :: & \mathsf{GeneSeq} \rightarrow [\mathsf{Pid}] \\ \mathsf{GenBankP} & :: & \mathsf{Pid} \rightarrow \mathsf{PrSeq} \end{array}$$

where BLAST returns a list of promoter-ids for a given gene sequence, and where another GenBankP function takes a single promoter-id and produces the promoter sequence. Intuitively, a scientist might want to chain together the output of BLAST with the input of GenBankP. However, since the types are different ($[\mathsf{Pid}] \neq \mathsf{Pid}$), GenBankP cannot accept the output from BLAST. The corresponding type system will indeed reject this connection, and a type inference algorithm may even suggest a special "adapter" (in our terminology) to bridge the structural gap.

In this case, the well-known higher-order function map is such an adapter. It takes as input a function $f :: (\alpha \to \beta)$ and a list of type $[\alpha]$, and successively applies f to each list element, resulting in a list of type $[\beta]$:

$$\begin{array}{lrcl} \mathsf{map} :: & (\alpha \to \beta) \to [\alpha] & \to & [\beta] \\ & f\ [x_1, \ldots, x_n] & = & [f(x_1), \ldots, f(x_n)] \end{array}$$

The Taverna scientific workflow system has a similar construct called *implicit iteration* [26] to introduce an implicit map. In our case, we obtain

$$\mathsf{map}(\mathsf{GenBankP}) \ :: \ [\mathsf{Pid}] \to [\mathsf{PrSeq}]$$

which now is of the desired type and which can receive the BLAST output. In Figure 2, actors that have this kind of map around them are depicted with a double outline and darker. It should be clear from both the above functional program as well as the figure that workflow designs resulting from a functional programming model can exploit higher-order constructs, thus leading to cleaner designs. In Figure 4(b) we show a dataflow network enhanced with this functional approach. Another advantage when adopting this model for scientific workflow design is the ability to exploit rewriting rules known from functional programming.

For example, we can make use of the fact that

$$\mathsf{map}(f) \circ \mathsf{map}(g) = \mathsf{map}(f \circ g)$$

to replace the two mapped actors GenBankP followed by PromoterReg, by a single mapped composite actor containing their composition $\mathsf{PromoterReg} \circ \mathsf{GenBankP}$. Similarly, we can replace the zip followed by $\mathsf{map}(\mathsf{GPR2Str})$ in Figure 2 by a single $\mathsf{zipWith}(\mathsf{GPR2Str})$, where the higher-order function zipWith has the signature

$$\mathsf{zipWith} :: (\alpha \to \beta \to \gamma) \to [\alpha] \to [\beta] \to [\gamma]$$

and takes a function $f :: \alpha \to \beta \to \gamma$,[9] and applies it iteratively on pairs of the type (α, β) coming from the input lists $[\alpha]$ and $[\beta]$, yielding a result list of type $[\gamma]$. Figure 3 shows the resulting workflow. Note that this rewriting has integrated two independent actors into a single one. Such rewritings can also be exploited by the workflow engineer to optimize workflow execution. For example, if the two separate actors were originally sending large amounts of data over the wire, after the rewriting they are amalgamated into a single actor which is likely more efficient, given that data is available locally instead of over the wire.

[9] in our case $\mathsf{GPR2Str} :: \mathsf{Pid} \to \mathsf{PrReg} \to [\mathsf{Char}]$

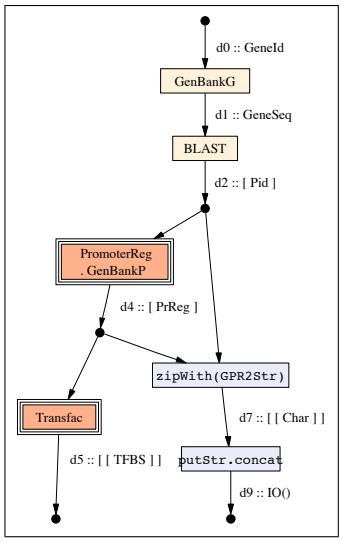

Figure 3. Simplified functional process network for PIW

5 Collection-Oriented Modeling & Design

The functional approach already marks a significant step forward in structured scientific workflow design. One reason is that higher-order functions such as map and fold can be used to eliminate complex backward loop structures from dataflow networks; *cf.* Figure 4(a-b). The type systems in functional programming are also quite flexible (*e.g.*, see *parametric polymorphism*), yet allow powerful type inference which can be exploited during workflow design.

Consider the following language of type expressions t:

$$\begin{array}{rcll} t & ::= & \mathbb{I} \mid \mathbb{R} \mid \mathbb{S} \mid \ldots & \textit{base}\ (\mathrm{int}, \mathrm{float}, \mathrm{string}, \ldots) \\ & \mid & (t_1, \ldots, t_k) & \textit{record} \\ & \mid & (t \to t) & \textit{function} \\ & \mid & [t] & \textit{list} \\ & \mid & \{\!|t|\!\} & \textit{bag} \\ & \mid & \{t\} & \textit{set} \end{array}$$

Here $\mathbb{I}, \mathbb{R}, \mathbb{S}, \ldots$ are base types *e.g.* for real numbers, strings, *etc.*; $(t_1, \ldots, t_k)$ denotes a k-tuple or record type; $(t \to t)$ is used for typing functions (including higher-order ones); and $[t]$, $\{\!|t|\!\}$, $\{t\}$ are type constructors for collection types, *i.e.*, lists, bags, and sets. The workflow design and programming style afforded by such (often nested) collections has proven useful for processing scientific data [10]. In addition, when dealing with MoCs that compute over unbounded streams, we can add *streams types* $\langle t \rangle$ and describe

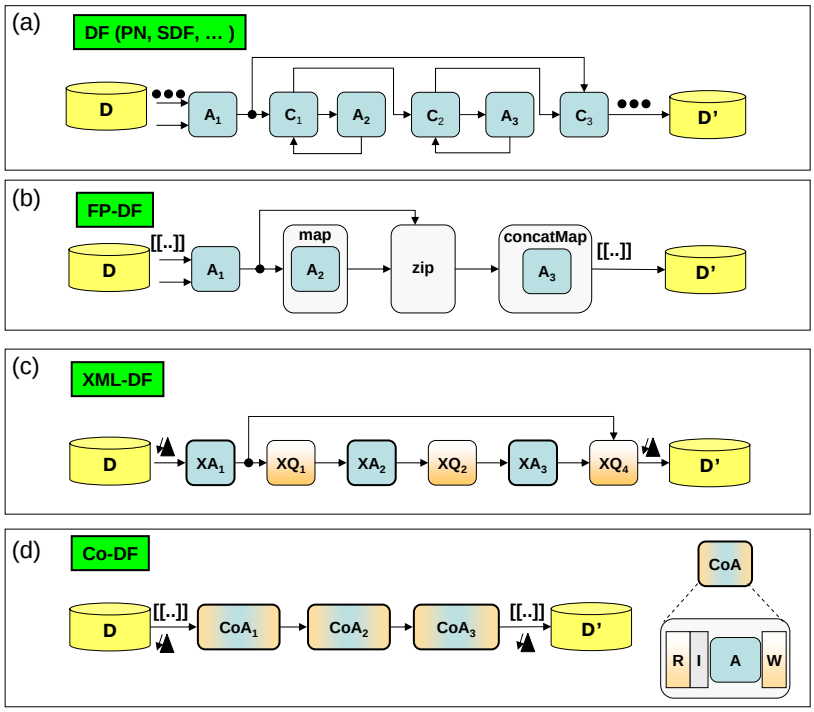

Figure 4. Different scientific workflow modeling paradigms: (a) vanilla process network, (b) functional programming process network, (c) XML processing network, (d) Collection-oriented Modeling and Design (CoMAD) network

Kahn processes as functions over these types [28]. However, despite its advantages, the functional approach still requires very precise and fine-grained data modeling.

On the other hand, in *XML processing networks* – see Figure 4(c) – we consider actors as consuming XML inputs and producing XML outputs, *i.e.*, an even more flexible *semistructured* data model is employed. As before, it is convenient to distinguish between user actors $\mathsf{XA}_1, \mathsf{XA}_2, \ldots$ which consume and produce XML, and *XML adapters*, *i.e.*, XML query and transformation actors $\mathsf{XQ}_1, \mathsf{XQ}_2, \ldots$

The *Collection-oriented Modeling and Design* approach (CoMAD) [24, 23] goes one step further and combines the modeling advantages of the functional approach, the XML transformation approach, and a general flow-oriented modeling style. In CoMAD, workflows are composed of *collection-oriented actors* (co-actors), *i.e.*, which contain the actual user actor A (Figure 4(d) right) plus built-in "gears" to facilitate the seamless chaining together of co-actors. More precisely, a co-actor has a *read-scope* (R in the figure), an *iteration scope* I, and a *write scope* W. R can be viewed as a query $q_{\mathsf{R}} :: \langle .. \rangle \rightarrow \langle (t_1, \ldots, t_k) \rangle$ mapping a *semistructured* XML stream into a *structured* stream of k-tuples. If the iteration scope I is empty, then the user actor $\mathsf{A} :: (t_1, \ldots, t_k) \rightarrow (t'_1, \ldots, t'_m)$ is fired once for each matched tuple (else I can be used to further group the tuple stream). Finally, the write scope W determines where the results of A firings are put back into the stream. By default, co-actors are add-only, *i.e.*, they inject computed results into the XML stream, immediately following the objects that had produced those results.

6 Conclusions

We have argued that scientific workflows are a core component of cyberinfrastructure for e-Science. Much work in computer science and engineering is focused on optimizing algorithms, system performance, and memory requirements. However, the most precious resource in e-Science is often neglected: human time. In this paper, we have argued that methods for better scientific workflow design are needed. To this end, we introduced actor-oriented modeling and design, extensions with semantic and hybrid types, functional and XML processing extensions, and finally collection-oriented modeling and design (CoMAD). Most of these underlying formalisms are not visible from the domain scientist's view, however can be important for the workflow engineer, or computer scientist. First experiences with phylogenetic workflows have shown that the CoMAD approach can lead to simpler, more reusable workflow designs [24, 23].

References

[1] Computational Thinking Seminar. `http://www.inf.ed.ac.uk/research/programmes/comp-think/`, 2006. University of Edinburgh.

[2] PTOLEMY II project and system. Department of EECS, UC Berkeley, 2006. `http://ptolemy.eecs.berkeley.edu/ptolemyII/`.

[3] D. E. Atkins, *et al.* Revolutionizing Science and Engineering Through Cyberinfrastructure: Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure, January 2003.

[4] S. Bowers and B. Ludäscher. An Ontology Driven Framework for Data Transformation in Scientific Workflows. In *International Workshop on Data Integration in the Life Sciences (DILS)*, LNCS 2994, Leipzig, Germany, March 2004.

[5] S. Bowers and B. Ludäscher. Actor-Oriented Design of Scientific Workflows. In *24st Intl. Conference on Conceptual Modeling (ER)*, Klagenfurt, Austria, October 2005. Springer.

[6] S. Bowers and B. Ludäscher. A Calculus for Propagating Semantic Annotations through Scientific Workflow Queries. In *Query Languages and Query Processing (QLQP): 11th Intl. Workshop on Foundations of Models and Languages for Data and Objects*, LNCS, Munich, Germany, March 2006.

[7] S. Bowers, T. McPhillips, B. Ludäscher, S. Cohen, and S. B. Davidson. A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows. In Moreau and Foster [25].

[8] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. Principles of Programming with Complex Objects and Collection Types. *Theoretical Computer Science*, 149(1):3–48, 1995.

[9] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *IEEE/ACM Supercomputing Conference*, Dallas, TX, 2000.

[10] S. B. Davidson, G. C. Overton, V. Tannen, and L. Wong. BioKleisli: A Digital Library for Biomedical Researchers. *Int. J. on Digital Libraries*, 1(1):36–53, 1997.

[11] S. R. Eddy. Antedisciplinary Science. *PLoS Computational Biology*, 1(1), 2005.

[12] G. C. Fox and D. Gannon, editors. *Concurrency and Computation: Practice & Experience, Special Issue: Workflow in Grid Systems*, volume 18. Wiley, August 2006.

[13] T. Grust. Monad Comprehensions: A Versatile Representation for Queries. In P. M. Gray, L. Kerschberg, P. J. King, and A. Poulovassilis, editors, *The Functional Approach to Data Management: Modeling, Analyzing and Integrating Heterogeneous Data*. Springer, 2003.

[14] J. Hughes. Generalising Monads to Arrows. *Science of Computer Programming*, 37(1-3):67–111, 2000.

[15] D. Hull, R. Stevens, P. Lord, C. Wroe, and C. Goble. Treating "Shimantic Web" Syndrome with Ontologies. In *First AKT Workshop on Semantic Web Services*, The Open University, Milton Keynes, UK, 2004.

[16] G. Kahn. The Semantics of a Simple Language for Parallel Programming. In J. L. Rosenfeld, editor, *Proc. of the IFIP Congress 74*, pages 471–475. North-Holland, 1974.

[17] S. A. Klasky, B. Ludäscher, and M. Parashar. The Center for Plasma Edge Simulation Workflow Requirements. In *Post-ICDE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow)*, Atlanta, GA, April 2006.

[18] E. A. Lee and S. Neuendorffer. Actor-Oriented Models for Codesign: Balancing Re-Use and Performance. pages 33–56, 2004.

[19] E. A. Lee and T. Parks. Dataflow Process Networks. *Proceedings of the IEEE*, 83(5):773–799, May 1995.

[20] B. Ludäscher and I. Altintas. On Providing Declarative Design and Programming Constructs for Scientific Workflows based on Process Networks. Technical Report SciDAC-SPA-TN-2003-01, San Diego Supercomputer Center, 2003.

[21] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. [12], pages 1039–1065.

[22] B. Ludäscher and C. A. Goble. Guest Editors' Introduction to the Special Section on Scientific Workflows. *SIGMOD Record*, 34(3), 2005.

[23] T. McPhillips, S. Bowers, and B. Ludäscher. Collection-Oriented Scientific Workflows for Integrating and Analyzing Biological Data. In *3rd Intl. Workshop on Data Integration in the Life Sciences (DILS)*, LNCS, European Bioinformatics Institute, Hinxton, UK, July 2006. Springer.

[24] T. M. McPhillips and S. Bowers. An Approach for Pipelining Nested Collections in Scientific Workflows. *SIGMOD Record*, 34(3):12–17, 2005.

[25] L. Moreau and I. Foster, editors. *Intl. Provenance and Annotation Workshop (IPAW)*, LNCS, Chicago, May 2006. Springer.

[26] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. [12].

[27] L. Peterson, E. Yin, D. Nelson, I. Altintas, B. Ludäscher, T. Critchlow, A. J. Wyrobek, and M. A. Coleman. Mining the Frequency Distribution of Transcription Factor Binding Sites of Ionizing Radiation Responsive Genes. In *New Horizons in Genomics, DOE/SC-0071*, Santa Fe, New Mexico, 2003.

[28] H. J. Reekie. *Realtime Signal Processing: Dataflow, Visual, and Functional Programming*. PhD thesis, School of Electrical Engineering, University of Technology, Sydney, 1995.

[29] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, editors. *Workflows for e-Science: Scientific Workflows for Grids*. Springer, 2006.

[30] P. Wadler. XQuery: A Typed Functional Language for Querying XML. In J. Jeuring and S. L. P. Jones, editors, *Advanced Functional Programming*, volume 2638 of *Lecture Notes in Computer Science*, pages 188–212. Springer, 2002.