A Modeling and Execution Environment for Distributed Scientific Workflows*

Ilkay Altintas♮ Sangeeta Bhagwanani+ David Buttler* Sandeep Chandra+ Zhengang Cheng+
Matthew A. Coleman‡ Terence Critchlow‡ Amarnath Gupta♮ Wei Han* Ling Liu*
Bertram Ludäscher♮ Calton Pu* Reagan Moore♮ Arie Shoshani† Mladen Vouk+

1 Introduction

The Scientific Data Management Center project (short: SDM) is part of a large research program sponsored by the US Department of Energy (DOE) to enable Scientific Discovery through Advanced Computing [SDM02, Sci]. SDM brings together research teams from DOE labs and universities to address and resolve novel data management challenges that arise due to the new data and information centric ways in which science is conducted today.

This demonstration illustrates how a domain scientist can perform a complex scientific task by interleaving data access, querying, and manipulation, as well as analytical steps and computations in complex, problem specific ways. We show how our system is used by a geneticist for solving the problem of discovering so-called "co-regulated" genes by interlinking data and computation from several web sites, local computations, as well as local and remote databases. The main distinctive features of our system (compared, *e.g.*, to the ZOO environment [ILGP96]) include (i) executable workflows run as web services, (ii) abstract workflows employ concept names and semantic types that are higher-level (and thus more "scientist friendly") than executable workflows, and (iii) our system supports automatic translation of the latter into the former.

A Scientist's Problem: Promoter Identification Workflow (PIW). Through the Human Genome Sequencing Project a wealth of information has been gained at the nucleotide level. With the advent of DNA-based microarrays the wealth of data for interpretation is quickly becoming daunting. A starting point for discovery is to link genomic biology approaches such as microarrays with bioinformatics to *identify and characterize eukaryotic promoters* – here, we call this the *promoter identification workflow* or PIW.[1] To clearly identify co-regulated groups of genes, high throughput computational molecular biology tools are first needed that are scalable for carrying out a variety of tasks such as identifying DNA sequences of interest, comparison of DNA sequences, and identification of transcription factor binding sites, etc.

Some of these steps can be executed by querying web-accessible databases and computation resources. However, using web sources "as-is" to enact scientific workflows requires many manual and thus time-consuming and error-prone steps. It is desirable to automate the scientific workflows such as the PIW as much as possible. A number of information technology and database challenges have to be overcome:

- Most current web sources are made for human interaction and thus do not lend themselves easily to automation. Semiautomatic or automatic *wrapping techniques* have to be applied in order to turn interactive web sources into remote function invocations and database queries.
- An *execution environment* for running distributed workflows over the web has to be devised. This includes capabilities for monitoring workflow execution, checkpointing, and re-running or resuming suspended runs. This is hard due to the autonomous nature of sources, their heterogeneous and limited access capabilities, and their occasional, unpredictable downtimes.
- The *design of scientific workflows* poses unique challenges both to the domain scientist who drives the overall design and the IT expert who is charged with defining the specific data and control flow. This is due to the complexity of the scientific data, the complexity of the (often hidden) semantic links between the different data sources, and the complexity of the syntactic and procedural intricacies that have to be overcome when chaining together actual web sources in the PIW.

*Georgia Institute of Technology, †Lawrence Berkeley Laboratory (LBL), ‡Lawrence Livermore National Laboratory (LLNL), +North Carolina State University (NCSU), ♮San Diego Supercomputer Center (SDSC). This work was supported by DOE LLNL contract No. W-7405-Eng-48, SciDAC/SDM contract No. DE-FC02-01ER25486, and NSF grant No. ITR 0225676 (SEEK).

[1] a promoter is a subsequence of a chromosome that sits close to a gene and regulates its activity

Our system demonstration illustrates how the above problems are addressed and resolved using a web service oriented execution environment based on XWraped sources [LPH01] and a semantic mediation approach [LGM01]. We present the overall approach and system components in the following.

2 Scientific Workflow Approach and Architecture

Our system for managing scientific workflows borrows some ideas from database mediation, most notably the use of wrappers to provide uniform data access (in XML syntax) to heterogeneous web sources. However, the promoter identification workflow PIW cannot be modeled using simple declarative database queries. Here, by (scientific) *workflow* we mean a directed graph over domain-specific tasks and control structures. A scientific workflow in our system can be seen as a user-specified abstract "query plan" whose operators include not only the usual database operations but domain-specific functions such as gene_sequence_homology and cluster_analysis. In our architecture, an extensible repository of such *abstract tasks* is made available to the scientist.[2] Thus the core idea of our approach and system is that the scientist designs an *abstract workflow* (AWF) from the repository of problem-oriented abstract tasks while the system tries to derive from AWF an *executable workflow* (EWF) in terms of the available web services. The use of two separate vocabularies of abstract tasks and executable tasks (the actual available web services) has several advantages:

By hiding the low-level details and intricacies of actual web services, the scientist user can focus on the design of the given scientific workflow at the conceptual level. For example, there are several web sources that we have wrapped as web services that implement the abstract task of gene_sequence_homology. Taken together such services provide a similar functionality, but are often implemented using different algorithms and/or applied to different data sets, *e.g.*, see [BLAa, FAS02, BLAb].

We provide a mechanism to specify how the functionality of abstract tasks can be defined in terms of the available executable web services. We call this the *abstract-as-view* (AAV) mapping, since we express the abstract task as a *view* over the existing executable tasks. Our view definition language for the AAV mapping differs, *e.g.*, from Datalog through its use of procedural constructs (*e.g.*, task order, parameter binding patterns, and guarded commands), data types, and semantic types. Differences among web services that play a role in the implementation of an abstract task are hidden in the AAV mapping and do not interfere with the design of the abstract workflow. Instead the system creates a distributed executable workflow plan EWF from AWF and AAV.

[2] We are not aware of a specific standard API for recurring tasks such as gene_sequence_homology. Once such standards emerge, they can be easily added to our repository.

2.1 System Architecture and Components

Figure 1 depicts the system architecture. The user interacts with a GUI called the *workflow pilot* for designing the abstract workflow AWF and enacting and monitoring the exectuable workflow EWF. The *workflow compiler* translates AWF to EWF using the AAV mapping. In order to assist the user in the abstract workflow design, abstract tasks have associated semantic types that come from a linked domain ontology. Using semantic types, the system can automatically introduce conversion steps (see below). Abstract tasks are stored in the *abstract task repository*, while executable tasks are stored in a web services repository. The signatures of web services are described using a WSDL extension; process communication is done using SOAP. After invoking the WF-Compiler, the WF-Pilot can display and run the executable workflow plan. Executing EWF includes invocation of generated XML wrappers.

WF-Pilot. The WF-Pilot is a Java/Swing based GUI and allows the user to design the abstract workflow in an intuitive manner using graphical primitives for the WF language constructs explained below. The WF-Pilot is also tightly coupled with the WF-engine, *i.e.*, a web service-oriented runtime environment for enacting and monitoring workflows. Features of the WF-Pilot include checkpointing of intermediate data nodes, *e.g.*, for subsequent analysis or reruns of parts of the workflow with different parameters.

WF-Compiler. An important step during the AWF $\stackrel{\text{AAV}}{\longrightarrow}$ EWF translation is to ensure that abstract tasks are "fed" with correctly typed data, and that binding pattern restrictions of executable tasks (web services) are observed. The underlying language for defining workflows is based on the following nodes and edges:

- *Task nodes* represent functions such as gene_sequence_homology. A task node has various *ports* connecting it to other nodes, *i.e.*, data-in, data-out (for the main data flow), parameter-in (for control parameters), and exit-code. The latter is used to determine whether a task has been

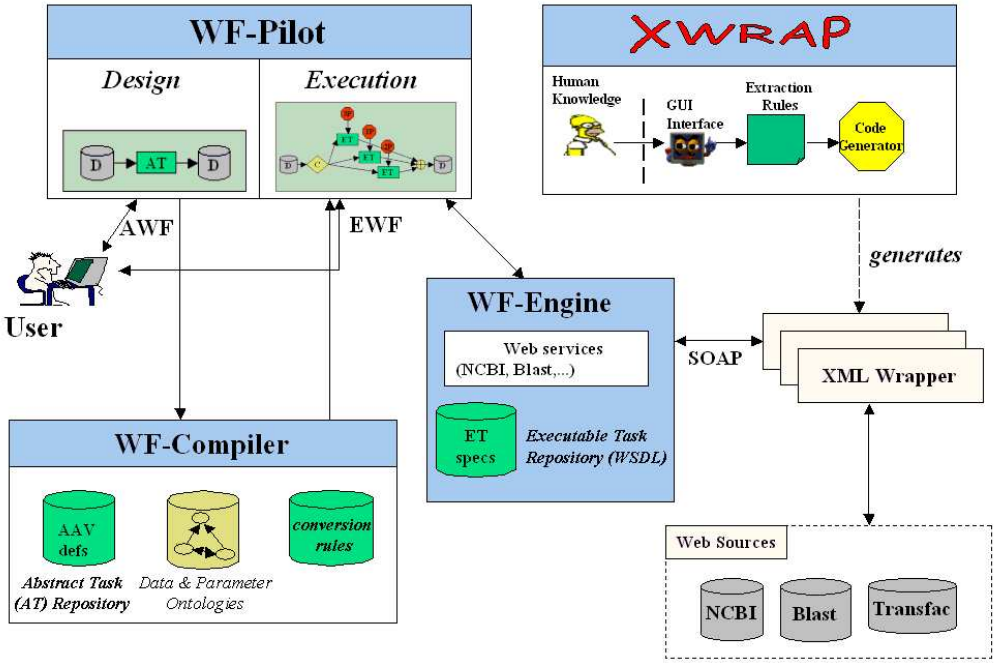

Figure 1. Scientific Workflow Management System architecture

executed successfully and can be used to specify contingency actions.

- *Control nodes* are used to specify branching based on runtime conditions. A control node has data-in and parameter-in ports and k *conditional* data-out ports, each guarded with a condition φ_i, for $i = 1, \ldots, k$. Data (and control) flows through data-out$_k$ iff φ_k is true.

- *Data nodes* characterize the data types and semantic types (concept names from a shared ontology) of the data flowing between tasks, while *parameter nodes* characterize the data types and semantic types of the control parameters of tasks.

XWRAP-Composer. This component generates wrappers that extract relevant information from different interlinked pages and compose a wrapped XML document, containing the combined information from those pages. In the PIW example, each executable step requires access to some Bioinformatics web sources. For instance, for the task of identification of DNA sequences that may have similar reactions to those gene expression profiles resulting from the microarray experiments, one needs to use the full sequence obtained from the search result of GenBank database to run a BLAST query [BLAa] over NCBI [NCB02] sources. To obtain a set of homologues (fragments of similar DNA sequences) from BLAST, one needs to traverse the outgoing links from the BLAST summary page to get to each of the BLAST detail pages. The XWRAP-Composer encodes both the link reference flow graph and the data extraction flow graph over multiple pages and utilizes them to build wrapper programs that capture the relevant information from multiple selected pages using a single web service invocation. XWRAP-Composer is built on top of the XWRAP toolkit [LPH01].

3 Demonstration

The system demonstration will show our current prototype developed for the use of a real-world molecular biologist trying to find "co-regulated genes" using the PIW. The demonstration highlights both the usage and several internal features of the prototype:

AWF Design Mode. The end-user uses the WF-Pilot to define the abstract promoter identification workflow. To accomplish any of the PIW abstract tasks, the user will search through the abstract task library. Suppose the user chooses the task gene_sequence_homology as a way of finding promoters. As the task is graphically put in the WF-Pilot workspace, the system notices that there are multiple possible overlapping instantiations for this task. This triggers a *semantic disambiguation routine* that asks the user whether any of the predefined semantic preconditions apply to the input (*e.g., "Is the length of the sequence always less then 5000?"*). As the user answers these questions, the system selects the appropriate instantiations, or possibly keeps several of them

as alternate sources to be selected based on runtime availability and performance. As the user builds an abstract workflow by drawing edges between consecutive abstract tasks, the system checks for *data and semantic type compatibility* between the output types and input types of the predecessor and successor tasks, respectively. For example, the result of a BLAST search from the NCBI web site produces a genomic sequence, and the TRANSFAC site [TRA02] accepts a genomic sequence. However, the BLAST output can come from either of the two gene strands (5' or 3'), but for the subsequent comparison using TRANSFAC derived data, a single orientation is needed. Therefore, the system **inserts** a *semantic type conversion rule* that complements and reverses the sequence (complementation swaps the bases: A ↔ T and G ↔ C).

AAV Design Mode. In this mode, a workflow engineer defines single abstract tasks in terms of executable tasks using the WF-Pilot GUI or a text-oriented interface. For example, the abstract task of cluster_analysis is first defined as a procedure which, given microarray observational data returns a hierarchical grouping of genes, together with additional metadata for each gene (like its distance from the cluster mean). The AAV designer creates a concrete instance of cluster_analysis by associating it with a specific cluster analysis tool such as CLUSFAVOR. Like abstract tasks, executable instances of abstract tasks come with pre- and post-conditions that guard their applicability. In general the AAV designer may create multiple executable instances of the same abstract task, *e.g.*, gene_sequence_homology will be defined in terms of both the BLAST web-site from NCBI and the BLAT web-site from UC Santa Cruz. In this case, the AAV designer has to specify the conditions that allow the system to select at runtime one or more of the executable tasks in place of the abstract task. If the conditions do not determine a unique instantiation, the user is prompted for a decision at runtime. Possible relationships between different executable tasks include equivalent (in which case the system can pick either one, based on availability and performance), complementary (in which case the designer has to state mutually disjoint conditions), and overlapping (so task selection at runtime usually involves prompting the user).

EWF Execution Mode. Similar to the AWF, the EWF can be viewed and even edited through the WF-Pilot. In order to enact this workflow, the user is first prompted for remaining runtime parameters that have not been specified as part of the AWF. In execution mode, the user can also add ad-hoc breakpoints to inspect intermediate results, and decide which intermediate data should be made persistent. In general, all user-defined parameter settings and answers to interactive steps are logged by the system, thus allowing the user to rerun a workflow with the same or adjusted parameter settings.

Given the growing complexity and volume of the genomic information, robust computational approaches such as the PIW approach are especially needed. As presented, this is a novel system that can link microarray data to genomic database information and pass it to multiple tools. The resulting PIW database and workflow tools become a predictive model to form and test hypotheses related effects based on gene/pathway interactions. There are currently no easily accessible methodologies that allow for what the PIW demo offers. Our model provides the biologist with a quickly adaptable way to identify genes and functional pathways that are coincidentally related to gene expression patterns. As new algorithms and databases are developed they can be quickly absorbed by the workflow, so that the biologist can focus his work on new research hypotheses instead of spending his time on data management.

References

[BCC+02] D. Buttler, M. Coleman, T. Critchlow, R. Fileto, W. Han, C. Pu, D. Rocco, and L. Xiong. Querying Multiple Bioinformatics Information Sources: Can Semantic Web Research Help? *SIGMOD Record*, 31(4), 2002.

[BLAa] Basic Local Alignment Search Tool (BLAST). `http://www.ncbi.nlm.nih.gov/BLAST/`, 2002.

[BLAb] UCSC Genome Bioinformatics – BLAT FAQ. `http://genome.ucsc.edu/FAQ.html#188`.

[FAS02] European Bioinformatics Institute – FASTA. `http://www.ebi.ac.uk/fasta33/`, 2002.

[ILGP96] Y. E. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti. ZOO: A Desktop Experiment Management Environment. In *VLDB*, 1996.

[LGM01] B. Ludäscher, A. Gupta, and M. E. Martone. Model-Based Mediation with Domain Maps. In *Intl. Conf. on Data Engineering (ICDE)*, 2001.

[LPH01] L. Liu, C. Pu, and W. Han. An XML-Enabled Data Extraction Tool for Web Sources. *Information Systems, Special Issue on Data Extraction, Cleaning, and Reconciliation*, 2001.

[NCB02] National Center for Biotechnology Information (NCBI). `http://www.ncbi.nlm.nih.gov/`, 2002.

[Sci] Scientific Discovery through Advanced Computing (SciDAC), Department of Energy (DOE). `http://www.er.doe.gov/scidac/`.

[SDM02] Scientific Data Management Center (SDM). `http://sdm.lbl.gov/sdmcenter/`, 2002.

[TRA02] TRANSFAC – Transcription Factor Database. `http://transfac.gbf.de/TRANSFAC/`, 2002.