Domain-Specific Languages for Composing Signature Discovery Workflows

Ferosh Jacob[1,2], Adam Wynne[1], Yan Liu[1], Nathan Baker[1], and Jeff Gray[2]
fjacob@crimson.ua.edu, Adam.Wynne, Yan.Liu, Nathan.Baker@pnnl.gov, gray@cs.ua.edu
[1]Pacific Northwest National Laboratory
Richland, WA
[2]Department of Computer Science
University of Alabama, AL

ABSTRACT

Domain-agnostic signature discovery entails study across multiple scientific disciplines. The breadth and cross-disciplinary nature of this work requires that existing executable applications be integrated with new capabilities into workflows, representing a wide range of user tasks. An algorithm may be written in multiple programming languages for various hardware platforms, and so workflow composition requires integrating executables from any number of remote hosts. This raises an engineering issue on how to generate web service wrappers for these heterogeneous executables and to compose them into a scientific workflow environment (e.g., Taverna). In this position paper, we summarize our work on two simple Domain-Specific Languages (DSLs) that automate these processes. Our Service Description Language (SDL) describes key elements of a signature discovery service and automatically generates its implementation code. The Workflow Description Language (WDL) describes the pipeline of services and generates deployable artifacts for the Taverna workflow management system. We demonstrate our approach with a real-world workflow composed of services wrapping remote executables.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Domain-specific architectures

General Terms

Design

Keywords

SDL, WDL, workflow, modeling

1. INTRODUCTION

A signature is a unique or distinguishing measurement, pattern, or collection of data that detects, characterizes, or predicts a target phenomenon (object, action, or behavior) of interest. Signatures are valuable to a wide range of application domains - including medicine, network security, and explosives detection - for anticipating future events, diagnosing current conditions, and analyzing past events. However, current approaches suffer from a lack of re-use of existing algorithms, tools, and techniques across application domains and scientific disciplines. At the Pacific Northwest National Laboratory (PNNL), we have been developing a generalized signature development methodology that is applicable to any signature discovery problem, along with a service-oriented platform that implements the methodology [1]. We refer to this combination of methodology and platform as the Analytic Framework (AF). In the AF, legacy code is wrapped and exposed as web services, which are orchestrated to create re-usable tasks that can be retrieved and executed by users. Currently, we are using Taverna [2] for service orchestration, but our approach is applicable to any workflow management system. This service-oriented approach provides for a robust platform, but also causes challenges for scientists in integrating their algorithms into the system.

1.1 Accidental complexity of creating service wrappers

One challenge is the complexity of building robust services from legacy code. To make existing executable applications available as services in our platform, we follow best practices by employing the Legacy Wrapper pattern [3]. This encapsulates existing logic, while providing a standard interface so that services can be orchestrated with each other to create re-usable workflows. Informally, we refer to these services as *wrappers*. Programmers creating wrapper code for an existing executable typically follow a common set of steps: 1) identify the input files and output files in the script; 2) retrieve the input files from the data management system; 3) run the executable; and 4) upload the output files to the data management system. This process for converting an executable often results in significant extra code and operations. For example, in our system, manually wrapping a simple script that has a single input and output file requires 121 lines of Java code (in five Java classes) and 35 lines of XML code (in two files). Additionally, the engineer needs to then manually import each web service into the Taverna workbench so that they can be composed into a workflow.

1.2 Lack of end-user environment support

Many scientists are not familiar with service-oriented software technologies. In this case, they will be forced to seek the help of software developers to make the web services encapsulating their executables available in the workflow environment. This technology barrier may degrade the efficiency of sharing signature discovery algorithms, because any changes or bug fixes of an algorithm require a dedicated programmer to navigate through the engineering process.

We applied Domain-Specific Modeling (DSM) techniques to model the process of wrapping remote executables. The executables are wrapped inside AF web services using a

```
service submitBlast {
  use ssh_oly;
  cmd "sh runJob.sh";
  resource "jobScript.sh", "runJob.sh";
  in doc blossum, params, fasta;
  out jobID, outDir;
   /*
    *Inside runJob.sh
    * echo "jobID=$SLURM_JOBID" >.properties
    * echo "outDir=$OUTDIR" >>.properties
    */
}
```

Figure 1: Service description for BLAST submission

Domain-Specific Language (DSL) [4] called the Service Description Language (SDL). The SDL-created web services can then be used to compose workflows using another DSL, called the Workflow Description Language (WDL). In this position paper, we illustrate our approach with a set of services and a workflow that are commonly used in our signature discovery work.

2. EXAMPLE APPLICATION: BLAST EXECUTION

BLAST is an executable commonly used to find regions of similarity between biological sequences. Scientists at PNNL have parallelized BLAST [5] and applied it to other application domains, effectively making it part of set of generalized sequence-based signature discovery tools. In the AF, a BLAST workflow is usually executed in three steps: 1) submit a BLAST job to the queue management system for a cluster; 2) check the status of the job; and 3) download the output files upon completion of the job. We model and orchestrate the scripts to execute the BLAST workflow using our approach in two steps, as explained in the following sub sections.

2.1 Creating service wrappers using SDL

We have used the SDL to create service wrappers for each step of the workflow. The SDL code to create a job submission service is shown in Figure 1. A BLAST job is submitted using two script files "jobScript.sh" (a SLURM[1] file) and "runJob.sh" (a BASH file). The script file "runJob.sh" executes the SLURM file and writes `jobID` and `outDir` to the `.properties` file. As shown in the figure, the `submitBlast` service has two outputs, the execution directory (`outDir`) and the job identifier (`jobID`). Both outputs are declared as the default type (string type); hence, the generated code downloads and reads their values from a `.properties` file.

Other services (`blastResult` and `checkJob`) are command SDL service wrappers (no script files) and are not shown. The service `checkJob` checks the status of a given `jobID` and returns status "Running," "Pending," or "Done." The service `blastResult` downloads the files from a given directory.

2.2 Creating workflows using WDL

A WDL file generates a Taverna workflow based on the descriptions specified by the user. A WDL workflow involves communication and interactions of various service wrappers among each other and also with other workflows.

[1]SLURM, `https://computing.llnl.gov/linux/slurm/`

```
 1 use "SigAnalysis.sdl"
 2 workflow BlastSearch (
 3 in blosum, in params, in fasta,
 4   out outFile, out status){
 5
 6   //Linking workfow inputs and submitBlast service
 7   blosum->submitBlast.blosum
 8   params -> submitBlast.params
 9   fasta -> submitBlast.fasta
10
11   //Linking sub-workflow with main (Loop)
12   call checkJob
13     till status="Done"
14     with  submitBlast.jobID,  status
15
16   //Linking blastResult after checkJob
17   submitBlast.outDir
18      ->blastResult.outDir after checkJob
19
20   //Linking workflow output and blastResult
21   blastResult.outFile->outFile
22 }
23 /*
24  *Sub-workflow checkJob
25  */
26 workflow checkJob (in wf_jobID,  out wf_status){
27
28   wf_jobID->jobStatus.jobID
29   jobStatus.status->wf_status
30 }
```

Figure 2: Workflow description for BLAST

A WDL file can contain many workflows, but the top-most workflow is considered the "main" workflow with all others treated as sub-workflows. In Figure 2, two workflows are defined, `BlastSearch` (main workflow, lines 2-22) and `checkJob` (sub-workflow, lines 26-30). The main workflow takes input required for the BLAST execution as input (line 3) and, a file (`outFile`) and status as output (line 4). A service defined in the WDL file is executed automatically as soon as all the inputs for that service are set. Three inputs required for the service are set using the workflow inputs (see Figure 2, lines 7-9), thus the `submitBlast` service is executed after line 9. In some cases, this may not be a desired behavior. As an example, for the BLAST execution, the `blastResult` service that downloads the output files needs to wait for the BLAST job to complete. To allow a service to be executed only after the specified workflow or service, the "after" keyword is provided. The "after" keyword can be added to any service invocation (Figure 2, line 18). As shown in the figure, (lines 12-14), using the `call-till-with` structure, the main workflow can iteratively "call" a sub workflow "till" it meets a condition ŞwithŤ main workflow ports. Hence, the "call-till-with" structure is a replacement for function calls and loops in WDL. The executable workflow file opened in the Taverna workbench is shown in Figure 3.

3. IMPLEMENTATION OVERVIEW

The overall framework of the tool to generate both wrappers and workflows is given in Figure 4. This tool expects the scripts in a template format with template variables as defined in service descriptions. The outputs are Web service wrappers and a workflow file deployable to a workflow engine (such as Taverna). Each Web service wrapper is created from scripts and the associated SDL files; and the workflow file is created using both SDL and WDL files. The code generation occurs in two stages:

1. Web application creation. The tool creates Web ser-

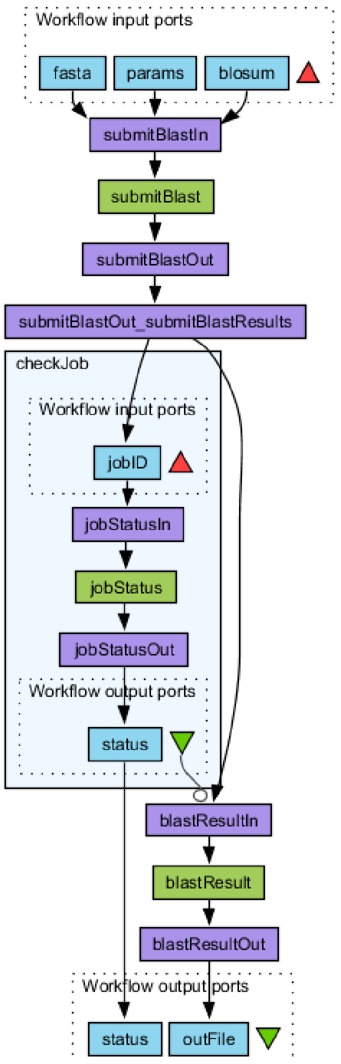

Figure 3: BLAST execution workflow in Taverna

vices from the SDL files that describe the key elements of a script;

2. Workflow creation. The SDL file from the first stage defines a deployable SDI Web services. It is passed together with the WDL file to create the workflow constructs. These constructs are the basic elements for the Taverna engine to create a workflow as defined in WDL.

At runtime, the function of the script can be executed in the remote host through an SSH session with the help of existing signature discovery libraries. These libraries are responsible for making the input files available before execution and uploading the output files to a dedicated document management system after execution. A template engine is used to apply runtime values of the services on scripts.

4. CONCLUSION

We successfully designed and implemented two DSLs (SDL and WDL) for converting remote executables into scientific

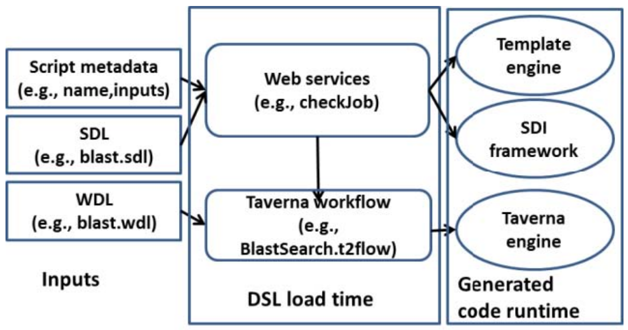

Figure 4: Block diagram showing implementation

workflows. SDL can generate services that are deployable in a signature discovery workflow using WDL. We separated the domain-specific information required to create the workflows from the accidental complexities introduced by web services and the Taverna workflow engine, which allows end-users (scientists) to design and develop workflows. The two DSLs are introduced using a BLAST execution workflow, a frequently used workflow in scientific community. This project is an initial step towards scientists designing and developing complex workflows in various signature domains utilizing the best algorithms available on best resources. As a future work, we plan to evaluate our work by comparison with other existing tools through empirical studies with human subjects. Providing a workflow development environment specifically for signature workflows is also included as our future work.

5. REFERENCES

[1] Signature discovery initiative. `http://signatures.pnnl.gov/`.

[2] Taverna workflow management system. `www.taverna.org.uk/`.

[3] T. Erl. *SOA Design Patterns*. Prentice Hall PTR, 1st edition, 2009.

[4] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

[5] C. Oehmen and J. Nieplocha. ScalaBLAST: A scalable implementation of blast for high-performance data-intensive bioinformatics analysis. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):740–749, 2006.