Towards Scalable Search-Based Model Engineering with MDEOptimiser Scale

Alexandru Burdusel
Department of Informatics
King's College London
30 Aldwych, London, WC2B 4BG
alexandru.burdusel@kcl.ac.uk

Steffen Zschaler
Department of Informatics
King's College London
30 Aldwych, London, WC2B 4BG
szschaler@acm.org

***Abstract*—Running scientific experiments using search-based model engineering (SBME) tools is a complex task, that poses a number of challenges, starting from defining an experiment workflow, to parameter tuning, finding optimal computational resources to run on, collecting and interpreting metrics and making the entire process easily reproducible.**

Despite the proliferation of easily accessible hardware, as a result of the increased availability of infrastructure-as-a-service providers, many SBME tools are rarely using this technology for accelerating experimentation. Running many experiments on a single machine implies much longer waiting times and reduces the ability to increase the speed of iterations when doing SBME research, thus, slowing down the entire process.

In this paper, we introduce a domain-specific language (DSL) and a framework that can be used to configure and run experiments at scale, on cloud infrastructure, in a reproducible way. We will describe our DSL and framework architecture along with an example to showcase how a case study can be evaluated using two different model optimisation tools.

***Index Terms*—model driven engineering, reproducible research, evolutionary search, search based model engineering, workflow, cloud, middleware**

I. Introduction

Search-Based Model Engineering (SBME) is a methodology that combines search techniques with Model-Driven Engineering, to help domain experts find optimal models that satisfy metamodel constraints [1]–[5]. One problem faced by SBME researchers is the long-running time and considerable computational resources required by SBME tools to find good results.

Typical research experiments aim to evaluate several configurations, for at least one SBME tool. Each configuration consists of a set of parameters, case studies, and input models. Such systematic approaches generate a large number of experiment permutations and evaluating such experiments often requires running the tools for a long time. In addition to this, to achieve statistically significant results, each configuration experiment must be repeated at least 30 times [6]. This requirement considerably increases the time and resources required to run SBME experiments.

SBME researchers are faced with at least two technical challenges: 1. Running experiments at scale efficiently, and 2. Comparing results obtained by different runs for different algorithms, tools, or configurations. To compare experiment results, the metrics generated by the evaluated configurations have to be interpreted and processed. This task is often done using a data processing script and requires that the data is first normalised so that it can be parsed and statistically evaluated. Most SBME tools employ their own standard for the frequency and the format of the data collected during the experiments. This data format discrepancy requires that researchers have to write their own scripts to translate between various data schemas and formats, to a common schema and format, in order to process the experiment data. This translation step makes the process cumbersome and error prone.

In this paper, we are proposing MDEOptimiser Scale (MDEO Scale), a task distribution DSL and framework aimed at SMBE tools. MDEO Scale allows users to schedule SBME experiments on IaaS platforms, to parallelise the experiment configurations and reduce the time required to run them. The tool offers a data collection interface, that aims to standardise the format for collecting data from SBME experiments, and offers support for calculating common experiment metrics and summary statistics for single and multi-objective search problems. In this paper we focus on SBME problems, however, the approach we describe can be extended to support the execution of any experiments which evaluate search techniques.

In this paper we make the following contributions:

1) we propose a DSL that allows users to specify experiment configurations for SMBE tools;
2) we describe the architecture of a framework that can run experiments on IaaS hardware;
3) we propose a framework for capturing and performing common analysis of experiment data.

The remainder of this paper is structured as follows: In Sect. II we introduce relevant background. Section IV contains the main contributions, describing our motivations, design goals and tool architecture. Section VI describes the DSL and the implementation details of our approach and in Section VII we discuss related work.

II. Background

In this section, we introduce concepts and definitions that are relevant to the research described in this paper. We give definitions for key terms used in our DSL, followed by an introduction to IaaS concepts. We then briefly introduce

SBME, followed by an overview of two different SBME approaches.

A. Infrastructure-as-a-Service

Infrastructure-as-a-Service (IaaS) is a concept used to describe services that offer users the ability to rent, computational hardware on-demand, over the Internet. IaaS services can scale up and down, based on the user requirements, and the computational resources used are billed by fixed time intervals, such as per hour or second. Some IaaS providers, in addition to computational resources, also offer additional services such as automatic scaling, analytics management or machine learning pipelines.

In this paper, we use Amazon Web Services[1] (AWS) as an IaaS provider. The AWS IaaS product offering computational resources is called EC2. When using EC2, users can opt to use several types of instances, depending on the type of tenancy on the underlying server they prefer. For our tool configuration, we use EC2 Spot instances, which are unused capacity instances from the AWS inventory. EC2 Spot instances can be rented at a price discounted up to 90%, a significant cost reduction compared to running similar experiments on normal EC2 instances [7]. The disadvantage of using EC2 Spot instances is that they can be terminated unexpectedly in cases when the unused computational capacity they are generated from, becomes needed by another IaaS service.

AWS Batch[2] is a service offered by AWS, that enables users to schedule batch computing jobs to run on AWS infrastructure. Batch, automatically provisions the optimal quantity and type of compute resources and allocates them for executing the user-submitted Batch jobs. Users can interact with AWS Batch using the web-based AWS Management Console or the AWS REST API.

B. Search-Based Model Engineering

Search-based model engineering is a methodology that combines search-based software engineering (SBSE) ideas with Model-Driven Engineering (MDE). SBSE is a methodology that applies meta-heuristic search algorithms to software engineering problems to find near-optimal solutions [8]. To specify SBSE problems a number of elements are needed: • a problem domain representation • a method for specifying solution candidates in the problem domain; • a set of guidance functions that can indicate solution quality • a set of search operators to manipulate solution candidates.

SBME is based on the idea that domain experts already have the problems specified in a domain-specific modelling language, and uses MDE artifacts to specify SBSE problems. More specifically, the problem domain is specified by the metamodel, solution candidates are instance models of the metamodel, guidance functions are model queries and search operators are model transformations.

[1]http://aws.amazon.com

[2]http://aws.amazon.com/batch/

Two paradigms have been proposed for solving SBME problems. In this section we are going to use two example implementations.

MDEOptimiser (MDEO) is an SBME approach that aims to run the search directly over models, by using the models themselves as a representation for solution candidates [5]. The search space is explored by applying endogenous transformations to solution models and evaluating the quality of the resulting models using a model query expressed using Java or OCL.

MOMoT uses an alternative approach for specifying SBME problems [9]. In this approach, solution candidates are encoded as chains of model transformations applied to a problem input models. New solutions are generated by applying mutations and crossover to the model transformations vector, and then re-applying the transformations chain to the initial input model and evaluating the quality of the obtained solution model using a model query in a similar way to MDEO.

The two SBME tools described in this section use MOEAFramework [3] for implementing the supported optimisation algorithms. MOEAFramework is a Java-based library, which offers a number of single and multi-objective algorithms along with the functionality for instrumenting and evaluating their performance.

In this paper, we are demonstrating our MDEO Scale DSL using the two SBME approaches introduced in this section.

III. Running Example

In this section, we introduce an example optimisation problem to show how our proposed DSL can be used to configure experiments using different optimisation tools. The Class Responsibility Problem (CRA) is a case study from the field of software engineering [10].

The goal of this problem is to transform a software application implemented using a procedural approach to an object-oriented architecture while finding optimal values for cohesion and coupling. The quality of the produced solutions is measured using the CRA index defined in [10], as a single objective. The problem supplies a responsibility dependency graph, that contains a set of functions and attributes with dependencies between them. In the metamodel, these entities are instances of the abstract type Feature.

To solve this problem, the user is required to create `Class` entities in the `ClassModel` and assign `Features` to them such that: all `Features` are assigned to a `Class` and the model with the highest CRA index value is found. The problem has an additional constraint requiring that each Feature is assigned to only one `Class` at a time.

IV. Language and infrastructure design

In this section, we introduce the motivation for developing the MDEO Scale tool. We then continue to list the design goals and give an overview of the proposed prototype architecture and the supported infrastructure.

[3]http://moeaframework.org/

A. Definitions

In this paper, we use the following terms to describe our proposed DSL:

A *Task* refers to a single executable problem instance, with preconfigured inputs. An example of a `Task` would be running an SBME tool, for a specific input model from an evaluated case study.

A *Batch* refers to a single execution of a `Task`. When evaluating case studies using SBME tools that use meta-heuristic algorithms, `Tasks` are generally executed multiple times, to ensure the results obtained are statistically significant. Each `Task` execution counts as a `batch`.

An *Experiment* refers to a case study for which an effect needs to be evaluated using a collection of tasks that have different configuration inputs. Each experiment may contain any number of input models, and `Tasks` are grouped by the input model they evaluate.

B. Motivation and Use Cases

The motivation for implementing MDEO Scale came from the need to run SBME experiments using multiple tools and configurations while reducing the waiting time for the results to be ready and being able to easily interpret the results.

During the evaluation of the work published in [3], the authors ran 498 configurations using three tools that required over 2500 hours of MDEO runtime. Running these experiments on a single machine, required over 100 days of continuous runtime. By parallelising the experiment tasks using AWS Batch, the authors were able to reduce the total time required for running the experiments to under 5 days.

One other problem with running large numbers of experiment configurations is keeping an audit log of what experiments have been executed, on what kind of hardware and what metrics correspond to a specific experiment batch. Furthermore, large numbers of experiment configurations generate a large set of result files. When running the same experiment with multiple tools, the data processing step must align the tool outputs to make sure that the result files are in the same format and the correct values are compared.

C. Design Goals

In this section, we describe the design goals of the MDEO Scale DSL. Our DSL aims to satisfy the following requirements:

1) Offer a text-based DSL for specifying SBME experiments;
2) Require minimal user configuration and setup;
3) Provide an extensible framework that makes it easy to add new tools and additional hardware support for automated experiments execution;
4) Propose a common interface for collecting experiment metrics from SBME tools;
5) Offer support for automatically interpreting experiment metrics using common statistical metrics.

TABLE I
SUMMARY OF GENERATED SUMMARY STATISTICS FOR SINGLE AND MULTI-OBJECTIVE EXPERIMENTS.

Single Objective	**Multi-Objective**
Algorithm Steps	Algorithm Steps
Objective Mean	Hypervolume Mean
Objective Median	Hypervolume Median
Objective Minimum	Hypervolume Minimum
Objective Maximum	Hypervolume Maximum
Objective Standard Deviation	Hypervolume Standard Deviation
Objective Skewness	Hypervolume Skewness
Objective Kurtosis	Hypervolume Kurtosis
N/A	Reference Set Size
N/A	Reference Set Contributions
N/A	Ratio of Best Solutions

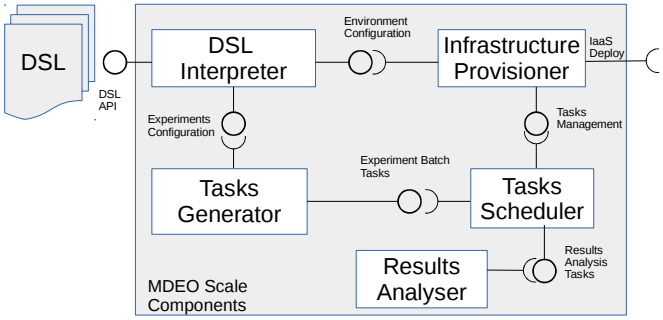

Fig. 1. Summary of the MDEOptimiser Scale DSL Architecture showing the main tool components

D. Architecture

This section presents the architecture for the MDEO Scale tool, a DSL that helps users configure and deploy SBME experiments on IaaS hardware.

In Fig. 1 we include an overview of the tool architecture. The user-facing DSL is implemented using XText. The tool contains five main components, which will be described in the following sections.

1) DSL Interpreter: The DSL interpreter components load the user-specified DSL, and checks the user inputs for consistency. The user is warned if any of the specified configuration files or their dependencies could not be found at the specified locations.

2) Tasks Generator: The task generator component creates task instances for each user configured task in the DSL. Each task is associated with the corresponding tool implementation, based on the user inputs.

3) Tasks Scheduler: The tasks scheduler component loads the generated tasks and sends them to the infrastructure provisioner for execution. This component also provides status information for the tasks that have been scheduled.

4) Infrastructure Provisioner: This component implements the IaaS provider API. It provides an interface to send the configured tasks to the job execution queue.

5) Results Analyser: The results analyser component is used in MDEO Scale to automatically calculate summary statistics and common metrics for the configured experiments. The evaluation of solution quality for single-objective op-

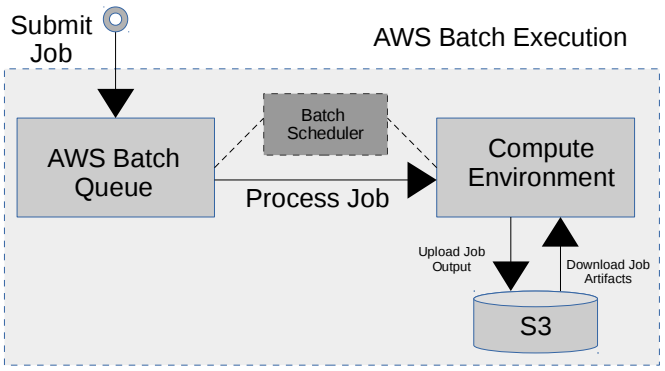

Fig. 2. Overview of AWS Batch Infrastructure.

timisation problems is straightforward. However, for multi-objective problems, solution quality evaluation is not a trivial problem and a number of solution quality indicators have been proposed to simplify this task [11].

MDEO Scale automatically determines if a problem solved in an experiment has one or multiple objectives. Depending on the number of objectives, the summary statistics included in Table I are calculated for each specified problem model instance. The calculated statistics have been implemented based on the author's past single and multi-objective search problems performance evaluation requirements [3]–[5]. The results evaluation framework can be extended to add further support for additional quality indicators and statistical evaluation calculations. For single-objective problems, the result summaries are calculated using the objective value. For multi-objective problems, the tool uses the hypervolume quality indicator [12].

For each problem instance model, the tool also performs statistical significance tests using the Mann-Whiney U test [13] and calculates Cohen's d effect size [14]. The tool generates a results analysis task for each user-defined experiment in the DSL. To ensure that the collected metrics are available before the results analysis starts, results analysis tasks are dependent on the successful completion of the main experiment tasks.

E. Infrastructure

In this section, we describe how MDEO Scale deploys the configured SBME tasks on the AWS infrastructure. We show how the tool uses additional services offered by AWS for storing metrics and experiment artifacts needed by the orchestrated jobs.

The *Infrastructure Provisioner* component described in the previous section offers an interface to the IaaS infrastructure provided by AWS Batch. In Fig. 2 we are including a high-level overview of the AWS Batch architecture. The service relies on a messaging queue, which accepts job requests that have to be executed. The queue is connected to a *Compute Environment* which defines the type of computing resources that can be used to process the queued tasks. Depending on the type of compute resources configured, the AWS Batch scheduler can execute the submitted jobs immediately, or it will wait until the required type of compute resources are available.

In the current version of the DSL, users configure the *Compute Environment* using the `environment` keyword in the `infrastructure` block of the DSL. The servers processing the tasks interact with the S3 storage system to fetch artifacts required for the `task` and to publish the output results. The output results are processed by the `tasks` scheduled by the *Results Analyser* component.

F. Data Collection Interface

In this section, we introduce the data collection interface used by MDEO Scale when running experiments with SBME tools. The current version of MDEO Scale focuses on running SBME experiments with the two tools introduced in Sect. II. Both MDEO and MOMoT are implemented using the MOEAFramework, which provides an interface for collecting metrics for analysis. We have used this interface and the proposed data format in our results analysis component. The data collection interface in MOEAFramework is described in the `org.moeaframework.analysis.collector` [4] package. To ensure compatibility with the results analyser in MDEO Scale, additional tools for which support is implemented, have to generate experiment results in the format specified by the data collection interface implemented in MOEAFramework.

V. Implementation

In this section, we give an overview of the DSL implemented in our tool, followed by details about the implementation and the technology stack used for the implementation. We start by describing the DSL using an example case study. We then give an overview of the tool framework and extension points that can be used to implement support for additional SBME tools and infrastructure providers. The implementation described in this section can be found on Github [5].

A. DSL

In Fig. 3 we include an example specification for the Class Responsibility Assignment [10] (CRA) case study that shows how to use our DSL to configure an experiment workflow that runs MDEO and MOMoT for the first two input models of the case study.

An MDEOptimiser Scale specification has two mandatory sections that have to be specified by the users. The first section, defined by the `infrastructure` keyword defines the IaaS configuration details. The mandatory configuration details are the type of infrastructure provider (currently, only AWS is supported), the name of the account that is configured on the machine running the DSL and a compute environment template [6], that defines the compute resource types, which will be used by the tool to orchestrate the experiment tasks.

[4] http://moeaframework.org/javadoc/org/moeaframework/analysis/collector/package-summary.html

[5] https://github.com/mde-optimiser/scale

[6] https://docs.aws.amazon.com/cli/latest/reference/batch/create-compute-environment.html

```
infrastructure "aws batch" {
    type aws
    account "default"
    environment "compute_environment.json"
}
experiment "CRA Case Study" {
    parameters {
        batches: 30
        artifacts: "src/java/resources/models/"
    }
    model "TTC_InputRDG_A" {
        task "MDEO" {
            run "cra_model_a.mopt"
            dependencies "./libraries/cra.jar"
        }
        task "MOMoT" {
            run "cra_model_a.momot"
            dependencies "./libraries/cra.jar"
        }
    }
    model "TTC_InputRDG_B" {
        task "MDEO" {
            run "cra_model_b.mopt"
            dependencies "./libraries/cra.jar"
        }
        task "MOMoT" {
            run "cra_model_b.momot"
            dependencies "./libraries/cra.jar"
        }
    }
}
```

Fig. 3. MDEOptimiser Scale DSL example showing an experiment configured for the CRA case study with two input models. Each input model is evaluated with MDEO and MoMOT.

The next section is defined by the `experiment` keyword. This block is used to define the experiment workflow, organised by the evaluated case study and the input models that have to be evaluated. The `experiment` keyword requires a name parameter to be specified. This value will be used as a unique identifier for the experiment, and the tool uses this value to tag the underlying components.

For each experiment the DSL allows users to specify a set of `parameters` that are applied to all child tasks. The two parameters supported are `batches`, to define the number of `task` repetitions, and the `artifacts` parameter to indicate the disk location which contains all the case study artifacts used by the task specifications loaded by the `run` keyword.

Each `experiment` block requires that at least one input model is specified using the `model` keyword. Similarly to the `experiment` keyword, `model` also requires the user to specify a name for the current model being evaluated.

Inside each `model` block, the DSL requires at least one `task` to be specified. A `task` defines a single configuration for an SBME tool, that can be used to evaluate a specific configuration for the input model of the case study, for which it has been specified. Currently, the DSL only supports the two tools described in Section II.

```
MDEO Scale usage instructions

java -jar scale.jar [options...] arguments...
 -projectPath VAL : Tool base path used to load artifacts defined in the DSL.
 -specPath VAL    : Specification file name to execute.

Example: java -jar scale.jar  -projectPath VAL -specPath VAL
```

Fig. 4. MDEOptimiser Scale standalone mode. This figure shows the printed help menu displayed when running the tool from the command line.

A complete specification must be written in a file with the `sc` extension. The user-defined `sc` files can be executed using the tool as an Eclipse plugin, from inside the Eclipse IDE or using the standalone mode from the command line. In Fig. 4 we include the tool standalone instructions. This feature is useful when running MDEO Scale outside of Eclipse IDE.

B. Extensibility

In this section we describe how the MDEO Scale can be extended to support additional tools.

To implement additional tools, users have to implement the `IScaleTask` interface and register the extension class in the `TaskFactory`. Tools are instantiated by the `TaskFactory` class by using the extension of the SBME specification file specified by the `run` keyword in the DSL. For each user configured task in the DSL, a new tool instance class is created with the user-provided parameters. `IScaleTask` implementations for new tasks have to provide a Docker container with a configured instance of the new tool using the MDEO Scale tool execution wrapper service implemented in the `ToolExecutor` class. The tool wrapper service ensures that the Docker container configured for each tool can interpret the DSL task specification messages received from the AWS Batch queue. The container must be published on the Docker Hub. Container images stored on private repositories are currently not supported.

The current version of the tool relies extensively on the AWS Batch service. The tool can be extended to support additional IaaS providers by implementing additional providers in the infrastructure provisioner component.

VI. Example Run

In this section, we describe a demonstrative run of MDEO Scale using the DSL configuration described in 3. This example run uses the case study introduced in III specified using MDEO and MoMOT. The case study consists of five input models, however, because of space limitations, for this demonstration, only input models A and B will be used. A complete evaluation of this case study using the two tools used in this configuration can be found in [4].

Running the tool using the specification in Fig. 3, MDEO Scale generates for 120 AWS Batch jobs for the specified tasks. An individual job is generated for each batch execution of the configured tasks. Following the scheduling of the task messages on the queue, the AWS Batch Scheduler determines and automatically initialises the optimal number of servers that have to be initialised for processing the jobs

from the queue. The processing can begin immediately or with a delay, depending on the configuration specified using the `infrastructure` block in the DSL. Once the processing servers become available, the job processing begins in parallel, until the queue is emptied. At the end of each batch, the generated results files are saved and uploaded to AWS S3 for processing by the Results Analyser.

The Results Analyser is executed after all the AWS Batch queue jobs have been processed successfully and the results artifacts are available for processing in AWS S3. The generated results files for each processed task are loaded by the tool from AWS S3 and the generated processed to calculate the metrics described in Table I. For each configured `model` the Results Analyser generates a plot showing the evolution of the objective value for single-objective problems and hypervolume for multi-objective problems.

VII. Related work

Recently there have been a number of tools developed with the goal of helping researchers to configure and run experiments, in a way that shifts the focus from building orchestration software and makes the process reproducible [15]. While most of these systems are intended for data intensive fields, such as life sciences and astronomy, to the best of our knowledge, there is no such tool that provides support for SMBE tools.

A. Cloud Deployment

The Open Cloud Computing Interface (OCCI) is a specification that proposes a common standard for interacting with computational resources offered by IaaS, PaaS providers [16]. The OCCI has been proposed by the *Open Grid Forum* and consists of an API specification that seeks to standardise how users interact with APIs provided by different XaaS providers. An alternative specification standard is The Topology and Orchestration Specification for Cloud Applications (TOSCA) [17].

In [18] Erbel et al. propose an approach that can schedule workflow architectures to be executed on cloud infrastructure using OCCI. This approach aims to run on OCCI compliant clouds and to maximise the resource usage for the provisioned computation nodes, by running a simulation to evaluate performance metrics before a deployment is made.

A similar approach for reproducing scientific workflows on the cloud, using TOSCA, is proposed in Qasha et al. in [19]. The authors propose a framework that aims to use Github and DockerHub [7] as repositories for storing workflows and tasks. This framework uses Cloudify [8] as a TOSCA compliant workflow engine.

Our approach is similar to the two frameworks described in this section. The main difference is that MDEOptimiser Scale is aimed at the SBME community and offers a number of additional tools, such as metrics collection and results summary generation. Using one of the two frameworks above, users still have to solve the problem of capturing experiment data, and interpreting it.

[7]https://www.docker.com/products/docker-hub

[8]https://cloudify.co/

B. Batch Processing

Batch processing tools are applications that can process computationally-intensive tasks sequentially or in parallel, on a single server or on a cluster of servers.

Apache Mesos is a distributed task scheduling framework that allows tasks to be executed across a server cluster [20]. The goal of Mesos is to offer good resource utilisation, while scaling to thousands of nodes in a server cluster, while supporting a large number of tasks to be scheduled and executed. The tasks are isolated using container technologies, such as Docker or a built-in containerisation implementation.

An alternative task scheduling framework is Bistro [21]. Bistro is a distributed task scheduler, that runs data-intensive tasks on live production systems that have unused computation capacity. The goal of this scheduler is to process data at runtime, without having to create an immutable backup. The tool uses a task scheduling algorithm to assign tasks to workers, and can process the tasks without affecting the performance of the customer-facing live applications.

Luigi is a batch jobs pipelines orchestrator implemented using Python [22]. The tool offers features to visualise the status of currently running batch jobs, handle failures, and manage workflows. Tasks are executed using a scheduler, which can be either local, for jobs executed locally, or central for jobs distributed across a number of job executing workers. This tool is not as complex as Mesos or Bistro, however, it is a good example of a simple task scheduler tool that can be used to manage simple workflows.

Our DSL can be extended to support as backend batch processing technologies any of the tools presented in this section. The disadvantage that comes with using Mesos or Bistro as a backend is that there is an initial overhead required to configure these tools so that they can be used. This option is viable when there is an existing server cluster available, which do not have a resource orchestration framework deployed. Our approach uses IaaS hardware, provided by AWS and eliminates the requirement to own and maintain a cluster.

C. Experiment Workflows

Experiment workflow systems are computational tools that enable researchers who perform data-intensive experiments to focus on their work instead of building data processing pipelines. These tools are focused on data-intensive research areas such as bioinformatics, chemistry or physics. A curated list of existing workflow systems can be consulted in [23].

Common Workflow Language (CWL) is a workflow specification standard [24]. CWL can be used to build portable analysis workflows. CWL based workflows consist of orchestrated command line tools, which can run on platforms that implement the standard. Another computational pipeline workflow DSL is Nextflow [25]. The tool allows users to use Linux executable scripting languages to define processes which can

be orchestrated to form a pipeline. Netflow supports by default Google Computing Cloud and Amazon Web Services.

The DSL we introduced in this paper can be extended to generate CWL or NextFlow workflow instances. This will allow us to extend the number of deployment platforms supported while focusing on the needs for the SBME practitioners who use our DSL and using a more mature workflow engine as the backend for our workflow architecture management.

VIII. Conclusion and Future Work

In this paper, we have introduced a prototype of the MDEO Scale DSL, a tool that enables researchers running experiments using SBME tools to easily create an experiment workflow that can be executed on IaaS hardware. The tool offers an accessible approach to sharing SBME experiment workflows, to make it easy to publish reproducible research artifacts. We described our motivations for building the tool and gave an overview of the architecture and the approach used to orchestrate experiments on Iaas hardware.

The tool presented in this paper is a prototype in the early stages of development. In the future we are planning to extend the tool in a number of directions. The following is a list of ideas currently being evaluated:

- Extend support for more than one IaaS provider;
- Improve the extensibility of the DSL by allowing users to add support for additional tools by simply specifying in the DSL a class implementing an interface for supporting additional tools;
- Allow users to specify custom metrics that can be calculated on the generated metrics;
- Offer the ability to easily load case studies from a pool supported by the tool to make it easier to evaluate SBME research ideas;
- Evaluate experiment reproducibility threats and mitigation strategies when using cloud hardware;
- Extend the DSL to support the execution of tools and problems from the wider field of SBSE;
- Extend the algorithm instrumentation functionality to simplify integration with optimisation frameworks other than MOEAFramework.

Acknowledgment

This work has been supported by the *Engineering and Physical Sciences Research Council* grant number 1805606.

References

[1] I. Boussaïd, P. Siarry, and M. Ahmed-Nacer, "A survey on search-based model-driven engineering," *Automated Software Engineering*, vol. 24, no. 2, pp. 233–294, 2017.

[2] D. Efstathiou, J. R. Williams, and S. Zschaler, "Crepe complete: Multi-objective optimisation for your models," in *Proc. 1st Int'l Workshop on Combining Modelling with Search- and Example-Based Approaches (CMSEBA'14)*, 2014.

[3] J. Murphy, A. Burdusel, L. Michael, S. Zschaler, and E. Black, "Deriving persuasion strategies using search-based model engineering," in *Computational Models of Argument: Proceedings of COMMA 2018*, vol. 305, pp. 221–232, IOS Press, 2018.

[4] S. John, A. Burdusel, R. Bill, D. Struber, G. Taentzer, S. Zschaler, and M. Wimmer, "Searching for optimal models: Comparing two encoding approaches," in *Theory and Practice of Model Transformation: 12th International Conference, Accepted*, ICMT '19, 2019.

[5] A. Burdusel, S. Zschaler, and S. John, "Automatic generation of atomic consistency preserving search operators for search-based model engineering," in *IEEE / ACM 22nd International Conference on Model Driven Engineering Languages and Systems*, MODELS, 2019.

[6] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.

[7] P. Fazenda, J. McDermott, and U.-M. O'Reilly, "A library to run evolutionary algorithms in the cloud using mapreduce," in *European Conference on the Applications of Evolutionary Computation*, pp. 416–425, 2012.

[8] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–11:61, 2012.

[9] R. Bill, M. Fleck, J. Troya, T. Mayerhofer, and M. Wimmer, "A local and global tour on MOMoT," *Software & Systems Modeling*, pp. 1–30, 2017.

[10] M. Fleck, J. Troya, and M. Wimmer, "The class responsibility assignment case," in *Proceedings of the 9th Transformation Tool Contest* (A. Garcia-Dominguez, F. Krikava, and L. M. Rose, eds.), vol. 1758, pp. 1–8, CEUR, 2016.

[11] M. Li and X. Yao, "Quality evaluation of solution sets in multiobjective optimisation: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, p. 26, 2019.

[12] E. Zitzler, D. Brockhoff, and L. Thiele, "The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration," in *Evolutionary multi-criterion optimization*, pp. 862–876, 2007.

[13] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Ann. Math. Statist.*, vol. 18, pp. 50–60, 03 1947.

[14] J. Cohen, "Statistical power analysis for the behaviors science.(2nd)," *New Jersey: Laurence Erlbaum Associates, Publishers, Hillsdale*, 1988.

[15] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *Journal of Grid Computing*, vol. 13, pp. 457–493, Dec 2015.

[16] A. Edmonds, T. Metsch, A. Papaspyrou, and A. Richardson, "Toward an open cloud standard," *IEEE Internet Computing*, vol. 16, no. 4, pp. 15–25, 2012.

[17] "OASIS topology and orchestration specification for cloud applications." https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca. Accessed: 2019-07-04.

[18] J. Erbel, F. Korte, and J. Grabowski, "Scheduling architectures for scientific workflows in the cloud," in *System Analysis and Modeling. Languages, Methods, and Tools for Systems Engineering*, pp. 20–28, 2018.

[19] R. Qasha, J. Cała, and P. Watson, "A framework for scientific workflow reproducibility in the cloud," in *2016 IEEE 12th International Conference on e-Science (e-Science)*, pp. 81–90, IEEE, 2016.

[20] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center.," in *NSDI*, vol. 11, pp. 22–22, 2011.

[21] A. Goder, A. Spiridonov, and Y. Wang, "Bistro: Scheduling data-parallel jobs against live production systems," in *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, pp. 459–471, 2015.

[22] "Luigi." https://github.com/spotify/luigi. Accessed: 2019-07-04.

[23] "Computational data analysis workflow systems." https://s.apache.org/existing-workflow-systems. Accessed: 2019-07-04.

[24] P. Amstutz, M. R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, M. Scales, S. Soiland-Reyes, and L. Stojanovic, "Common Workflow Language, v1.0," 7 2016.

[25] "NextFlow a dsl for parallel and scalable computational pipeline." https://www.nextflow.io/. Accessed: 2019-07-04.