A Language-centered Approach for
Transparent Experimentation Workflows

Frank Kühnlenz
Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
kuehnlenz@informatik.hu-berlin.de

Abstract A common approach to handle computer-based model experimentation is to use scientific workflow management systems. Such systems are dataflow-oriented. To match this paradigm, the experimenter has to “translate” the objectives of his experimentation into terms of tasks with their composition and in- and output data. This challenge causes a significant effort, requires workflow expert knowledge and makes design and comprehension of such workflows problematic for the target audience who are usually not computer scientists.

Hence, experimenters often do not use any adequate tool-support that would help them to describe and execute their experimentation processes completely. We argue, experimenters need new, more adequate means of expressions to handle their experimentation processes. Thus, we introduce our new language-centered approach that combines common workflow concepts, experimentation process characteristics and specifics of the domain where experimentation takes places. To demonstrate the practicability, we developed a prototype of such an experimentation workflow system.

Keywords: model experimentation, experiment management, scientific workflows, domain-specific language, modeling and simulation, ExpL

1 Introduction

In this paper experimentation means performing computer-based experiments on structural and behavioural system models, which are executed in special computing environments (e.g., a simulation framework) to get model observations. Systematic experimentation by varying the model input parameters leads to lots of possibly costly experiments that must be planned, executed, monitored, evaluated and documented, to ensure comprehension, reproducibility and re-usability (summarized as transparency).

To the authors knowledge, there exist currently two approaches to tackle this problem. One is to use simulation frameworks that were designed for simulation modeling (e.g., JiST/ SWANS\textsuperscript{1} for computer networks, DESMO-J\textsuperscript{1} for general purpose or ODEMx [4] for both); rarely with extensions for experiment

\footnote{http://jist.ece.cornell.edu http://desmoj.sourceforge.net}
management, e.g., DISMO [6] for DESMO-J. But, specifying systematic experimentation with varying the model input parameters is usually not supported. Additionally, there is no clear separation between the experiment description and the simulation model, which are both described with framework-dependent language concepts. Hence, comprehension, re-using or sharing experimental work with other scientists is limited by the particular framework. That is a problem, if certain experimentation goals make it necessary to switch the particular framework. In that case, simulation model and experimentation description must be re-implemented.

The second approach is using scientific workflow management systems (S-WfMS) (e.g., Taverna [14] or Kepler [13]), which are applicable, since the process of experimentation can be defined as a specifically structured scientific workflow. S-WfMS are dataflow-oriented. To match this paradigm, the experimenter has to "translate" the objectives of his experimentation into terms of resources and tasks with their composition and in- and output data. This challenge causes a significant effort, requires workflow expert knowledge and makes design and comprehension of such workflows problematic for the target audience who are usually not computer scientists. Hence, the experimenter has no proper means of expressions, e.g., to express systematic parameter variations.

The rest of this paper is organized as follows: section 2 gives a brief overview about workflow terminology and experimentation workflows. Section 3 presents our new approach of using domain-specific languages in combination of metamodel-based technologies to improve model experimentation. This is followed by an example in section 3.1 and finally the conclusions and future work in 4.

2 Experimentation Workflows

The Workflow Management Coalition (WFMC) concentrates on business workflows, but the basic terms [1] are independent from the business objective. Firstly, a (business) process is what is intended to happen. It is defined in a process definition that is a representation (or abstraction) of what is intended to happen (and may contain sub-process definitions). A process definition is composed of activities (also referred to as tasks) that can be managed as part of a workflow system (automated activity) or not (manual activity). A process definition that is only composed of automated activities is called workflow definition\(^2\). A WfMS controls automated aspects of the process: it is used to create and manage workflow instances on base of their corresponding workflow definitions.

A scientific workflow (S-Wf) is a formal definition of a process for accomplishing a scientific objective [12]. S-WFs have specific characteristics like data- and resource orientation [3]. Additionally, their workflow life cycle (actions needed to set up and run workflows) distinguishes from its business counterpart. Especially, the focus of scientists’ work is usually on single instances and hence, they

\(^2\) Often, the word definition is omitted and it is just called workflow or workflow model.
do not distinguish between workflow models and instances or are not aware of a
difference between models and instances, respectively [7].

We introduce an *experimentation workflow* (Exp-Wf) as a specifically struc-
tured S-Wf that comprises the ordered phases planning, execution and evalua-
tion. This structure represents also the Exp-Wf life cycle. The basic idea of an
Exp-Wf is to concentrate on the central artifact: the model on which the experi-
mentation is applied to – we call this *model under experimentation* (MUX). The
MUX has to satisfy only one of the most general characteristic of a behavioural
model: applying (the same) input parameter values results in (the same) output
values. Hence, it is independent from the used formalism (e.g., DEVS, DESS).

Therefore, typical steps in the planning phase include describing the input
and output ports of the MUX, how to get and systematically vary the input
parameter values, what should be observed and which activities are necessary in
order to evaluate the observations. Zeigler et al. [18] names this information an
*experimental frame* that originates in the objectives of the experiments.

The Exp-Wf definition as used in the execution phase contains all aspects
of how to actually execute the MUX by a certain *execution environment*. In
this sense, the term *execution environment* means the same, what Zeigler et al.
[18, p. 30] names as "a simulator [that] is any computer system [...] capable of
executing a model to generate its behavior".

We prefer the more general term *execution environment*, because it is not
intuitively associated with a simulation framework context; an execution envi-
ronment may reference, e.g., a simulation framework or a testbed. Hence, from
the Exp-Wf definition point of view, it is exchangeable, because of a clearly sep-
aration from the descriptions of the MUX-ports and the parameter variations.

Switching the execution environment is especially useful for a MUX that is
specified platform independently (following the approach of *model-driven engi-
neering*, MDE). Platform dependent MUX can only be executed in one partic-
ular execution environment – supporting several execution environments results
in re-implementing for each a suitable, equivalent MUX.

In the case of a platform independent MUX, the workflow definition includes
activities that perform transformations in order to be able to execute it on
the desired target execution environment. E.g., such transformations can be
code generation and compilation. Hence, an Exp-Wf definition must be able to
handle such variations in different MUX-versions as well as their corresponding
execution environments (see Figure 1).

Activities in the evaluation phase process the observations of the experiments.
E.g., for each experiment that produces a set of observations this could be the
computation of a fitness function, a specific metric or aggregation. Based on this,
a comparison of experiments can be made and presented, e.g., as textual reports
or diagrams. It could also be possible to use a fitness function as base for genetic
algorithms. In this case, a new set of parameter variations could be derived.

\[3\] For further information see [17] for a comparison between scientific and business
workflows and [3] for an overview of workflow system features and capabilities.
3 Experimentation Language (ExpL)

Our approach allows modeling of Exp-Wfs on a framework-independent, experimentation domain-specific level with the general philosophy of separation of concerns: descriptions of artifacts and experiments in design, execution and evaluation. Therefore we identified general experimentation workflow concepts and formalized them as a meta-model-based domain-specific language (DSL) that we name experimentation language (ExpL). The approach is based on the idea to apply the paradigm of model-driven engineering (MDE) to the domain of Exp-Wfs. This closes the gap between the emerging model-driven, framework-independent development of the MUX and currently existing, framework-dependent approaches for the Exp-Wf [5]. ExpL provides common workflow concepts like activities and resources, enhanced by declarative elements (example in listing 1.1). This combination is a novelty that helps to provide a proper level of abstraction to the experimenter. However, BioFlow [9] as a web-based workflow language for life sciences uses also declarative language elements. But, BioFlow as a S-WfMS has the same limitations as mentioned and is not applicable for Exp-Wfs in the desired sense. Additionally, its technology is less flexible than ExpL’s meta-model approach.

Each experimentation domain (e.g., computer networks) has certain specifics in configuration, execution and evaluation. Addressing this adequately, we propose to separate the concerns and use ExpL with three complementary DSLs, which are exchangeable in order to be adaptive for the experimentation domains (Figure 2). Theisselmann et al. introduced this principle of combining different DSLs on base of the same meta-meta-model as a language-centered approach (LCA) [15]. A configuration DSL formalizes domain-specific concepts, e.g., for structured data types, operators and units. An execution DSL mainly describes

---

4 Such elements define the logic (the desired goal or result) and not the control (how we achieve the desired goal), which is a characteristic of declarative languages. However, ExpL is not a declarative language at all, since it defines control flow by workflow concepts like task composition.
the relation between the MUX and the execution environment. An *evaluation DSL* has to provide means to formulate metric functions and the application of optimization methods. The LCA is essentially more powerful in adaptation to a new domain than a S-WfMS-based approach. Adapting a S-WfMS basically means to implement domain-specific activities, for which a high-level description of domain-specific, structured data types are not supported generally. In contrast, LCA as a meta-model-based approach provides concepts of inheritance and relations. Hence, existing activities can be re-used and extended with preserving all existing relations to other activities.

**Figure 2.** LCA: adaptable combining Exp-Wf concepts (ExpL DSL) with specifics in the particular experimentation domain.

## 3.1 Example

A simplified, but realistic example illustrates the usage of the implemented prototype of ExpL [10] during planning and execution phases (due to space limitations the evaluation phase is omitted). In this example, the goal of the experimentation is to evaluate the performance of a new routing protocol in a wireless mesh network.

It is assumed that an adequate model (the MUX) for this purpose already exists, which can be parametrized with a network topology (describing spatial positions of nodes and links between them), a radio model and a bit rate (used by the radio model). For evaluating the performance of the routing protocol, the gained throughput of a flow between the most distant nodes in the network is measured. Hence, the output of the model (model observations) is a time series of tuples in the form of `modeltime` and the corresponding throughput. It is furthermore assumed that an executable simulator binary exists, which is generated from the MUX using a suitable simulation framework. Usually, this would be represented by activities in the Exp-Wf invoking a compiler toolchain.

**Planning** Listing 1.1 shows the definition of an `ExperimentPlan` using the textual syntax of ExpL. It was written using a *smart editor* supporting syntax highlighting and code completion. Due to the meta-model-based technology of ExpL, such editors are generated automatically.
An ExperimentPlan is a declarative language element of ExpL and specifies parameter variations (that are defining the coverage of the complete parameter space) and how the simulator binary has to be executed. In the example (listing 1.1), we have three kinds of parameter variations: a stepwise variation from 1 to 10 using an equidistant stepwidth (#1 in line 3), an iteration over a set of values (line 6) and an iteration over the output of another experiment plan SynthesizeTopology that defines, how to synthesize several network topologies (omitted, referenced in line 9). Each reference (after keyword vary) points to one typed input port declaration of the MUX MyRoutingProtocol.java (line 1).

Each of these variation definitions produces a set of values. One element of the cartesian product of these sets is actually a tuple of input parameter values for one experiment. So far, experiments defined by an ExperimentPlan are considered independently by each other and can be executed in parallel (line 2). Often, it is not practicable (or desired) to compute and evaluate all experiments that would result from the cartesian product. Hence, ExpL provides means of expressions to subset the cartesian product using the language element ConstraintModel (line 12). It consists of constraints (or expressions) over input parameter declarations. In the example this could be used to specify...
a dependency of that kind, that a certain radio model can only handle a certain maximum bit rate.

At last, the ExecutionEnvironment (line 14) adds the specification, how the input parameters that where specified by the experiment plan have to be passed to the simulator binary and on what machine it has to be executed.

Execution Figure 3 illustrates the steps, which the ExpL-WfMS performs in order to execute an Exp-Wf that is formalized as ExpL-Wf model.

Figure 3. Steps necessary to execute an ExpL-Wf model using the ExpL-WfMS.

Automated model-to-model transformation (a) in Figure 3 generates from the ExpL workflow model an executable workflow. This transformation includes, e.g., to compute the input parameter values for the MUX execution. The executable workflow is visualized in Figure 4. Generating such an executable workflow enhances transparency and comprehension because the experimenter can inspect what is intended to be executed before the possibly long running experiments are actually executed. Automated transformation (b) in Figure 3 is a model-to-text transformation to generate the runtime representation of the executable workflow in a format that is suitable to be executed by an existing workflow system. Both transformations are described on base of the ExpL-meta-model.

Figure 4. Visualization of the executable example-workflow. Its right part is the result of an automated transformation (following a notation as used in UML activity diagrams).
4 Conclusions and Future Work

We presented a new language-centered approach for designing and management of Exp-Wfs that combines common workflow concepts and experimentation domain-specific means of expressions on an experimentation goal-oriented and framework-independent level. It consists of ExpL in conjunction with a Configuration and Evaluation DSL using meta-model-based technology. Tailored means of expressions in the experimenter’s vocabulary for the specific domain make experimentation easier to handle as compared to S-WfMS, especially for non-computer scientists. In addition with a consistent documentation and clear provenance of all involved artifacts this leads to an improved understanding of the experiments. In consequence, the experimentally gained knowledge and the way, of how it was achieved, can be shared with other scientists more transparently.

A first case study using ExpL in geography, is modeling experiments with the land use change model SLEUTH [11]. To demonstrate the power of LCA, a simple Evaluation DSL realizing GIS-based spatial operations was coupled [15]. A second, ongoing case study in physics is modeling experiments for developing optical nanostructures [16].

As we argued, the flexibility of Exp-Wf definitions in principle allows to switch easily from one model execution environment to another, without loosing the experimentation description. E.g., this is highly desired for experiments with computer networks at which the number of nodes is often a parameter of big interests. Due to high financial costs, even large testbeds (like DES [8] or HWL [2]) comprise in the best case hundreds of wireless, meshed nodes. Trade-offs to increase the number of nodes are virtualization (thousands of nodes) and simulations (ten thousands of nodes), but to the cost of less effects that can be observed in comparison to the real world system. On the other hand, within simulations usually abstractions are made to reduce the complexity and to focus on one particular experimentation goal or question. Hence, it is desired to combine the knowledge that can be achieved by this different methods. To demonstrate this by the help of ExpL is a challenging future work.

Acknowledgements

This work is supported by grants from the Deutsche Forschungsgemeinschaft (DFG), Research Training Group METRIK (GRK 1324).

References