Supporting Simulation Experiments with Megamodeling

Sema Çam[1,2], Orçun Dayıbaş[2], Bilge K. Görür[3], Halit Oğuztüzün[2], Levent Yilmaz[4], Sritika Chakladar[4], Kyle Doud[4], Alice E. Smith[5] and Alejandro Teran-Somohano[5]

[1]*Department of Command Control and Combat Systems, HAVELSAN A.Ş., Ankara, Turkey*

[2]*Department of Computer Engineering, Middle East Technical University, Ankara, Turkey*

[3]*Department of Computer Engineering, Hacettepe University, Ankara, Turkey*

[4]*Department of Computer Science and Software Engineering, Auburn University, Alabama, U.S.A.*

[5]*Department of Industrial and Systems Engineering, Auburn University, Alabama, U.S.A.*

Keywords: Model-Driven Engineering, Global Model Management, Megamodel.

Abstract: Recent developments in computational science and engineering allow a great deal of experimental work to be conducted through computer simulation. In a simulation experiment, a model of the phenomena to be studied is run in a computing environment under varying model and environment settings. As models are adjusted to experimental procedures and execution environments, variations arise. Models also evolve in time. Thus, models must be managed. We propose to bring Global Model Management (GMM) to bear on simulation experiment management by using techniques and tools from megamodeling. The proposed approach will facilitate model management tasks by providing an interface to query the model repository, relate models with each other, and apply model transformations from/to simulation models. Our proposed Megamodel for Simulation Experiments is based on SED-ML (Simulation Experiment Description Markup Language).

1 INTRODUCTION

In many science and engineering problems, theories and hypotheses about what makes a system work, or to explain some phenomena in terms of cause and effect relationships, are put forth. Then, experiments are conducted to test the hypotheses. Values of the input variables of the system are changed intentionally, and the resulting output values are observed and measured. Experiments produce evidence whether proposed theories are supported or not (Montgomery, 2006). With recent developments in computational science and engineering, modeling and simulation technologies, experiments are now performed on computers to avoid the risk, or even the practical impossibility, of conducting experiments in the real world. Furthermore, simulation experiments usually require less time, cost and effort. Such experiments are known as *in silico* experiments. Therefore, the use of simulation experiments is common among the experimental scientists and engineers. There is a growing number of simulation experiment projects such as myExperiment (Goble et al., 2010) and Experiment (Denny Luan, 2017) projects, which are essentially social web sites for researchers sharing scientific workflows. These projects involve the development of various kinds of simulation experiment models. However, these models need a supporting environment that is easy to use by non-programmers to be sustainable and manageable. The experiment models in the environment should be accessible and manipulable (e.g. loading/saving/editing/deleting/searching/executing models). In that respect, Global Model Management (GMM) is a suitable concept for this problem. GMM aims to manage a large and varied set of artifacts produced in *modeling-in-the-large* (globally dealing with models, metamodels and their properties and relations (Bézivin et al., 2005) efforts in projects that adopt Model Driven Engineering (MDE).

The main objective of this paper is to apply the GMM concept for simulation experiments to provide an environment for scientists, who deal with standalone simulations, simulation data (configuration, parameter, and input and output data), and multiple simulation models over time. Although a single model

Çam, S., Dayıbaş, O., Görür, B., Oğuztüzün, H., Yilmaz, L., Chakladar, S., Doud, K., Smith, A. and Teran-Somohano, A.
Supporting Simulation Experiments with Megamodeling.
DOI: 10.5220/0006586703720378
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 372-378

might be relatively easy to manage, in real-life situations experimenters have to deal with large sets of interacting simulation models and associated data which are difficult to maintain manually. Management of the evolution of models and related artifacts requires configuration management support, especially because, developers need to search and reuse models from previous projects. Also, collaboration among multiple developers require cooperative management of models (Koegel and Helming, 2010).

Models keep evolving along with the scientific endeavor. Their sophistication and variety tend to increase over time. When a large number of models are involved, users are faced with formidable system management issues, including the need to maintain consistency. New types of relations between models can be configured during model evolution. Additionally, the system should be extensible so that it can be readily adjusted to new domains. Furthermore, simulation experimentation brings about logical complexity at multiple levels such as domain modeling (conceptual modeling) issues, model complexity, simulation complexity (method for generating behavior from a model), complexity of operational environments for simulation execution, heterogeneity of the experiment design, complexity of the management of simulation input and output data and other scenario-related data, and challenges about presentation of results (including visualization).

To alleviate the above mentioned problems, we propose to carry Simulation Experiments into an MDE environment and provide a user interface for Simulation Experiment models that enables the query, relating and applying model transformation from/to available models. We contend that a megamodel will be helpful to manage the models involved in a family of simulation experiments. More specifically, in this article we report on ongoing work about how to support simulation experiments through megamodeling techniques. Our work takes advantage of the Xperimenter. Xperimenter is a domain specific language (DSL) that aims to provide a declarative medium for experiment specifications. We also present a case study that aims to query Xperimenter models. The case study is realized by using Xtend (Efftinge, 2017), a programming language that is a flexible and expressive dialect of Java, particularly effective for dealing with model transformations.

The rest of this paper is organized as follows: in Section 2, some related works about GMM and megamodeling are discussed. Then, Xperimenter DSL for Simulation Experiment overview is given in Section 3 for background. In Section 4, our megamodel is introduced. Further, we have a small case study, introduced in Section 5, for querying Xperimenter models. Finally, conclusions and future work directions are given in Section 6.

2 RELATED WORK

Global Model Management (GMM) aims to handle models and metamodels, and their properties and relations, in a model-engineering environment. It is a sophisticated way of creating, storing, viewing, accessing, modifying, and using the information associated with all these modeling elements. As for a megamodel, it is a model that contains models and relations between them. The megamodel represents the Model Driven Engineering artifacts, including the transformation composition and execution within a model. Basically, GMM provides a framework for managing the large sets of heterogeneous and complex MDE artifacts and a megamodel is a model defined in GMM that contains MDE artifacts.

The proposed approaches for GMM are summarized in (Hebig et al., 2011). In (Jouault et al., 2010), megamodels are combined with model weaving and proposed as a new infrastructure for GMM. However, the proposed technique does not support automated production of traceability links for model navigation nor does it provide model synchronization. As for identification of the model relationships, the GEMOC initiative (Combemale et al., 2014) defines three relations: interoperability, collaboration, and composition. Interoperability provides information exchange among the models. Collaboration supports coupling between models and coupled models affect each others development. Finally, composition enables one to combine information from different models to create a new one.

As for megamodel, it is a model that contains models and relations between them. The megamodel represents the Model Driven Engineering artifacts, including transformation composition and execution within a model. Megamodeling is applied for several practical purposes. In (Favre and NGuyen, 2005), it is applied for modeling software evolution through transformations. In (Fritzsche et al., 2009), the model driven concept is utilized for a non-functional property and megamodeling is applied to Model Driven Performance Engineering. Furthermore, megamodeling is reportedly applied in such diverse areas such as data analysis (Ceri et al., 2013), consistency checking of industrial product lines (Vierhauser et al., 2012) and an e-government project (Büttner et al., 2014). In particular, Simmonds and coworkers (Simmonds et al., 2015) undertook a megamodel study for Soft-

ware Process Line modeling and evolution. They assert that megamodeling facilitates achieving a uniform mechanism for process definition, variability, tailoring and evolution.

In our work, we aim to build a megamodel that facilitates scientific experiments with simulation models. The Xperimenter DSL is used to specify and execute simulation models. Unlike prior work, our study aims to contribute to the management of simulation models, the abstraction of simulation experiments at design and execution levels, and the management of simulation artifacts. We intend to specify and run simulation experiments and analyze the results with the support of megamodeling techniques. Additionally, we propose to create an extendable environment that adapts existing simulation models developed with different technologies.

3 BACKGROUND

3.1 Xperimenter for Simulation Experiment

Xperimenter is a DSL for simulation experiment design and execution. The DSL has three main objectives. The first one is to provide a medium for specifying simulation experiments. The second is to manage simulation experiment variability by mapping fragments of an experiment specification to higher level abstractions, namely, features. The third is running a simulation experiment on a target platform, such as a scientific workflow management system. A simplified metamodel definition of Xperimenter is given in Figure 1. The elements of the metamodel are briefly described below to provide a conceptual framework for the components of a simulation experiment as well as the relationships among them.

Experiment: The attributes of this class include identifying information related to an experiment, such as the experiment name, date and description. An experiment is composed of following main components: Simulation model, objective, simulation runs , design, design matrix, statistical analysis and visualization methods.

SimulationModel: It is the core aspect of the simulation experiment. On the other hand, in simulation experiments, the simulation model is the primary source of information.

Objective: The class defines the purpose of the experiment. This definition influences the experiment type and the number of runs that are required to achieve the experiment's goals.

Run: The number of simulation runs required is not necessarily known at the time of experiment design; it may depend on the actual progress of the experiment. Each run has a start and end time.

Design and DesignMatrix: Design class captures the structural aspect of the experiment. The experimental structure is defined by the responses, the factors and their levels, and user-provided value ranges. Based on this design, a design matrix can be created, which specifies the actual experimental runs.

StatAnalysis: Statistical analyses on the experimental data can provide a wealth of useful information about the influence of factors on the responses. Xperimenter enables the use of ANOVA analysis, hypothesis testing, and confidence intervals in its present version.

Visualization: This part of the model denotes the method of visual representation of the analysis.

Variable (Response and Factor): Each variable is identified by its name and type. Currently, four types of variables (Integer, Float, Boolean, String) are supported in the metamodel. These variables can be divided into two distinct classes, namely, responses and factors. Responses, which correspond to dependent variables, represent the output values of the experiment. An experiment is conducted by varying these factor values and recording the outcomes. Each factor can have multiple levels (also called treatments). There is a one-to-one mapping between a factor level and a factor value. The factor values are the values that are actually fed into the simulation model. Response values are generated at each experimental run. An experimental run is a run of the simulation model with a set of input parameters. The input parameters are the factor values for that run.

SamplingInstance: A sampling instance is basically an aggregation of a variable and its actual value. It can be an input to the model (factor variable) or output of it (response variable). Sampling instances can be used in a design matrix.

4 MEGAMODEL FOR SIMULATION EXPERIMENTS

4.1 Conceptual Overview

The groundwork involves building a metamodel for simulation experiment megamodels so that conforming megamodels can be defined by users for their experiments. We call the megamodel GMM4SE, short for Global Model Management for Simulation Experiments. GMM4SE defines the supported types of

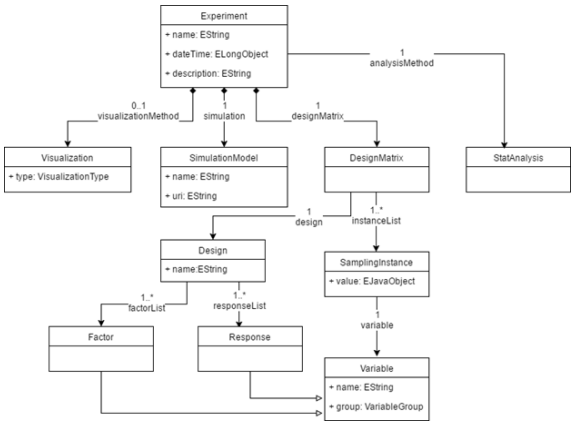

Figure 1: Simulation Experiment Domain Model.

modeling artifacts; presently these are metamodels, transformations, and the relationships between metamodels and model transformations. Figure 3 depicts our simplified megamodel definition. The workspace shown in the upper part of Figure 2 refers to the model workspace. It contains a model-based solution comprised of three modeling artifacts: (i) a metamodel for Xperimenter, which was introduced in Section 2, (ii) Xperimenter models, and (iii) a model-to-model transformation that produces models conforming to the Xperimenter metamodel from models conforming to the same metamodel. All of the modeling artifacts in the workspace represent data for a particular megamodel. The megamodel keeps information about their locations in the directory and the relationships among them (e.g., dependency between the source and target models of model transformation). The megamodel conforms to the GMM4SE metamodel.

There are two fundamental mechanisms to specify model relations in our megamodel. The first mechanism is used to define weaving relationships among the models, and the second mechanism is used to define constraints between them. Model weaving is an operation for defining fine-grained relationships between models and metamodels and produces a weaving model from the relationship. Also, operations are executed on them based on the semantics of the weaving relations (Bézivin, 2005). For instance, when a weaving relation occurs, a weaving model is produced and it represents a mapping between the related models.

As for the constraints, they are provided by defining rules among the models. However, rules specific to simulation experiments are currently missing in our megamodel. We are planning to add rule definition functionality to the megamodel by using Xtend. We will define a number of constraints to help megamodel users analyze various properties of simulation experiments. For instance, when a source metamodel definition of a model transformation is updated, associated source models must be checked for conformance. If the associated models do not conform to the updated metamodel, model transformation fails. A rule for checking the conformance relation on updates can be useful. Being an element of another model can be given as another rule example. An experiment model includes the information only related to the specified experiment, its simulation, design etc. The graphical user interface is given in another model. The graphical user interface model becomes an element of the experiment model. A rule between these models can be utilized to check that if meaningful visualization of the experiment results is feasible or not.

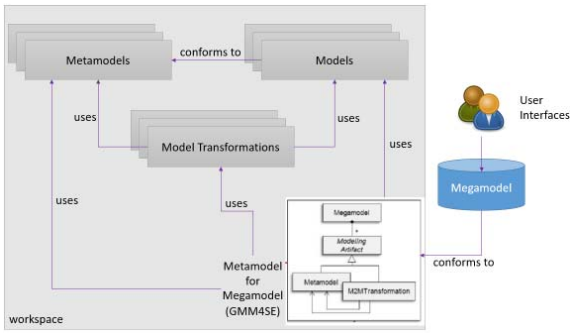

Figure 2: Overview of Megamodel for Simulation Experiments.

4.2 Construction of a Megamodel for Simulation Experiments

Our GMM environment for Simulation Experiments is implemented in the Eclipse environment. It is comprised of three main elements: (i) a workspace (model repository) for modeling artifacts, (ii) GMM4SE (metamodel definition for a megamodel), and (iii) model interfaces in Xtend for model operations such as model loading and model querying. By using the Model Interfaces, which are specified and generated by using Xtend, megamodel users are able to load, edit and delete models, query available models, create links among them and apply model transformation from/to available models. Additionally, the Model Interfaces layer aims to separate the megamodel from user interfaces. This separation leads to a flexible and reusable model management environment. The implementation components and the available user operations are shown in the Figure 3.

In addition, all metamodel definitions are specified by using ECORE (a part of EMF) in the Eclipse Modeling Framework (EMF) (Steinberg et al., 2009). We choose to employ the Eclipse-based platform because of implementation concerns. Eclipse is at the

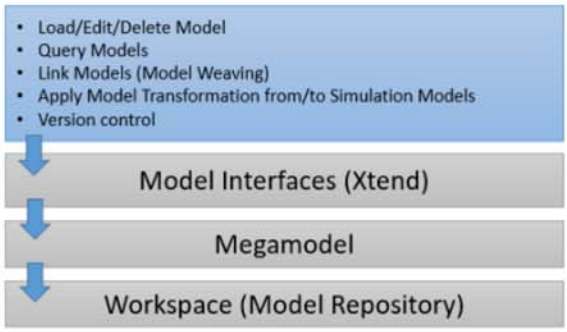

Figure 3: Megamodel Structure for Simulation Experiments components and User Operations.

foundation of an ecosystem that supports free and open software tools and languages like ECORE, EMF and Xtend. Additionally, we intend to use Eclipse for integration of the tools that we use for modeling, such as Atlas Model Weaver (Didonet et al., 2006), to establish relationships between models.

5 CASE STUDY: QUERYING XPERIMENTER MODELS

In this case study, an experiment involving a quadcopter is modeled using Xperimenter. A quadcopter, also known as a quadrotor helicopter or quadrotor, is a multi-rotor helicopter that is lifted and propelled by four rotors. Flight control of a quadcopter is a good example of how a Proportional-Integral-Derivative (PID) controller can be used to adjust some operational variables to hold an output variable at a setpoint. Validating the controller of the quadcopter by using actual test flights can be hazardous and costly. Therefore, we need to find a practical way to validate a controller. As the name suggests, PID control involves three basic coefficients, namely, proportional, integral and derivative (Kp, Ki, Kd). These coefficients can be varied to get an optimal response. We want to experiment with the model to find near optimal PID gains.

The quality of a controller depends on the gain values, and tuning these parameters require an experts intuition and time. By using Xperimenter this tuning is facilitated. First we need to define a research question: "Which gain parameter is most important to determine the quality of the controller?". Assume that we have a differential equation model for quadcopter flight and its gain parameters are configurable (preprocessed for this experiment). The Xperimenter code snippet in Figure 4 articulates a full factorial design for this experiment.

Our Quadcopter Xperimenter model in Figure 4

```
experiment QuadcopterExperiment {
	desc "Which PID parameter effects the controller quality most";
	objective COMPARATIVE;
	design CompQuadcopterExpDesign;
	simulation QuadcopterSim;
	analysis AnovaAnalysis;
	visual DEFAULT;
	target KEPLER;
}
variable Ki: INTEGER group FACTOR [1, 10];
variable Kp: INTEGER group FACTOR [1, 10];
variable Kd: INTEGER group FACTOR [1, 10];
variable AV: INTEGER group RESPONSE;
design CompMachineIntExpDesign {
	method FULLFACTORIAL;
	varlist Ki Kp Kd AV;
}
simulation QuadcopterSim {
	modelFile "c:\\quad_pid_sim.m";
	modelType DISCRETEEVENT;
	inport Ki: Ki;
	inport Kp: Kp;
	inport Kd: Kd;
	outport AV: AV;
}
analysis AnovaAnalysis {
	file "http://ceng.metu.edu.tr/~e1564178/xperimenter/anova-service";
}
```

Figure 4: Quadcopter Experiment model.

has a simulation element called *QuadcopterSim* and there exist four variables: *Ki, Kp, Kd* and *AV*. The model file and model type are specified in the simulation element. Additionally, the inports and outports are identified. An inport is a link to pull data from the outside of the simulation, and an outport is a link to push data to outside. The experiment design is called *CompMachineIntExpDesign* and the *FullFactorial* method is applied in the design. Finally, an Anova Analysis takes place; the Anova service accessible from the given URL is used.

Table 1: Quadcopter Models Input Variables.

	name	lowValue	highValue
Quadcopter 1 Variables	Ki	3	6
	Kd	5	9
	Kp	6	8
Quadcopter 2 Variables	Ki	2	7
	Kd	3	5
	Kp	1	8

Querying on two different Xperimenter models has been implemented by using Xtend. The example Xperimenter models are shown in Figure 4 and Table 1. The implementation of the querying is shown as an Xtend code snippet and the query result is shown in Figure 5. The Xperimenter domain models are rather simple for the purpose of illustration: Two models in the repository are queried whether their input variables lowest and highest values are between the required value.

As for the Xtend implementation for the querying in the code snippet, first the Xperimenter metamodel (defined in Section 2), is registered to the EMF registry, and then the two Xperimenter models are loaded as resource sets. These resource sets manage a collection of related resources that are received from

URIs and they can be loaded into a collection. In our query example, Xperimenter models are loaded into resource sets. From these resource sets, classes like experiment, simulation, design are obtained. Then, the query is executed. The query finds the input variables of the models which have the lowest value bigger than 2 and the highest value smaller than 8. The Xperimenter models input variables are described in Table 1. Finally, the query result is shown on the Eclipse console.

```
class XperimenterQuery {
    def static void main(String[] args) {
      InputOutput.<String>println
         ("\nXperimenter 1")
      new XperimenterQuery()
         .queryModel("Quadcopter1.xml")
      InputOutput.<String>println
         ("\nXperimenter 2")
      new XperimenterQuery()
         .queryModel("Quadcopter2.xml")
   }
   // Query the model
   def queryModel(String file) {
      // Register the EMF model
      doEMFSetup
      val resourceSet = new ResourceSetImpl
      // Load the model and get the resources
      val resource = resourceSet.getResource
         (URI.createFileURI(file), true)
      for (content : resource.contents)
         validateModel(content)
   }
   // Validate the model
   def validateModel(EObject o) {
      val expImpl = o as ExperimentImpl
      //find the variable which is between 2
          and 8
      for(v: expImpl.design.variables){
         if (v.lowValue > 2 && v.highValue<8)
            InputOutput.<String>println(v.name
              +"["+v.lowValue+",
                   "+v.highValue+"] is between
                   2 and 8")
      }
   }
   // Register Xperimenter packages to EMF
   def doEMFSetup() {
      EPackage.Registry.INSTANCE.put
         (XperimenterPackage.eINSTANCE.nsURI,
         XperimenterPackage.eINSTANCE);
      Resource.Factory.Registry.INSTANCE.
         extensionToFactoryMap.put("xml",
         new XMIResourceFactoryImpl);
   }
}
```

An important point is that the models are checked whether they satisfy structural conformance to their metamodels. At the beginning, Java class packages are generated from the Xperimenter Ecore metamodel. *doEMFSetup* method registers these packages to the EMF registry and checks metamodel conformance. Then, the model is loaded as a ResourceSet. If loading fails, this means the model conformance control has failed.

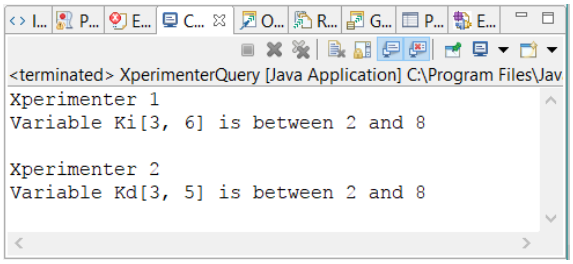

Figure 5: Xperimenter model validation result.

In this case study, we managed to load and query multiple Xperimenter models. This case study gives a basic idea about utilizing a megamodel in terms of models and metamodels, and it is an initial step for building an environment for Simulation Experiments.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we report on our ongoing effort aimed at creating a Global Model Management environment for simulation experiments. We take advantage of megamodeling techniques, which promote the reuse of available simulation experiment models by interrelating the models and using the models for a variety of operations such as, loading, editing, deleting, querying, applying model transformation. We presented a case study for Quadcopter experiment, implemented using Xtend, for rudimentary querying of Xperimenter models.

By using the GMM concept and megamodeling techniques, we substantiate that the concept is effective in connecting different technologies that serve the same purpose. we are planning to extend the environment for Simulation Experiments by integrating with other scientific workflow systems such as Kepler (Altintas et al., 2004) and Apache Taverna (Belhajjame et al., 2008) to support model variety. Moreover, we intend to base our simulation experiment environment on SED-ML (Waltemath and Novre, 2014). SED-ML is a standard language to encode simulation experiments. Extending our proposed megamodel using SED-ML will be a complementary step toward standardization of the model management environment. Additionally, with SED-ML, simulation experiment descriptions becomes exchangeable among

simulation software, and the validation and reuse of simulation experiments in different tools are enabled (Waltemath et al., 2011). Therefore, SED-ML compliance will promote the replicability of simulation experiments among users and software tools.

REFERENCES

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., and Mock, S. (2004). Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, pages 423–424.

Belhajjame, K., Wolstencroft, K., Corcho, O., Oinn, T., Tanoh, F., William, A., and Goble, C. (2008). Metadata management in the taverna workflow system. In *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 651–656.

Bézivin, J. (2005). On the unification power of models. *Software and System Modeling*, 4(2):171–188.

Bézivin, J., Jouault, F., Rosenthal, P., and Valduriez, P. (2005). *Modeling in the Large and Modeling in the Small*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Büttner, F., Bartels, U., Hamann, L., Hofrichter, O., Kuhlmann, M., Gogolla, M., Rabe, L., Steimke, F., Rabenstein, Y., and Stosiek, A. (2014). Model-driven standardization of public authority data interchange. *Sci. Comput. Program.*, 89:162–175.

Ceri, S., Palpanas, T., Valle, E. D., Pedreschi, D., Freytag, J.-C., and Trasarti, R. (2013). Towards megamodeling: A walk through data analysis experiences. *SIGMOD Rec.*, 42(3):19–27.

Combemale, B., DeAntoni, J., Baudry, B., France, R. B., Jezequel, J.-M., and Gray, J. (2014). Globalizing modeling languages. *Computer*, 47(6):68–71.

Denny Luan, C. W. (2017). Experiment. https://experiment.com/. Accessed: 2017-06-12.

Didonet, M., Fabro, D., Bzivin, J., and Valduriez, P. (2006). Weaving models with the eclipse amw plugin. In *In Eclipse Modeling Symposium, Eclipse Summit Europe*.

Efftinge, S. (2017). Xtend. https://www.eclipse.org/xtend/. Accessed: 2017-05-12.

Favre, J.-M. and NGuyen, T. (2005). Towards a megamodel to model software evolution through transformations. *Electron. Notes Theor. Comput. Sci.*, 127(3):59–74.

Fritzsche, M., Brunelière, H., Vanhooff, B., Berbers, Y., Jouault, F., and Gilani, W. (2009). Applying megamodelling to model driven performance engineering. In *16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, ECBS 2009, San Francisco, California, USA, 14-16 April 2009*, pages 244–253.

Goble, C. A., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D. T., Newman, D. R., Borkum, M., Bechhofer, S., Roos, M., Li, P., and Roure, D. D. (2010). myexperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic Acids Research*, 38(Web-Server-Issue):677–682.

Hebig, R., Seibel, A., and Giese, H. (2011). On the Unification of Megamodels. In Amaral, V., Vangheluwe, H., Hardebolle, C., Lengyel, L., Magaria, T., Padberg, J., and Taentzer, G., editors, *Proceedings of the 4th International Workshop on Multi-Paradigm Modeling (MPM 2010)*, volume 42 of *Electronic Communications of the EASST*.

Jouault, F., Vanhooff, B., Bruneliere, H., Doux, G., Berbers, Y., and Bézivin, J. (2010). Inter-dsl coordination support by combining megamodeling and model weaving. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2011–2018, New York, NY, USA. ACM.

Koegel, M. and Helming, J. (2010). Emfstore: a model repository for emf models. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 2, pages 307–308.

Montgomery, D. C. (2006). *Design and Analysis of Experiments*. John Wiley & Sons.

Simmonds, J., Perovich, D., Bastarrica, M. C., and Silvestre, L. (2015). A megamodel for software process line modeling and evolution. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 406–415.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition.

Vierhauser, M., Grnbacher, P., Heider, W., Holl, G., and Lettner, D. (2012). Applying a consistency checking framework for heterogeneous models and artifacts in industrial product lines. In *Proceedings of the 15th International ACM/IEEE Conference on Model Driven Engineering Languages & Systems (MODELS)*, Innsbruck, Austria.

Waltemath, D., Adams, R., Bergmann, F. T., Hucka, M., Kolpakov, F., Miller, A. K., Moraru, I. I., Nickerson, D., Sahle, S., Snoep, J. L., and Le Novère, N. (2011). Reproducible computational biology experiments with sed-ml - the simulation experiment description markup language. *BMC Systems Biology*, 5(1):198.

Waltemath, D. and Novre, N. L. (2014). Simulation experiment description markup language (sed-ml). In Jaeger, D. and Jung, R., editors, *Encyclopedia of Computational Neuroscience*. Springer.