Toward Model-Driven Engineering Principles and Practices to Support Model Replicability

Joseph Ledet[a], Alejandro Teran-Somohano[b], Zachary Butcher[a]
Levent Yilmaz [a], Alice E. Smith[b],
[a]Computer Science and Software Engineering
[b]Industrial and Systems Engineering
Samuel Ginn College of Engineering
Auburn University

Halit Oğuztüzün, Orçun Dayıbaş, Bilge Kaan Görür
Department of Computer Engineering
School of Engineering
Middle East Technical University

Keywords: Model Replicability, Model-driven Reproducibility, Transformation-driven Replication, Experiment Modeling

Abstract

Recent years have seen a proliferation of the use of simulation models in computational science. Most of these models have never been independently replicated by anyone but the original developer. Furthermore, there is a growing credibility gap due to widespread, relaxed attitudes in communication of experiments, models, and validation of simulations used in computational research. We examine various issues and challenges involved in model replication and simulation experiment reproducibility. Model-driven simulation engineering principles and model transformation concepts are adopted as solution strategies to improve replicability of models and reproducibility of experiments. A process model, an architectural framework, and an implementation strategy is introduced to address identified issues in simulation experiment management and model replication.

1. Introduction

Reproducible research is a fundamental principle of the scientific method (Morin et al., 2012; Fomel & Hennenfent, 2007). It refers to the ability to reproduce the experiments, and, if needed, independently replicate computational artifacts associated with published work. Emergence of *reproducibility* as a critical issue is based on growing credibility gap due to wide spread presence of relax attitudes in communication of the context, experiments, and models used in computational science (Mesirov, 2010; Stodden, 2010; Donoho, Maleki, Rahman, Shahram, & Stodden, 2009; Peng, 2009).

Replicability involves implementation of a conceptual model in a simulation study that is already implemented by a scientist or a group of scientists. Unlike reproducibility of results by (re)using the original author's implementation via *executable papers* (Nowakowski et al., 2011), *workflow systems and repositories* (Davidson & Freire, 2008; Freire, Bonnet, & Shasha, 2011), or *provenance-based infrastructures* (Koop et al., 2011), replications creating a new implementation differ in some way (e.g., platform, modeling formalism, language) from the original. Yet the original and replicate are sufficiently similar so that experiments conducted on both generate results that achieve pre-specified similarity criteria: they *cross-validate*. The premise of independent replication is based on the following observation. Although eventual exposure to the original model and its source code is important, if done too early, it may result in "groupthink" whereby the replicater, possibly unintentionally, adopts some of the original developer's practices: features of the original model are essentially "copied". In so doing the replicater has failed to maintain scientific independence. In other situations, replicaters may have different implementation tools and infrastructure, or may be unfamiliar with the original model's platform. Therefore, providing the ability to implement a conceptual model under specific experimental conditions and analysis constraints across multiple platforms and formalisms is critical to lowering the barrier to – and enabling broader adoption of – the practice of reproducibility. Furthermore, by ignoring the biases of the original model and replicating a model, differences between the conceptual and implemented models may be easier to observe.

To facilitate replicability, it is critical to provide the larger community with an extensible and platform neutral interchange language for specification, distribution, and transformation of model, simulator, and experimental frame elements. Support for – and a lowered technical barrier to – independent replication will increase trust in computational experimentation. Cross-validation will demonstrate (or not) that the original findings and observed results are not ex-

ceptional. Successful replications will strengthen the theories represented by the models. The objective of this article is to present a technical strategy to streamline the replicability of simulation-based computational research. The premise of our proposed approach stems from integrating platform neutral model, simulator, experiment design, and constraint specification languages with software engineering advances in model transformation to enable practical independent replication of computational research for cross-validation.

2. Related Work

Increasing number of computational science communities are emphasizing the role and significance of reproducibility. For instance, the MultiScale Modeling Consortium of the Interagency Modeling and Analysis Group (`http://www.nibib.nih.gov/Research/IMAG`) promoted credibility in multiscale modeling in biomedical, biological, and behavioral systems as a critical challenge. Among the proposed strategies include *executable papers* and *scientific workflow environments*. The Elsevier 2011 *Executable Paper Grand Challenge* provided a venue for exploring such practical and promising solutions. For instance, the *Collage* environment (Nowakowski et al., 2011), enables authors to seamlessly embed chunks of executable code (called assets) into scientific publications and allow repeated execution of such assets on underlying computing and data storage resources. *SHARE* (Sharing Hosted Autonomous Research Environments) is a web portal that enables authors to create, share, and access remote virtual machines that can be cited from research papers (Van Gorp & Mazanek, 2011). Gavish and Donoho (2011) introduce web and cloud-computing oriented concepts, which exploit the web infrastructure to achieve standard, simple and automatic reproducibility in computational scientific research.

In addition to executable papers, the scientific workflow systems research resulted in e-Science environments that improve reproducibility. While scientific workflow systems and repositories (Freire et al., 2011) such as *Swift*, *Taverna*, *Kepler*, and *Pegasus/Wings* describe experiments at different levels of abstraction and support reproducibility (Oinn et al., 2004; De Roure, Goble, & Stevens, 2009), they often lack support for tracing data, models, and computations to published manuscripts. *Vistrails* (Koop et al., 2011), however, as an open-source scientific workflow and provenance management system, allows users to create papers and Web publications whose figures and results can directly be tied to the computations that generated them. Sites like *crowdLabs* and *myExperiment* also support the sharing of workflows which describe computational workflows, data analyses, and visualizations. Crowdlabs also supports a Web-based interface for executing workflows and displaying their results on a Web browser. Code and text weaving systems such as *Sweave* and *Compendium* also provide support for reproducing papers, but rather assume availability of derived data to dynamically generate contents of a manuscript. Both tools enable embedding code in LaTeX documents in a way similar to the literate programming concept to create dynamic reports that are updated automatically if data or analysis change. However, text weaving tools do not provide packaging, distribution, introspective access to specifications for dynamic workflow interpretation, annotations, source files, dynamic updating for consistency management, and traceability between research artifacts encapsulated in a package. Ongoing efforts in executable papers and workflow systems are critical and made substantial advances toward reproducibility using the computational artifacts provided by the original developers. However, while reusing existing workflow and code scripts help verify published results, they carry the biases of the original implementation. Ongoing reproducibility work can be complemented with new strategies that exclusively aim to support independent replication of a study. To this end, we adopt a different perspective and unique approach to facilitate independent replication while streamlining cross-validation of computational research.

3. Issues and Challenges in Model Replication and Reproducibility

Replicating a simulation model in a target platform, possibly under a new formalism, requires transforming an existing representation into a behaviorally consistent version that can be used to conduct at least the same experiments for the purpose of the investigation. This requires the provision or generation of a platform-independent representation of both the problem and the solution space. Filtering platform-specific details while also focusing on aspects critical to replicating a model in a new platform, one has to identify the minimal information needed to facilitate cross-validation after a model is replicated. Manual mapping of a platform-specific model to a target platform-specific representation is error-prone. Furthermore, most developers are not familiar with the syntax and semantics of multiple platforms and/or formalisms, requiring automation of the process. High-level principles for automating the *replication* of simulation models include the following:

- Models must be defined in a well-defined notation allowing effective communication of their machine-processable abstract syntax.
- Specifications of simulations must be organized around a set of models and associated transformations facilitating mappings from their abstract syntax to common standard models that bridge multiple platforms.
- Models of both the abstract syntax and the semantics

of simulations must be explicitly defined to formulate meaningful and behavior-preserving mappings.

- For each modeling platform, using dedicated formats (e.g., SLX for SimuLink XML), syntactic mappers should inject simulation software as models . Similarly, extractors should be provided to map target abstract syntax onto target simulation software.

- Semantic mappers need to be provided to align the concepts, mechanisms, and constraints in the source formalism to the concepts, mechanisms, and constraints in the target formalism.

Following the replication of a model in the target formalism, simulation experiments need to be conducted to reproduce results for cross-validation. Experiments drive simulators that interpret and execute models to generate their behavior and hence are critical in reproducing results. To facilitate *systematic reproducibility*, we promote the following requirements in relation to *experiment models*:

- Experiments should have an explicit model and be managed in a way similar to simulation models. In practice, however, experiments are not explicitly modeled, communicated, and transformed.

- Experiment designs should at least conform to standard Design of Experiments (DOE) model and be interpreted by a repeatable and executable experiment workflow that coordinates simulators in accordance with the experiment model.

- Experiment models should be specified at a high-level and platform-neutral abstraction consistent with the DOE model and be amenable to translation into low-level scripts that execute experiments on a selected (distributed) platform.

4. Model-Driven Engineering: Principles and Practical Implications

The purpose of Model-Driven Engineering (MDE) is to enhance the development of software artifacts by creating *domain models* during each phase of the software development process. In this way, the team developing software can develop the functionality and the high-level aspects for a system using modeling tools rather than implementing in a particular programming language (Atkinson & Kuhne, 2003). MDE can be useful for addressing the issues discussed in Section 3. Meta-models provide a standard structure for the models they represent. This provides for greater likelihood of models to be well-defined, provided the meta-models are properly defined. MDE's use of model transformation processes allows for multiple platforms to be used to represent models used in simulations. Automated transformations allow for the development of models in multiple simulation environments without requiring the developers to have knowledge of both platforms involved in the transformation. In fact, if Platform Independent Models (PIMs), another artifact in MDE, is utilized, then a developer would not need familiarity with a particular modeling platform.

4.1. Metamodeling and Model Transformation

One of the hallmarks of MDE is the use of *meta-modeling* to define the structure of a domain model (Schmidt, 2006). Each model can be considered to be an instance of the meta-model defining the relationships among the components. Also, each level is connected strongly to the one above it in that any artifacts created at one level must conform to the definitions given in the level above it.

Meta-modeling is a useful tool in properly performing model transformations. The relationships between the source and target models and their respective meta-models determine one aspect of the type of transformation being performed. In endogenous transformations, target and source models conform to same meta-model. On the other hand, in exogenous transformations, meta-models of target and source models are distinct. The relative abstraction levels of the source and target models determine the other aspect of the transformation. Horizontal transformations are performed when target and source models are at the same abstraction level; vertical transformations are when target and source models have differing abstraction levels (Mens & Van Gorp, 2006). The nature of the transformation performed using the intersection of these aspects is shown in Table 1.

Table 1. Orthogonal dimensions of model transformations

	Horizontal	**Vertical**
Endogenous	Refactoring	Formal Refinement
Exogenous	Language Migration	Code Generation

In terms of the above definitions, simulation model replication is a language migration type of transformation and reproducibility involves code generation from explicit experiment models to conduct simulation experiments.

4.2. Megamodeling

To facilitate registration of various models and simulations that will be used for enhancing replicability, we will make use of the *megamodeling* technology (Bézivin, Jouault, & Valduriez, 2004). Megamodeling is an effective strategy for maintaining a repository of models, algorithms, simulators, experiment models, and experiment results. By storing and sharing the details and results of simulation experiments,

model replicability and computational reproducibility can be improved.

The *Simulation Experiment Markup Language (SED-ML)* is developed to exchange models in the biological modeling domain (Waltemath et al., 2011). A complete SED-ML specification of an experiment can be formalized as a megamodel allowing us to leverage the principles established by SED-ML to address the pertinent challenges expressed in Section 3. SED-ML encapsulates multiple combinations of models and algorithms within the same SED-ML file. A SED-ML file can also include changes to the underlying details of a model (parameter values, constraints, etc.) without altering the source model. These features allow for researchers to change the details of a simulation experiment without requiring knowledge of the modeling platform nor making changes to the original model.

5. Transformation-driven Model Replication

In order to precisely address the issues presented in Section 3, we now present a specific proposal to implement the concepts presented in Section 4.

5.1. Models and Simulators

There are numerous modeling technologies representing a range of modeling paradigms. In an effort to enhance model replication by producing a horizontal, exogenous transformation process (see Table 1), following a vertical transformation from a Platform-Specific Model (PSM) to Platform-Independent Model (PIM), the selection of source and target modeling platforms is needed. We have selected MATLAB Simulink as our source platform. This modeling paradigm is focused on structural elements and complex mathematical operations. On the other hand, our target platform, RePast, is agent-based and deals with activity flows. These two platforms were selected due to the differences in paradigms to give a broader range for our transformation process to represent. A sufficient platform independent environment will have components that are able to represent the elements of the source model. The modeling paradigm selected should be a widely accepted standard in the field of engineering. We believe *SysML* is a strong candidate for representing PIMs generated from our transformation process for the following reasons:

- Similar to Simulink, SysML uses blocks, ports, and connections to represent the flow of data.
- SysML is considered an industry standard for model development.
- SysML includes the types of analysis diagrams, such as the Requirements Diagram, that are useful in the development life cycle.
- Extensible tools exist to view and create SysML diagrams (e.g., Eclipse - Papyrus, Rational Rhapsody, Modelio, Visual Paradigm).

5.2. A Process Model for Transformation-Driven Replication

The process for creating a PIM in SysML from a given source model in Simulink is shown in Figure 1. As can be observed, a user or model developer will submit a Simulink model file. The sub-process for generating an XMI representation of the Simulink source model for use in an ATL transformation is expanded in Section 5.2.1.

5.2.1. Model Discovery

Simulink supports two file formats, MDL and SLX. If, the format received is MDL, the equivalent SLX file is generated using a similar script to the one that is available at (*SLXTranslator*, 2013) The source XML is derived from the Simulink model by extracting the contents of the SLX file and navigating to the sub-directory named "simulink". In this folder, "blockdiagram.xml" contains the details of the model in an XML format. The details of how this XML file is translated into the equivalent XMI representation is shown in Figure 3. The primary necessity for generating the XMI representation is that ATL identifies the values of properties as attributes of the object. Therefore, through an XSL Transformation, the XML tag elements in the Simulink XML are converted to attributes with the attribute name "Value". The parsing of XML Tags, performed prior to the conversion of elements to attributes, is due to the nature of the non-standard representation that Simulink uses for array values. This sub-activity generates a standard array format.

Once the XMI representation for the Simulink model is realized, it is used in the ATL transformation process as shown in Figure 2. The ECORE meta-model and the meta-model for ATL rules are defined in EMF. However, the three meta-models for the individual models (Simulink, Source, and SysML) and the two sets of transformation rules (Simulink-to-Source and Source-to-SysML) must be defined.

For illustration, we present the corresponding SysML block diagram for the *Team 1 Strategy* model file. This model is a sub-model of the Robot Soccer model (*RoboSoccer*, 2011) used in our case study. The model includes three inputs (Simulink Inport Blocks), two outputs (Simulink Outport Blocks), and a Code Block (Simulink S-Function Block) with the MATLAB code embedded. The Simulink GUI representation of this example is given in Figure 5.

This model can reasonably be represented by the SysML Diagrams shown in Figure 6. The SysML representation for the Internal Block Diagram, which is shown in Figure 6(b), has a similar structure to the Simulink GUI source. However, in the source model file, for the Simulink S-Function block,

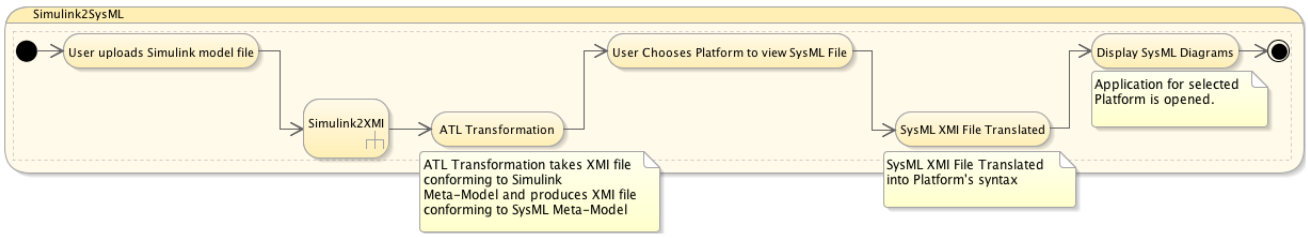


Figure 1. Process for SysML Model Generation

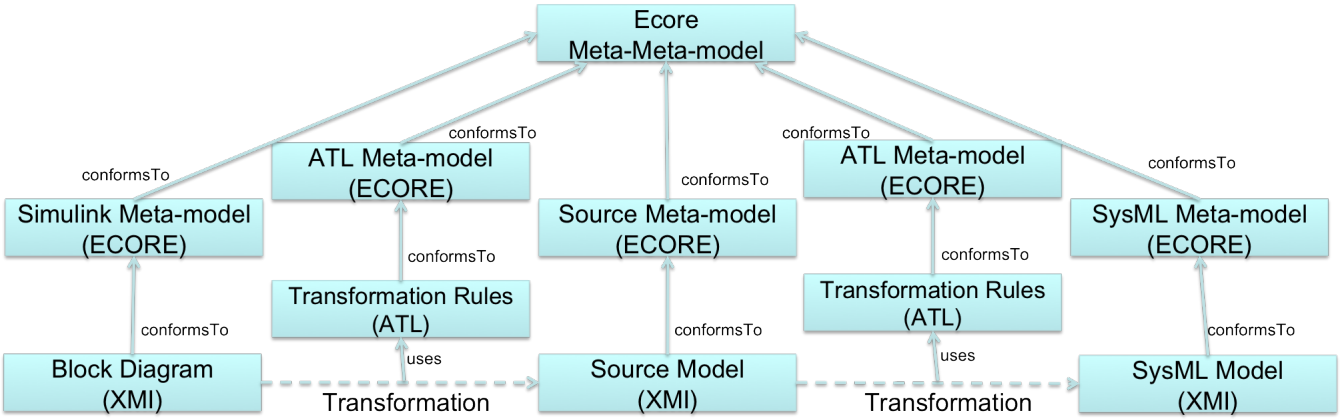


Figure 2. ATL Transformation

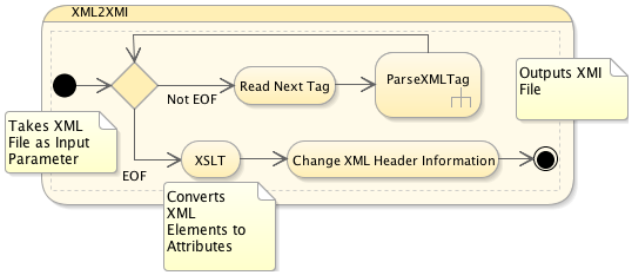


Figure 3. XML transformed to XMI

we observe additional Simulink blocks. The general structure of the source model is as follows:

- Embedded Code block is represented as a Simulink Block of type ”SubSystem”.
- This block has a Simulink System with the following structure:
  - One Simulink Block of type ”S-Function” with a Property called ”Tag” that connects to the Stateflow section for the source code behind the block. This block has n input ports and $m+1$ output ports where n and m are the number of inputs and outputs to the Embedded Code block, respectively.
  - n Simulink Blocks of type ”Inport”, each with Port number i connected to ”in” port number i of the S-Function Block.
  - m Simulink Blocks of type ”Outport”, each with Port number k connected to ”out” port number $k+1$ of the S-Function Block.
  - One Simulink Block of type ”Demux” with its ”in” port connected to ”out” port number 1 of the S-Function Block.
  - One Simulink Block of type ”Terminator” with its ”in” port connected to ”out” port of the Demux Block.

This format is visually represented using the Team 1 Strategy example in Figure 7. As such, we include in the ATL specification a rule for this given structure and transform the block properly. For instance, the ATL rule shown in Figure 4 aims to map a SimuLink block with an SFunction to a SysML block annotated as Embedded Code.

The source MATLAB code from the Embedded Code Block is given in Figure 8. The corresponding SysML Activity Flow diagram is shown in Figure 9.

```
rule SFunction2Block {
   from
      s: Simulink!Block (s.isSubSystem() AND s.hasSFunction
   to
      t: SysML!Block (
         name <- 'Embedded Code'
         operations <- 'Function(' + s.getInports() + ')'
      )
}
```

Figure 4. An ATL Rule

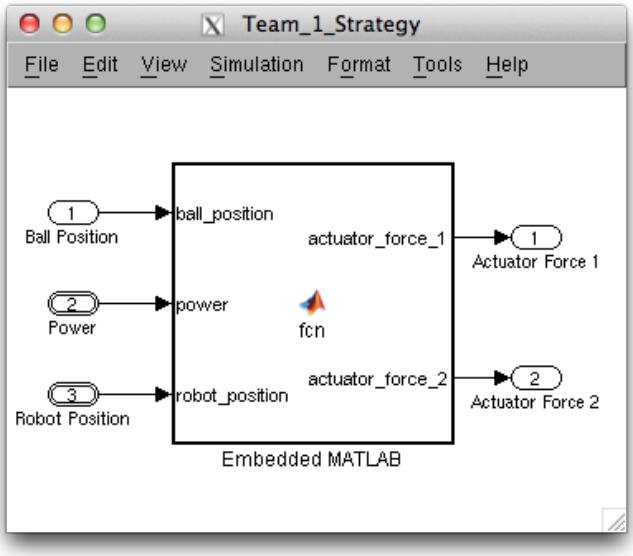
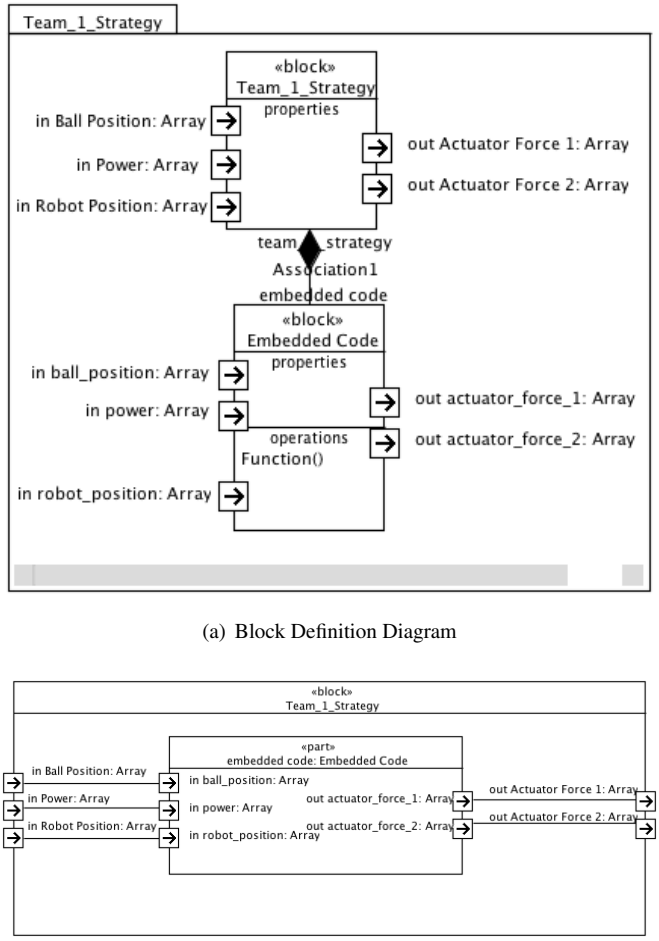


Figure 5. Simulink GUI for Team 1 Strategy



(a) Block Definition Diagram

(b) Internal Block Diagram

Figure 6. SysML Structural Diagrams for Team 1 Strategy

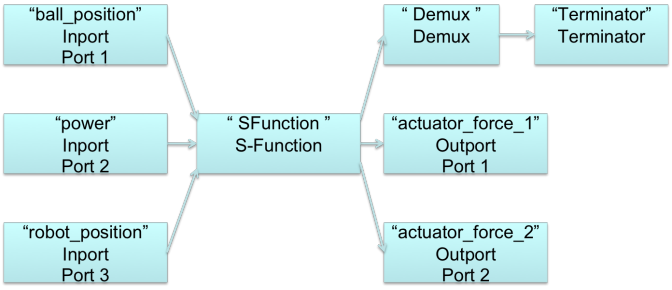


Figure 7. Simulink Block Structure for Embedded Code Block

5.2.2. Model Understanding

In order to accurately transform models from our SysML PIM representation into a target formalism, a process is needed for generating a version of the model that conforms to the target environment. While this representation would not be accurately defined as a PSM (i.e. it can still be considered a PIM), it will be based on features that are relevant and similar to the constructs of the target environment than the SysML block structure. The Simulink Blocks that can be routinely translated to SysML block elements, such as the Integrator Block, need to be refined into an activity flow for use in an agent-based environment, such as RePast. These representations can still be represented using SysML, but unlike the SysML representations generated from the Simulink transformation, these PIMs will place less emphasis on the block structure and gradually change the focus to the process-flow.

5.2.3. Model Generation

Once the bridge PIMs discussed in Section 5.2.2 are realized, a process for transforming these PIMs into an agent-based modeling paradigm is needed. This transformation is performed in a similar manner as the process for creating the PIM from the Simulink source. Initially, an XMI representation should be created from an ATL transformation. Once this XMI representation is complete, a translation process to create agent files that conform to RePast activity flow specification can follow. When the agent specifications are finalized,

```
function [actuator_force_1,
actuator_force_2] =
fcn(ball_position, power,  robot_position)

d1=(robot_position(1:2,1)-ball_position);
d2=(robot_position(3:4,1)-ball_position);
if dot(d1,d1)>5
   actuator_force_1=-22*d1;
else
   r1=(ball_position- [200;50]);
   actuator_force_1=-11*r1;
end
if dot(d2,d2)>5
   actuator_force_2=-25*d2;
else
   r2=(ball_position-[200;50]);
   actuator_force_2=-15*r2;
end
```

Figure 8. Embedded Code

RePast creates the necessary Groovy files to complete the model replication process. With the RePast model, simulation experiments need to be performed to verify that the replicate's behavior is sufficiently similar to the original model's behavior.

5.3. Future Work

Once the MDE-based process for transforming source models to/from a specific platform from/to a PIM is complete, we will start developing the *megamodeling* facility and implement a user interface to reproduce simulation experiments over the replicated model. A user will be able to submit a source model along with details such as range of variable values, experiment design, and the output styles. The system will present the simulation results using the output styles and the analysis types defined in the explicit experiment model. In this paper, we overviewed a process for creating Platform Independent Models from Platform Specific Models with a focus on replicability and reproducibility, This process, which is coupled with the Model-Driven Engineering principles and practices, contribute to model replicability, thereby enhancing experimental reproducibility. These models can be used to re-run simulations to cross-validate results of previous experiments or with altered details, such as simulation environment or change in parameter values, to expand existing understanding resulting from previous experiments. A focus on model replication and simulation reproducibility is expected to increase the credibility of simulation-based scientific research.

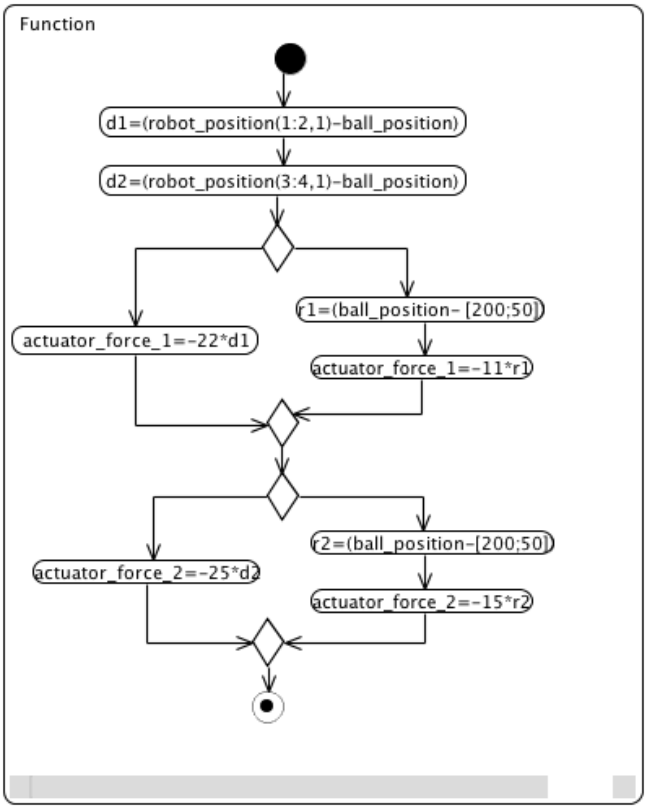


Figure 9. SysML Activity Flow Diagram for Team 1 Strategy

6. Conclusion

In this article, we define replicability and reproducibility as critical criteria to improve assurance and independent cross-validation of simulation systems. The principles and the methodology underlying the Model-Driven Engineering paradigm are described and proposed as a solution strategy to address issues and challenges in model replication and simulation reproducibility. A case study based on the SimuLink platform is used to introduce the concepts and to propose transformation-driven independent replication of models. Explicit separation of simulation experiment models from simulation models facilitate application of the MDE methodology to synthesize executable experiment scripts that serve as workflows coordinating the simulators and analysis engines to perform goal-directed experimentation. Besides its support for independent replication of models, the MDE approach can be used to support interoperability, model longevity, and modernization of legacy simulation systems.

In distributed simulation systems, interoperability refers to the ability of system models or components to exchange and interpret information in a consistent and meaningful manner. This requires both syntactic and semantic congruence

between systems either through standardization or mediators that can bridge the syntactic and semantic gap between peer components. The transformation models used in the MDE approach can also serve as mediator or adapter components by aligning the concepts of a simulation system with those adopted by a standard specification. Many organizations are facing challenges in maintaining, managing, and modernizing or replacing simulation systems and infrastructures that have been actively used for a long time. These simulation systems may continue to play critical roles in training, education, or decision-making, but they are often based on obsolete technology, and this makes them difficult to port and interoperate with applications developed using emergent advanced technologies. Modernization of such legacy simulation applications can also take advantage of MDE to generate useful high-level representations and model-based views of systems.

References

Atkinson, C., & Kuhne, T. (2003). Model-driven development: a metamodeling foundation. *Software, IEEE*, *20*(5), 36–41.

Bézivin, J., Jouault, F., & Valduriez, P. (2004). On the need for megamodels. In *Proceedings of the oopsla/gpce: Best practices for model-driven software development workshop, 19th annual acm conference on object-oriented programming, systems, languages, and applications.*

Davidson, S. B., & Freire, J. (2008). Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 acm sigmod international conference on management of data* (pp. 1345–1350).

De Roure, D., Goble, C., & Stevens, R. (2009). The design and realisation of the myexperiment virtual research environment for social sharing of workflows. *Future Generation Computer Systems*, *25*, 561–567.

Donoho, D. L., Maleki, A., Rahman, I. U., Shahram, M., & Stodden, V. (2009). Reproducible research in computational harmonic analysis. *Computing in Science and Engineering*, *11*(1), 8–18.

Fomel, S., & Hennenfent, G. (2007). Reproducible computational experiments using scons. In *Acoustics, speech and signal processing, 2007. icassp 2007.* (Vol. 4, pp. IV–1257).

Freire, J., Bonnet, P., & Shasha, D. (2011). Exploring the coming repositories of reproducible experiments: Challenges and opportunities. *Proceedings of the VLDB Endowment*, *4*(12), 9-27.

Gavish, M., & Donoho, D. (2011). A universal identifier for computational results. *Proceedings of the International Conference on Computational Science*, *4*, 637–647.

Koop, D., Santos, E., Mates, P., Vo, H. T., Bonnet, P., Bauer, B., et al. (2011). A provenance-based infrastructure to support the life cycle of executable papers. *Procedia Computer Science*, *4*, 648–657.

Mens, T., & Van Gorp, P. (2006). A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, *152*(125–142).

Mesirov, J. P. (2010). Accessible reproducible research. *Science*, *327*(5964), 415–416.

Morin, A., Urban, J., Adams, P. D., Foster, I., Salli, A., Baker, D., et al. (2012). Shining light into black boxes. *Science*, *336*, 159-160.

Nowakowski, P., Ciepiela, E., Harezlak, D., Kocot, J., Kasztelnik, M., Bartyński, T., et al. (2011). The collage authoring environment. *Procedia Computer Science*, *4*, 608–617.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., et al. (2004). Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, *20*(17), 3045–3054.

Peng, R. D. (2009). Reproducible research and biostatistics. *Biostatistics*, *10*(3), 405–408.

Robosoccer. (2011). http://www.mathworks.com/matlabcentral/fileexchange/28196-robot-soccer-an-exercise-in-learning-the-key-features-of-simulink.

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *Computer*, *39*(2), 25–31.

Slxtranslator. (2013). http://www.mathworks.com/help/simulink/examples/converting-from-mdl-to-slx-model-file-format-in-a-simulink-project.html.

Stodden, V. (2010). *The scientific method in practice: Reproducibility* (Tech. Rep. No. 4773-10). MIT Sloan Research Paper.

Van Gorp, P., & Mazanek, S. (2011). Share: a web portal for creating and sharing executable research papers. *Proceedings of the International Conference on Computational Science*, *4*, 589–597.

Waltemath, D., Adams, R., Bergmann, F. T., Hucka, M., Kolpakov, F., Miller, A. K., et al. (2011). Reproducible computational biology experiments with sed-ml-the simulation experiment description markup language. *BMC systems biology*, *5*(1), 198.