Peer Reviewed Paper Presented at AlaSim 2015

A Hybrid Transformation Process for Simulation Modernization and Reuse via Model Replicability and Scenario Reproducibility

Joseph Ledet
Department of Computer Science and Software Engineering, Auburn University, Auburn, Alabama, USA
jwl0008@auburn.edu
Sema Cam, B. Kaan Gorur, Orcun Dayibas, and Halit Oguztuzun
Department of Computer Engineering, and Modeling and Simulation R&D Center, Middle East Technical University, Ankara, Turkey
semacam_87@yahoo.com, kgorur@metu.edu.tr, odayibas@gmail.com, oguztuzn@ceng.metu.edu.tr
Levent Yilmaz
Department of Computer Science and Software Engineering, Auburn University, Auburn, Alabama, USA
yilmale@auburn.edu
Alice E. Smith
Department of Industrial and Systems Engineering, Auburn University, Auburn, Alabama, USA
smithae@auburn.edu

Keywords: Model Driven Engineering, Experiment Management

ABSTRACT:*Due to the increased use of simulation models in computational scientific experimentation, the need for reuse as well as continuous modernization and hence independent replication and reproduction of the models and results of experiment simulation executions has become essential. The desire to refrain from sharing all details of a simulation experiment can, and often does, come from legitimate and necessary reasons, such as a need to maintain security or intellectual property rights. However, the negative impact of not being transparent in these details has resulted in a decrease in credibility of the results obtained from such experiments.*

In this paper, we detail the developing of a process utilizing Model-Driven Engineering (MDE) principles, specifically Model Transformations, to produce Platform Independent Models (PIM) for the purpose of enhancing the reliability of the existing Platform Specific Models (PSM). Additionally, we describe the concurrent development of the process to transform existing PIMs into PSMs for the purposes of executing these simulations to validate the results obtained in previous deployments. We attempt to address the current state of the development of this process, including Atlas Transformation Language (ATL) rules and Extensible Stylesheet Language Transformations (XSLT) examples. We address many of the lessons learned from developing a hybrid process using multiple transformation tools. Finally, we address the expectations of how this process will be expanded in our future work.

1. Introduction

One of the hallmarks of scientific experimentation is the ability to independently verify the results obtained from previous experiments. As the ability to perform experiments using simulation technology has become more widespread, so has the capability to make these simulation models more complex for executing larger simulations and more detailed experiments. Additionally, the platforms available for developing these models has also increased. Unfortunately, this has resulted in increased difficulty in validating the results obtained due to the inability or lack of persistence in independently confirming the conclusions of prior simulation experiments [1, 2]. By providing a means to easily and accurately reproduce and replicate the models used in scientific simulation-based experiments, more confidence can be generated in the results of these experiments.

2. Related Work

2.1. Replicability and Reproducibility

Fundamental to following the scientific method for obtaining credible results is the ability to perform reproducible research [3, 4]. The reproduction or replication of experiments by independent researchers is critical to maintaining reliability in the science community. While *reproducibility* refers to the reuse of an original author's methodology and even platforms, models, and source code, *replicability* refers to the implementation of a similar, yet slightly different simulation to validate the results previously obtained.

2.2. Hybrid Transformations

For the purposes of this paper, we have defined *Hybrid Transformation* as a model transformation that utilizes one or more transformation language tools or technologies. These approaches are very rare due to the challenges of merging differing technologies [5]. Some approaches can be found for the purposes of generating deployable applications from UML models [6].

Another factor in the difficulty of utilizing multiple transformation technologies lies in the ability to “chain” the various transformations together in a seamless process. In this chain, it is required that the output of one transformation be a valid input to a second transformation [7, 8].

3. Transformation Utilities

Many useful tools and technologies have been made widely available for facilitating model-to-model transformations. A comprehensive discussion on all existing transformation utilities is beyond the scope of this paper. In this section we have undertaken the task of describing the two utilities we have chosen for use in our hybrid approach to executing a transformation from Simulink to SysML and vice versa.

3.1. ATL

The ATLAS Transformation Language (ATL) is the facilitator of model transformations for the ATLAS Model Management Architecture (AMMA), which is a framework for the development of models within the Eclipse project[9, 10]. AMMA is built onto the Eclipse Modeling Framework (EMF), a modeling framework and code generation utility for building tools and applications in Eclipse[11]. Some of the key features of ATL are as follows:

- To create a one-to-many mapping of a target element for each source element declarative rules can be declared. Ideally only these rules will be used in a transformation.
- Makes use of imperative structures.
    - Imperative rules can be called explicitly from declarative rules or imperative structures.
    - Imperative instruction blocks can be declared.
- Transformations are unidirectional.
    - In order to perform a transformation in the “reverse” direction a separate set of transformation rules must be developed.

A transformation process with ATL as its backbone will enjoy benefits because:

- ATL is supported by EMF and the developers will have access to a number of useful tutorials [12]
- Once the rules are developed, the execution of transformations is a simple process using an instance of Eclipse
- The three meta-models for source, target, and ATL transformation rules will conform to the same meta-meta-model, namely ECORE the format for defining meta-models in EMF as shown in Figure 1[13].

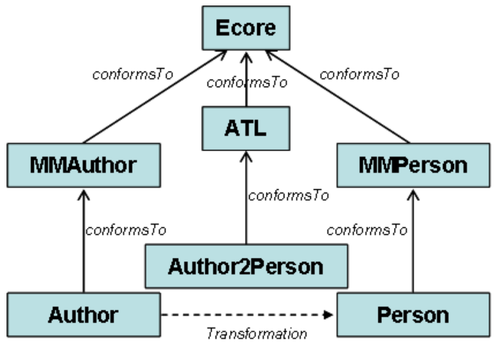

Figure 1. Graphical Representation of the ATL Process

To successfully complete a transformation using ATL, four components must be available or developed:

1. A meta-model for the source model
2. A meta-model for the target model
3. A source model conforming to the source meta-model
4. A set of ATL rules conforming to the ATL meta-model

When transformation is executed using these four components as inputs, a target model conforming to the target meta-model is produced.

3.2. XSLT

The Extensible Stylesheet Language Transformations (XSLT) is a simple language for transforming Extensible Markup Language (XML) documents into other XML documents without changing the original XML file [14]. A transformation utilizing XSLT should be developed conforming to the vocabulary of the Extensible Stylesheet Language (XSL). Similar to ATL, XSLT follows a transformation pattern of taking a source model that conforms to a particular meta-model, applying a set of transformation rules written in the XSL format, and generating a target model that conforms to a second meta-model. However, unlike ATL, neither of the meta-models must be explicitly defined.

3.3. Comparison of ATL and XSLT

Based on the experience we have gained concerning ATL and XSLT, we have seen the value that each can provide in performing a transformation or series of transformations. We have also noted some of the drawbacks or shortcomings that each has, as well. A comparison of these two transformation utilities can be seen in Table 1. Because of the some of the ways that each of these tools complementsthe other, we found much promise in the concept of developing a hybrid process for performing the overall desired transformations.

ATL	XSLT
Defined metamodels	No requirement for metamodels
Source model conforms to MM	Any source model is valid
Exceptions halt target model generation	Incorrect transformation rules produce unanticipated results
Each source object with a rule produces its target object(s)	Source objects must be explicitly transformed
Recursion difficult	Recursion easy
Looping over collections possible	Looping not possible
Difficult to implement	Easy for small changes

Table 1. Comparison of ATL and XSLT

4. Process

Here we have undertaken the task to briefly explain the methodology to develop our model transformation processes. We describe the case study we opted to use as a starting point for the development of the processes. We briefly define the metamodels to be used in the transformations. Finally, we explain some of the transformations developed to facilitate the overall transformation from Simulink to SysML and vice versa.

4.1. Case Study

The first step in the development of our process was to select an appropriate example model developed using Simulink to serve as a case study. Ideally, this model would contain a range of characteristics and functionality available in MATLAB and Simulink while still remaining simple enough for us to quickly gain an understanding of the processes and tools necessary to complete the transformation process, including the reverse engineering of Simulink models and the development of XSLT and ATL rules.

4.1.1. Robot Soccer

We selected the Robot Soccer model available through MATLAB Central [15] shown in Figure 2.The model actually consists of three separate Simulink model files listed as follows:

1. Robot Soccer
   a. Main model file
2. Team 1 Strategy
   a. Submodel for determining robot actuator force for robots 1 and 2
3. Team 2 Strategy
   a. Submodel for determining robot actuator force for robots 3 and 4

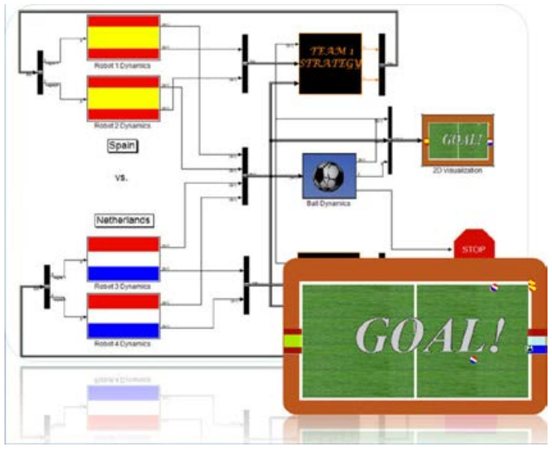

Figure 2. Robot Soccer Simulink Model

This model has the following characteristics:

- Utilizes 19 distinct Simulink block types
  - o 15 of these block types are listed in the 23 “Commonly Used Blocks” representing 65% of these block types
- Incorporates SubSystems
  - o Blocks which have structures inside of them including other blocks
- Model References
  - o Team Strategy submodels referenced in Robot Soccer
- Models with inputs and outputs
  - o Team Strategy submodels take inputs through inports and produce outputs through outports
- S-Function Blocks are used
  - o Blocks which have embedded MATLAB code

We felt that this model served as a good case study due to its representative qualities of a typical Simulink model without being too complex to serve as an introduction to Simulink.

4.1.2. Process

Similar to the way we envision our case study being an introduction to Simulink from which we will expand the process to encompass all of Simulink's functionality, we decided to begin our process by developing the transformation definition for the Team Strategy submodels. After this was complete, we would expand the transformation definition to handle the complete Robot Soccer model.

4.1.2.1. Team Strategy Models

The two Team Strategy models are very similar. They each take three input parameters:

1. Robot Position
    a. An 8 x 1 matrix representing the (x, y) coordinates of all four robots
2. Power
    a. A 4 x 1 matrix representing the remaining energy for the two robots on the respective team
3. Ball Position
    a. A 2 x 1 matrix representing the (x, y) coordinates of the ball

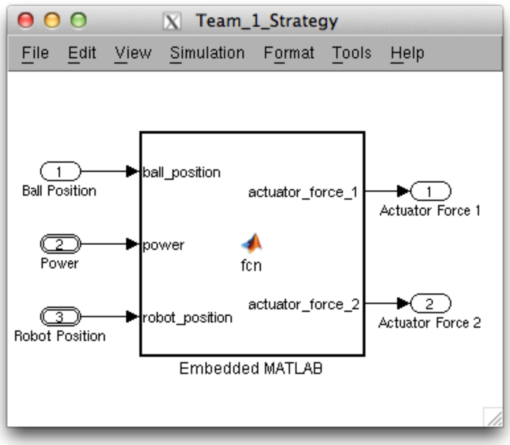

Figure 3. Team 1 Strategy Simulink Model

These models each generate two output matrices representing the calculated actuator forces for the two robots on the respective team. This calculation is performed using a single S-Function block. The structure for Team 1 Strategy is shown in Figure 3.

We considered that the transformation of these submodels would enhance our understanding for how to manage blocks, ports, and connections existing in a Simulink model.

4.1.2.2. Robot Soccer Expansion

The Robot Soccer model is a much more complex model that when executed will simulate all four robots on the two teams and their interactions with the ball and boundaries of the field. In the version of the model available for download, the robots, ball, and field are displayed using a two-dimensional graphical representation. We decided for the sake of comparison of the original model's generated data with the transformed model's data we would change the two-dimensional plot to a Simulink "To Workspace" block for use in exporting the data to a spreadsheet [16].

In addition to the Simulink functionality covered in the Team Strategy models, the following functionality will be analyzed and understood for the transformation process as part of the Robot Soccer expansion:

- Blocks – Simulink BlockTypes other than S-Function that perform specific calculations
- Systems – Simulink Blocks that contain a set of other Blocks and interconnections
- Branches – Connections within a Simulink System where the output from one Block is connected to the inputs on more than one other Block

4.2. Metamodel Definitions

In order to execute the transformations we intended to develop, we would also require access to the metamodels for any models that would be the source or target of any ATL transformation.

4.2.1. Simulink

We developed this metamodel by hand using the Ecore Diagram utility in Eclipse. We based this metamodel on the general structure of the Simulink models for Robot Soccer and the Team Strategy. We endeavored to make this metamodel as general as possible so as to have this metamodel be useful for any future Simulink models used in our transformation process.

4.2.2. Source

This metamodel was also developed by hand in Eclipse. It makes use of the Composite Design Pattern for the Block objects. Each Block can be a Leaf object or a System which is a Composite object.

4.2.3. SysML

For this metamodel we decided to use the existing metamodel available in an Eclipse distribution that has Papyrus installed. Specifically, we used the Kepler version of Eclipse with the Papyrus package installed. By using this metamodel, we have access to many objects that will not be necessary for our transformations, but we can be more confident that our resulting SysML models will be valid structurally for use in Eclipse-Papyrus.

4.3. Simulink to SysML

In this section we detail the process developed to perform the necessary transformations to generate an equivalent SysML model from a given Simulink source model. Figure 4 shows the overall transformation process including how the various models in the process conform to their respective metamodels. The remainder of this section is dedicated to specifying the steps in this process.

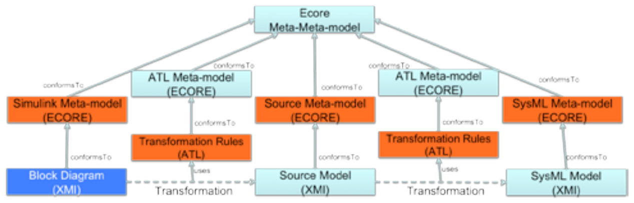

Figure 4. Transformations Process

4.3.1. Intermediate Models

Early in the implementation of this process, it became clear that the various pieces of information for the "Source" model in a transformation would need to be gained from differing sources within the Simulink model and built using a series of intermediate models. Generally, each of these models would build upon the previous intermediate model in the sequence adding the next level of pertinent information. The overall transformation process involving the intermediate models used can be seen in Figure 5. In the remainder of this section we briefly describe the purpose for each intermediate model. We also include example rules from XSLT and ATL to exhibit how each was used in our processes.

4.3.1.1. Simulink Settings

The Simulink Settings model serves as a container for necessary settings present as part of the XML within a Simulink model. Many of these settings are execution settings and information for use by MATLAB when executing a simulation. This model exists primarily for use in the SysML to Simulink transformation process detailed in Section 4.4.

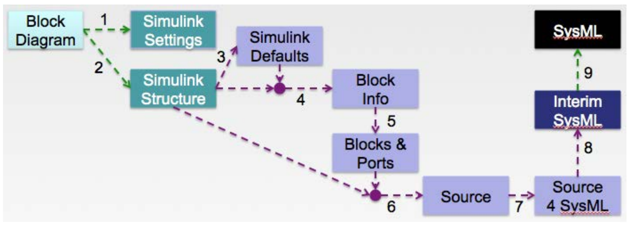

Figure 5. Simulink to SysML Transformation Process

4.3.1.2. Simulink Structure

The Simulink Structure model is the pertinent information gained from a Simulink block diagram dealing with structural elements such as Blocks, Lines, and Properties. The necessity for this model is primarily that:

1. The values for Properties in the XML are stored as Content values in the P tag as seen in Figure. However, ATL transformations are only able to access values stored as Attribute values. Therefore, it is necessary to change the storage type of these values.
2. Matrix representations in Simulink are expanded to be understood as array representations. This is necessary due to the multiple formats MATLAB uses to represent matrices.

In the example rule shown in Listing 1, when an element with a tag of "P" is encountered, a new element is created with the name "Property". Within that element, the template "ProcessPrimitiveType" creates another element called "Value" with an Attribute also called value. The value for the Attribute is passed as a parameter called "PrimitiveValue".

```
<xsl:template match="P">
<xsl:element name="Property">
<xsl:variable name="value" select="." />
<xsl:call-template
       name="ProcessPrimitiveType">
<xsl:with-param name="PrimitiveValue"
       select="$value"/>
</xsl:call-template>
```

Listing 1. Simulink Structure Rule

4.3.1.3. Simulink Defaults

The model by the first ATL transformation in the sequence has the purpose of extracting the default values for Simulink Block Properties and transforming them into the format that conforms to the Source metamodel. These default values will then be propagated to the Blocks where they are not explicitly defined in the next transformation.

The rule shown in Listing 2 shows the transformation for a simple property from the Simulink Structure model to the Simulink Defaults model. The test for if the Property should be transformed includes a call to a helper to determine if the Property is a primitive type and a determination if the Property parent is a Block Parameters section and not a Block from the main model. In the target object section the rule uses a call to a helper to determine the Property's type.

```
rule SimpleProperty2Property {
from
sp: Simulink!Property
        (sp.isSimpleProperty and
        sp.refImmediateComposite().oclIsTypeO
        f(Simulink!BlockParams))
to
p: Source!Property(
Name <- sp.Name,
Type <- sp.getPropertyType,
Value <- sp.Value.Value.toString()
)
}
```

Listing 2. Simulink Defaults Rule

4.3.1.4. Block Information

The second ATL transformation transforms the Blocks contained in a Simulink model into a model format conforming to the Source metamodel. The original Simulink Structure model elements are combined with the values from the Simulink Defaults model using the following algorithm:

- If the Property exists in the Model
    - Use the value in the Model
- Otherwise, if the Property is in the Defaults
    - Add the Property to the Block using the Default value

4.3.1.5. Blocks and Ports

The next step in the transformation process is to explicitly define the Ports on each Block in the model. Simulink uses a method of implicitly understanding the ports on each Block in the System. The connections between Blocks are defined as Line elements with source and destination strings representing a port on a Block, which is implicitly understood to exist. To facilitate the transformation to a SysML model with explicitly defined FlowPorts, we produce this intermediate model with the Ports explicitly defined from the information contained in the Properties of the Block.

4.3.1.6. Source

Once we have the Blocks and Ports defined in a System, we can use the Line information from the Simulink Structure model to add the information representing the interconnections between Blocks. This transformation is executed using ATL.

The helper displayed in Listing 3 shows the how the port for a source or destination for a Line in Simulink is found using the string that is the value of a src or dst attribute in the Line element. The string for these attributes has the form of Block ID followed by a "#" symbol followed by the direction of the port followed by the ":" symbol followed by the port number.

```
helper def: findPort(pName : String):
Source!Port =
Source!Port.allInstancesFrom('src')->
select(p | p.refImmediateComposite().ID +
        '#' + p.Direction + ':' + p.Num =
        pName)->any(sp | true);
```

Listing 3. SourceHelper

4.3.1.7. Source for SysML

Once we had a representation for the model that conforms to our Source metamodel that contains all pertinent information from the original Simulink model, we still had two more challenges to overcome:

1. Structure of an S-Function Block
    a. Represented in the user interface as a single Block with Inports and Outports
    b. Represented in the XML model as a System with Inport Blocks for each Inport, Outport Blocks for each Outport, S-Function Block, Demux Block, and Terminator Block
2. Connecting System ports to the SubSystem port Blocks

The first challenge was encountered and handled through the Team Strategy portion of our process development. The second challenge was handled when our process was expanded to Robot Soccer.

4.3.1.8. Interim SysML

Once we were able to produce models that contained the pertinent information from Simulink that also had the structure similar to our desired SysML model, we could develop a transformation using ATL to create the SysML model to be viewed in Eclipse.

The Eclipse-Papyrus structure for a SysML model requires three model files:

1. UML
    a. The model with the information contained in the SysML model

including Blocks, Properties, FlowPorts, Associations, and Connections

2. Notation
    a. The model that indicates how the various visual elements representing the model elements in the UML file should be arranged within the diagrams
3. DI
    a. The model that indicates which diagrams from the Notation file should be available and visible when viewing the model in Eclipse

In order to maintain the references between the models, this transformation required that we generate the UML and Notation files together. For the purposes of replicability and reproducibility, the file that is most important is the UML file. However, for the purposes of viewing and verifying the generated model, the Notation file and the DI file discussed in subsection 4.3.1.9 are necessary.

4.3.1.9. SysML

Once we have the UML and Notation files for a given SysML model generated from our process, we can run a simple XSLT transformation to produce the necessary DI file. Our contention is that it is desirable to be able to view all diagrams created in the Interim SysML transformation. Therefore, our transformation in this step searches the Notation file for any diagrams and adds them to the list to be displayed.

4.4. SysML to Simulink

After the transformation from Simulink to SysML, we also established a "round trip" transformation process providing the transformation from SysML to Simulink. This "reverse transformation" provides to reproduce the original Simulink model from the SysML model that was produced by the "forward transformation". Therefore, if developers need to modify certain elements in SysML model, then they are able to produce Simulink model of this updated model, as well.

The reverse transformation process is given in Figure 6. As it can be realized, the reverse transformation does not have as many models as the forward transformation, because some of the intermediate models that are created during the forward transformation are not necessary to create during the reverse transformation. For instance, the two intermediate models, "Block Info" and "Blocks & Ports", in the forward transformation have not been used in reverse transformation. Instead, the "Source" intermediate model has been directly transformed into the "Simulink Structure" intermediate model.

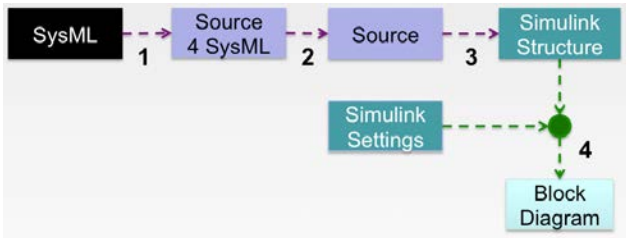

Figure 6. SysML to Simulink Transformation Process

Our reverse transformation cannot be evaluated as a stand-alone transformation, because we handle an intermediate model which has already been created during the forward transformation, namely Simulink Settings. So, in order to complete a reverse transformation, an already completed forward transformation is required at this point.

4.4.1. Intermediate models

All intermediate models in the reverse transformation contain the same information as their respoective intermediate models in the forward transformation. Transformation rules in the reverse transformations are for the most part simply the reverse operations from the forward transformation. The first, second and third steps of the reverse transformation are implemented using ATL and run respectively.

At the end of the first three steps, an intermediate model, "Simulink Structure", which conforms to the metamodel defined in Section 4.2.1 is generated. Then, this model and "Simulink Settings" are handled together to generate the final model: Block Diagram. This last step is implemented using XSLT, since it does not require metamodels but requires merging files and many syntactical modifications.

5. Future Work

In addition to the progress we have already made in this endeavor to develop a model transformation process for use in a scheme to enhance scientific experimentation, we have much work ahead. In this section we detail some of our future plans concerning this work.

5.1. Simulink Expansion

The next steps in developing our process of using model transformation for generating SysML models from Simulink source models and executable Simulink models from source SysML models is to expand both processes to encompass the remainder of Simulink Block types. In this way we can be reasonably assured that any well-formed model developed in Simulink or SysML will be able to be used as an input to the proper transformation process.

Following this, our plan is to expand our transformation processes further to be able to transform Simulink S-Function MATLAB source code into SysML Activity Diagrams and vice versa. We believe there is much promise in developing the process initially from SysML to Simulink. Then we can use the lessons learned from this process to assist in the development of the process from Simulink to SysML.

Additionally, our plan regarding the SysML to Simulink transformation will be to modify it to be able to handle a transformation without a previously executed Simulink to SysML transformation on the same model. Currently our process depends on the existence of the intermediate model defined in Section 4.3.1.1 in order to produce a valid, executable Simulink model from an existing SysML model. For our purposes, we shall require the development of a means to produce Simulink models when this intermediate model is not present.

5.2. Experiment Management

Inherently, every simulation is the part of an experiment and the management is a very important part of the experimentation. In terms of management, all data related to simulation models and other supporting artifacts are needed to be able to be processed easily. Therefore, those performing experiments are able to create a feedback loop to improve their processes. In this respect, two important aspects of MDE can be seen as facilitating factors. First, a formal PIM model is used to capture the details of experiments. It can be a useful method of extracting valuable features of the model. For example, in terms of simulation models, parameters may not have any relationship with each other; changing the value of one does not impact another. On the other hand, some of them are factors and some of them responses in terms of experimentation. Thus, an experimentation-aware metamodel helps to capture that information. Secondly, the aforementioned transformation approach enhances the ability to create a traceability to a lower level models. Thus, scientists can focus on analyzing results and tuning parameters rather than writing “glue codes” [17]. This paradigm shift facilitates the whole experimentation process.

6. Conclusions

We have briefly shown the development of a Model Driven Engineering process for utilizing model transformation to enhance the credibility of scientific experimentation by assisting with the ability to replicate and reproduce the simulation experiments that have become increasingly used in scientific research. When fully complete, this two-fold transformation process will greatly assist with the ability to manipulate aspects of simulation models while not requiring specific knowledge of the original model’s platform. By allowing other researchers to access the Platform Independent representation of the model used in an experiment, the experiment can be reproduced and the scientific community can have a great amount of confidence in the results obtained from such experiments.

7. References

[1] J. P. Mesirov: “Accessible reproducible research” Science, 327(5964), pp. 415–416, 2010.

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

[3] A. Morin, J. Urban, P. D. Adams, I. Foster, A. Salli, D. Baker: “Shining light into black boxes” Science, 336, pp. 159-160, 2012.

[4] S. Fomel and G. Hennenfent: “Reproducible computational experiments using scons” Acoustics, speech and signal processing, Vol. 4, pp. IV–1257, 2007.

[5] T. Mens and P. V. Gorp: “A Taxonomy of Model Transformation” Electronic Notes in Theoretical Computer Science (ENTCS), Vol. 152, pp. 125-142, March 2006.

[6] K. Ma and B. Yang: “A Hybrid Model Transformation Approach Based on J2EE Platform” 2010 Second International Workshop on Education Technology and Computer Science (ETCS), pp. 161-164, March 2010.

[7] J. S. Cuadrado: “Towards a Family of Model Transformation Languages” Theory and Practice of Model Transformations, Springer Berlin Heidelberg, 2012.

[8] D. Vagelaar: “Composition Techniques for Rule-based Model Transformation Languages” Theory and Practice of Model Transformations, Springer Berlin Heidelberg, 2012.

[9] The Eclipse Foundation: Atlas transformation language. http://www.eclipse.org/atl/, 2015. [Online; accessed 23-April-2015].

[10] J. Bézivin, F. Jouault, and D. Touzet: “An introduction to the atlas model management architecture” Rapport de recherche, (05.01), 2005.

[11] The Eclipse Foundation: Eclipse modeling framework. http://www.eclipse.org/modeling/emf/,2015. [Online; accessed 23-April-2015].

[12] The Eclipse Foundation: Atl tutorials. http://www.eclipse.org/atl/documentation/basicExamples_Patterns/, 2015. [Online; accessed 23-April-2015].

[13] The Eclipse Foundation: Eclipse modeling framework tools - ecore tools. http://www.eclipse.org/modeling/ emft/?project=ecoretools, 2015. [Online; accessed 23-April-2015].

[14] World Wide Web Consortium (W3C): Extensible stylesheet language transformations. http://www.w3.org/TR/xslt, 16-November-1999. [Online; accessed 23-April-2015].

[15] Mathworks MATLAB Central: Robot Soccer. http://www.mathworks.com/matlabcentral/fileexchange/28196, 21-October-2010. [Online; accessed 23-April-2015].

[16] Mathworks Documentation: To Workspace Block. http://www.mathworks.com/help/simulink/slref/toworkspace.html, 2015. [Online; accessed 23-April-2015].

[17] S. Cohen-Boulakia and U. Leser, “Search, adapt, and reuse: the future of scientific workflows” ACM SIGMOD Record 40 Vol. 2, pp. 6–16, 2011.

8. Author Biographies

JOSEPH LEDETis a PhD candidate in Computer Science and Software Engineering at Auburn University. He received his B.S. degrees in Electrical Engineering and Computer Engineering from Louisiana State University and his M.S. in Computer Engineering from the University of Louisiana at Lafayette. He also has over 8 years of experience as a software developer for private industry and government contractors. His email address is jwl0008@auburn.edu.

SEMA ÇAM is a PhD student in theDepartment of Computer Engineering at Middle East Technical University. She received her B.S. degree in Computer Engineering from Izmir Institute of Technology, and M.S. degree in Software Engineering from Middle East Technical University.She also has 5 years of experience in software development industry. Her email address is e177478@metu.edu.tr

B. KAAN GÖRÜR is a PhDcandidate in Department of Computer Engineering in Hacettepe University. He received his B.S. degrees in Department of Computer Engineering in Hacettepe University. He has been working as a research assistant in Middle East Technical University – Turkish Armed Forces Modeling and Simulation Research and Development Center for 5 years. His email address is kgorur@metu.edu.tr

ORÇUN DAYIBAŞ is a Ph.D. candidate (in Computer Engineering at METU) with particular interest in model-driven engineering and software architectures. He has more than 10 years of professional experience in developing software-intensive systems (both as a founder of his own start-up and as an employee of a leading defense contractor in Turkey). He holds a M.Sc. degree in Comp. Eng. from METU and B.Sc. degree in Comp. Eng. from Hacettepe University. His email address is odayibas@gmail.com.

HALİT OĞUZTÜZÜN is an associate professor at the Department of Computer Engineering, Middle East Technical University (METU), Ankara, Turkey. He got his PhD in Computer Science from University of Iowa, Iowa City, IA in 1992. His current research interests are model-driven engineering and distributed simulation. He participates in the activities of MODSIMMER (Modeling and Simulation R&D Center) at METU. His email address is oguztuzn@ceng.metu.edu.tr

LEVENT YILMAZis Professor of Computer Science and Software Engineering at Auburn University with a joint appointment in Industrial and Systems Engineering. He holds M.S. and Ph.D. degrees in Computer Science from Virginia Tech. He is the founding organizer and General Chair of the Agent-Directed Simulation Conference series and is currently serving as the Editor-in-Chief of Simulation: Transactions of the SCS. His email address is yilmaz@auburn.edu.

ALICE SMITH is the W. Allen and Martha Reed Professor of Industrial and Systems Engineering at Auburn University with a joint appointment in Computer Science and Software Engineering. She has authored papers with over 2,000 ISI Web of Science citations and has been a principal investigator on projects with funding totaling over $6 million. She is an area editor of INFORMS Journal on Computing and Computers & Operations Research and an associated editor of IEEE Transactions on Evolutionary Computation. Her email address is smithae@auburn.edu.