AToMPM: A Web-based Modeling Environment

Eugene Syriani[1], Hans Vangheluwe[2,3], Raphael Mannadiar[3], Conner Hansen[1], Simon Van Mierlo[2], and Huseyin Ergin[1]

[1] University of Alabama, U.S.A.
[2] University of Antwerp, Belgium
[3] McGill University, Canada

Abstract. We introduce AToMPM, an open-source framework for designing domain-specific modeling environments, performing model transformations, manipulating and managing models. It runs completely over the web, making it independent from any operating system, platform, or device it may execute on. AToMPM offers an online collaborative experience for modeling. Its unique architecture makes the framework flexible and completely customizable, given that AToMPM is modeled by itself, and external applications can be easily integrated. Demo: `https://www.youtube.com/watch?v=iBbdpmpwn6M`

1 Introduction

Today, several tools and technologies allow modelers to develop domain-specific modeling languages (DSMLs) and manipulate models, such as AToM^3 [1], DSLTools [2], EMF [3], GME [4], MetaEdit+ [5], and VMTS [6], just to name a few. They often require an *installation* of the tool and depend on external artifacts such as operating system (VMTS), middleware platform (DSL Tools), or virtual machine (EMF, AToM^3). Furthermore, the degree of *collaboration* between developers and their models is often restricted to the version controlled repository used by the tool (SVN, CVS, or GitHub). Nevertheless, one of the reasons for the success and popularity of EMF is its *plugin* framework that allows tremendous extensions of its core which gave birth to a suite of numerous modeling and transformation tools, such as ATL [7], Epsilon [8], XText [9], VIATRA2 [10]. However, the development of these extensions requires expertise in EMF, and its Java API.

In this paper, we introduce AToMPM (A Tool for Multi-Paradigm Modeling) [11], the successor of AToM^3. AToMPM is an open-source framework for designing DSML environments, performing model transformations, manipulating and managing models. It runs completely over the web, making it independent from any operating system, platform, or device it may execute on. AToMPM follows the philosophy of modeling everything explicitly, at the right level of abstraction(s), using the most appropriate formalism(s) and process(es), being completely modeled by itself (*i.e.*, bootstrapped).

2 Highlight of Features

AToMPM is a modern, versatile and theoretically sound multi-paradigm modeling environment. It is a tool for modeling any and every part of a system at the most appropriate level(s) of abstraction, using the most appropriate formalism(s). For instance, AToMPM is explicitly modeled using a combination of UML Class diagrams and Statecharts. The tool offers unique features given its web-based nature that we outline below.

2.1 Modeling in the Cloud

AToMPM runs entirely online and requires no client-side installation. It allows one to model in the cloud, although it is possible to install the server on-premise. The client consists only of an SVG-compliant web browser. Models can be downloaded locally if desired.

2.2 Graphical Modeling vs. Textual Commands

AToMPM is primarily a graphical modeling environment. On the concrete syntax side, all model elements displayed are SVG elements. All static and dynamic manipulation offered by SVG are fully supported (*e.g.,* translation, scaling, rotation, transparency, Bézier curves). Model manipulation, such as CRUD operations, can be performed with mouse clicks and movements as in traditional modeling environments. In AToMPM, it is also possible to write textual commands to perform the same manipulations, designed with a modeled textual DSL. Textual commands can be more useful for more advanced users, especially for creating/deleting/updating multiple elements at a time.

2.3 Synthesis of Domain-Specific Modeling Environments

As in any modeling tool, the specification and synthesis of DSMLs is central functionality of AToMPM. The default modeling language for defining meta-models is a simplification of UML class diagrams. Nevertheless, any modeling language can be used to define meta-models, as long as there is a transformation defined mapping that meta-modeling language to the default language. Synthesis of a DSML editing environment for that new DSML is automatically supported. Static constraints are expressed on top of the meta-model in a textual DSL for constraints or using the Javascript API directly.

Multiple concrete syntaxes can be assigned to the same abstract syntax. This allows different users to have their own representation of the same model. A graphical concrete syntax is defined by mapping a group of SVG elements to each meta-model element, both class-like and association-like elements. Since everything is modeled explicitly in AToMPM, the concrete syntax is itself specified by a DSML representing geometric shapes.

2.4 Model Transformation, Code Generation and Debugging

Model transformations are also explicitly modeled in AToMPM. All model transformations are based on T-Core [12], a minimal collection of model transformation operators. This has the advantage of executing automatically any custom-built rule-based transformation language. Given the input and output meta-models of a transformation, a language for designing domain-specific rules is automatically generated [13]. Rules are defined with a left-hand side, right-hand side and negative application conditions, as in graph transformations. The patterns inside the rules are represented using the concrete

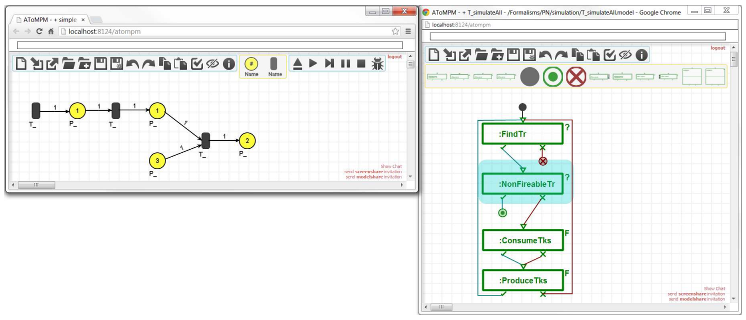

Fig. 1. Model transformation debugging.

syntax of the input and output languages, adapted to patterns. The default scheduling language of a model transformation is MoTif [14]. Nevertheless, any modeling language can be used to define the transformation language, as long as there is a higher-order transformation defined mapping that language to the T-Core language. Execution and debugging of that new model transformation language is automatically supported.

There are two modes of execution of model transformations in AToMPM. In *release* mode, the input model displayed on the canvas is sent to the server, transformed completely, and the new resulting model is then displayed on the canvas. In *debug* mode, the transformation is animated on the client's canvas. The execution can be continuous, or step-by-step. Breakpoints can be specified at the control flow level and a new window pops up to inspect the current state of the model and transformation as depicted in Fig. 1. The model transformation execution is deployed as a plugin in AToMPM, which may run on a dedicated server. The transformation engine is based on Himesis [15], a Python implementation.

2.5 Process Modeling

Any process enforced by AToMPM is also modeled explicitly by a UML activity diagram-like DSML. For example, the process for defining a meta-model, then assigning the concrete syntax and, finally, generating the modeling environment is a process model. A chain of model transformations is also modeled with this language. The activities in a process model can be automatic (like a transformation or plugin) or manual. In the latter case, a window pops up to let the user manipulate a model.

2.6 Collaborative Modeling

One motivation behind an online development environment is the ability to collaborate and share modeling artifacts among users. Multiple users can be logged in simultaneously with each having their own view of the models. Models and model elements visibility is controlled with permission roles that an administrator can assign and specify. Currently, AToMPM supports two types of real-time distributed collaboration mechanisms. *Screenshare* allows two or more clients to share the exact same canvas: any change made to a model (abstract or concrete syntax) is replicated on all observing clients. *Modelshare* only shares the abstract syntax of a model between clients. Each client has its own view of the same model, using its own concrete syntax. For example, if a client updates the value of an attribute of a model element, this may change the displayed text next to the represented element on that client's model while changing the color of that element on another client's representation.

3 Modeled Plugin Framework Architecture

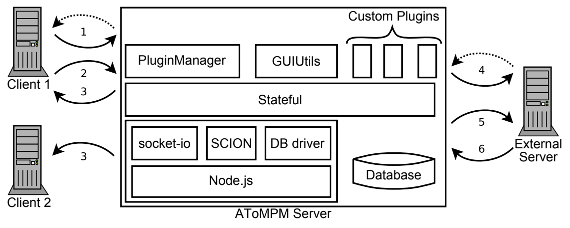

Fig. 2. High level architecture of AToMPM.

AToMPM is a completely bootstrapped environment. The client-server architecture depicted in Fig. 2 allows multiple web-browsers (client) to send and receive HTTP requests with AToMPM (server) following the MVC pattern. AToMPM is a Node.js server driven by SCXML statecharts executed within the SCION engine [16]. It consists of a minimal kernel "Stateful" deprived from any modeling-specific functionality. It is a plugin framework where every toolbar and functionality is modeled explicitly as a plugin.

On startup, Stateful loads the kernel statechart, which handles all requests to and responses from the server. The kernel also brings in the PluginManager plugin during initialization, which is enough to then drive the loading of various custom plugins, such as AToMPM. The PluginManager design provides register/unregister hooks so that any plugin can easily register and load or unregister and unload itself within the Plugin-Manager package system. Stateful allows for plugins to register their own statecharts to respond to specific server endpoints, which allows for plugins to easily extend the behavior of Stateful. Within this framework, if some degree of communication is needed with an external server then that behavior can be added, made accessible, and begin responding in Stateful. This provides the ability for a backend to be written in a completely different language than Javascript, while still being able to fully interact with Stateful using only a very small amount of mostly generatable code. While this design does allow for easy extension, it also relies heavily on plugins being properly designed which may make it less robust from the client's experience.

There are client side variants for the kernel and PluginManager components as well. When a user first attempts to access Stateful, these components are sent over which then load either the default plugins or plugins that are specific to that user's configuration if logged in. Any client side components that are available on the server can be loaded by a client at any time, allowing for plugins to be able to be dynamically loaded and unloaded.

As depicted in Fig. 2, there four asynchronous communication paths in this framework. (1) A call-back mechanism allows a client to send requests to the server and receive results. (2) A headless mode allows the client to send requests to be processed in batch. (3) A broadcast mechanism allows the server to notify multiple client observers. (4) AToMPM can communicate with external servers in the same way as it does for clients. The latter is very useful when interoperating external tools (such as transformation engine, model verification) with AToMPM. These communications are all modeled with plugins.

Acknowledgments

We would like to thank all the members of MSDL at McGill University and the University of Antwerp for their work and contributions to the built-in formalisms available in AToMPM: Sadaf Mustafiz, Levi Lucio, Maris Jukss, Johachim Denil, and Bart Meyers.

References

1. de Lara, J., Vangheluwe, H.: AToM^3: A Tool for Multi-formalism and Meta-Modelling. In: FSE'02. Volume 2306 of LNCS., Springer-Verlag (2002) 174–188
2. Cook, S., Jones, G., Kent, S., Wills, A.C.: Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley Professional (2007)
3. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. 2nd edn. Addison Wesley Professional (2008)
4. Lédeczi, Á., Bakay, A., Maroti, M., Völgyesi, P., Nordstrom, G., Sprinkle, J., Karsai, G.: Composing Domain-Specific Design Environments. IEEE Computer **34**(11) (2001) 44–51
5. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+ A fully configurable multi-user and multi-tool CASE and CAME environment. In: Conference on Advanced Information Systems Engineering. Volume 1080 of LNCS., Crete, Springer-Verlag (may 1996) 1–21
6. Levendovszky, T., Lengyel, L., Mezei, G., Charaf, H.: A Systematic Approach to Metamodeling Environments and Model Transformation Systems in VMTS. In: GraBaTs'05. Volume 127 of ENTCS., Amsterdam, Elsevier (mar 2005) 65–75
7. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming **72**(1-2) (jun 2008) 31–39
8. Kolovos, D., Paige, R., Polack, F.: The Epsilon Object Language (EOL). In: ECMDA-FA'06. Volume 4066 of LNCS., Springer (2006) 128–142
9. Eysholdt, M., Behrens, H.: Xtext: implement your language faster than the quick and dirty way. In: OOPSLA'10, ACM (2010) 307–309
10. Varró, D., Balogh, A.: The model transformation language of the VIATRA2 framework. Science of Computer Programming **68**(3) (2007) 214–234
11. `https://acom.cs.mcgill.ca/trac/AToMPM/`
12. Syriani, E., Vangheluwe, H., LaShomb, B.: T-Core: A Framework for Custom-built Transformation Languages. Journal on Software and Systems Modeling (jul 2013)
13. Syriani, E., Gray, J., Vangheluwe, H.: Modeling a Model Transformation Language. In: Domain Engineering: Product Lines, Conceptual Models, and Languages. Springer (2012)
14. Syriani, E., Vangheluwe, H.: A Modular Timed Model Transformation Language. Journal on Software and Systems Modeling **12**(2) (jun 2011) 387–414
15. Syriani, E., Vangheluwe, H.: Performance Analysis of Himesis. Technical Report SOCS-TR-2010.8, McGill University, School of Computer Science (aug 2010)
16. `https://github.com/jbeard4/SCION`