A Cloud Architecture for an Extensible Multi-Paradigm Modeling Environment

Jonathan Corley[1] and Eugene Syriani[2]

[1] University of Alabama, U.S.A.
[2] Université de Montréal, Canada

Abstract. We present the cloud architecture of AToMPM, an open-source framework for designing domain-specific modeling environments, performing model transformations, manipulating and managing models. AToMPM's cloud-based architecture makes it independent from any operating system, platform, or device it may execute on as well as any visualization of the modeling structures (*e.g.,* textual or graphical). AToMPM offers an online collaborative experience for modeling. Its unique architecture makes the framework scalable while also providing a flexible and completely customizable modeling environment.

1 Introduction

In the perspective of making software technologies scalable, current software-as-a-service providers offer hardware and software virtualization in the cloud [1]. Model-driven technologies can also benefit from this trend to make current model-driven engineering tools more scalable and appealing to industry needs. Model management tools, such as EMF [2] and GME [3], often rely on software and—for performance reasons—hardware dependencies that prevent domain-experts from using them.

Recently, several new systems have emerged embracing a new paradigm of in-browser software development environments including WebGME [3], Clooca [4], and Eclipse WebIDE [5]. These systems provide a fully featured development environment available through modern web-browsers. WebGME and Clooca both provide modeling environments. However, it is not clear whether any of them supports multiple views with multiple users working jointly on the same models.

In this paper, we describe the cloud architecture of AToMPM (A Tool for Multi-Paradigm Modeling) [4]. AToMPM is an open-source framework for designing DSML environments, performing model transformations, manipulating and managing models. The cloud architecture makes AToMPM independent from any operating system, platform, or device it may execute on. AToMPM's cloud architecture provides a scalable back-end for potentially many heterogeneous front-end environments including text-based command-line applications, graphical browser-based applications, and everything in between. 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).

[3] `http://webgme.org/`
[4] `http://www.clooca.com/`
[5] `http://www.eclipse.org/orion/`

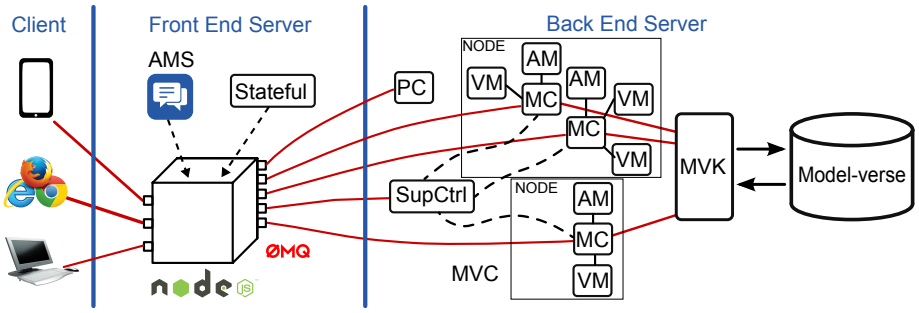

Fig. 1. High level architecture of AToMPM.

2 Cloud Architecture of AToMPM

An overview of the AToMPM architecture is illustrated in Fig. 1. The basic components are the client, the front-end web server, the Model View Controller (MVC) structure, and the Model-verse Kernel (MvK) along with the model-verse.

2.1 Client

An in-browser client UI is packaged with the front-end server for AToMPM as a default client user interface (illustrated in Fig. 2). The in-browser client UI provides a generic environment consisting of a canvas and the ability to render toolbars. This generic client interface enables basic modeling actions: editing models, generating modeling languages, concrete syntax definitions, model transformation execution. The interface is completely extensible. Providing a new toolbar only requires creating a model that conforms to the toolbar metamodel. The in-browser client UI provides a rich, extensible environment for modeling that is available on any platform supporting a modern web browser. Also, the in-browser client does not require any download or installation process to be used on the client machine. Once the AToMPM server is online, end-users may connect to the in-browser UI by navigating to `http://<server-address>/ATOMPM`.

However, the cloud architecture is not limited to supporting only the default in-browser client. It can support anything from a terminal-based client to a smartphone application client. A client UI system must be able to communicate with the front-end server via sockets. The client would then send and receive communication using a change log structure for messages. Using this protocol the client can interact with existing models, build new models, and any other action that could be performed by the default browser-based client. This includes accessing toolbars and other plug-ins as used by the in-browser client. Additionally, since the cloud architecture of AToMPM interacts with all client UI systems via the socket communication using the same message protocol, any variety of client UI systems may be connected concurrently and will receive the same live updates regardless of which client UI that generated the update. For example, Bob could be accessing a smartphone application and would receive an update made by Alice who is using the in-browser interface.

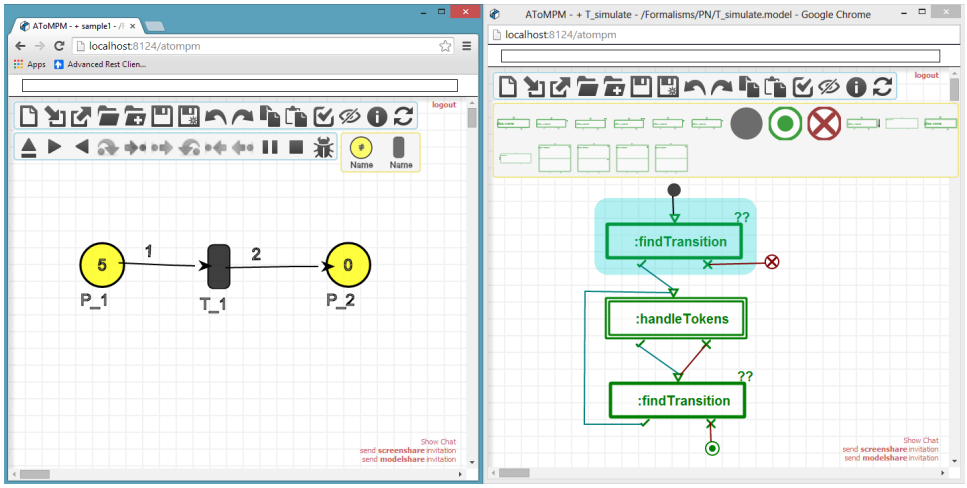

Fig. 2. Default in-browser client for AToMPM.

2.2 Front-End Web Server

All client UI systems connect directly to the front-end web server. The server manages all socket communications to and from clients. The front-end web server is built using node.js and maintains a caching system to keep track of all active sockets. Anytime a message is received the system utilizes the caches to identify where the message needs to be routed. There are a few exceptions where the web server may respond directly to a communication. For example, when a client closes their connection to a given model controller (discussed in Section 2.3) the server will remove the client from the appropriate cache and directly responds to the client. Additionally, the web server provides facilities for a basic messaging system to enable communication between end-users. However, the majority of messages are handled by the MVC structure which directly interfaces with the modeling kernel. The front-end web server maintains connections with end-users, and (as described in Section 2.1) the web server is agnostic of the type of client UI. The web server also has two subcomponents: AMS, a messaging system, and Stateful, a Statecharts execution environment used to execute the behavior of plug-ins that is modeled in Statecharts. These two components are included in the front-end server because they are both client concerns related to the run-time environment and not the modeling system.

The primary advantage of the front-end web server is decoupling the back-end server structure from the clients and providing a lightweight scalable routing system to manage incoming messages. This way, the back-end system needs to consider only a single direct user and does not need to worry about routing responses to an arbitrary number of clients. For example, a model representing a single large system (*e.g.,* a modern car) may be used by any number of users concurrently and a change would need to be routed to all of the users viewing the model. The front-end web server handles the communication overhead of sending messages to all relevant clients and the back-end

system only needs to handle the message appropriately and send a single response to the front-end web server. Thus, the front-end web server increases the scalability of the overall cloud architecture of AToMPM.

2.3 MVC

The back-end server consists of two components: the MVC structure and the modeling kernel. The MVC structure provides the controllers that handle the majority of client requests. The MVC consists of three primary controller types: supercontroller, model controller, and package controller. The supercontroller manages model controllers. The model controller manages a given model and all views of that model. The package controller manages requests independent of a specific model. The supercontroller and package controller register themselves with the front-end server, a process that creates an open channel for subsequent socket communication with these two controllers. However, the model controllers are only instantiated as needed. If the system has thousands of models and only one is being actively used, then only one model controller will be active. Thus we vastly reduce the constraints on our server architecture when few resources are being used. As new models become active (via clients sending requests for these models), the front-end server will request from the supercontroller to open new model controllers to handle these requests. The supercontroller then acts as a load manager for the model controllers. It ensures only a single model controller is active for each model. The model controller ensures consistency between its related models. The front-end server then makes sure all messages from the model controller are routed to all users subscribed to the relevant model. Thus, users maintain a consistent and up-to-date copy of the model, and the definitive copy of the model is kept on the server ensuring consistency.

The model controller separates models into two categories: abstract models and view models. An abstract model is the traditional notion of a model. An abstract model is any entity that is modeled in the system and conforms to some metamodel. A view model is any model that conforms to the view metamodel. A view model must reference another model as its abstract model. The view contains a set of elements from the abstract model that are included in the view. Thus, we enable developers to work with only a portion of a model. The view also contains a concrete syntax mapping for every element included in the view. Thus, separate modelers may have their own representation, arrangement, and sizing for the elements of their view, but consistency of the underlying model elements is maintained for all views. Finally, a view has a list of associated tools (such as toolbars). The tools list includes the concrete syntax for the abstract model. This enables developers to use separate concrete syntaxes for the same model.

2.4 Modeling Kernel

The second component in the back-end server is our modeling kernel, MvK. It contains and controls access to the model-verse (a repository of all models within the system) and handles executing all actions taken on any model in the model-verse. MvK offers an API to manage storing, retrieving, creating, altering, and removing models. MvK also

maintains consistency with conformance checks and verification of constraints. Finally, MvK contains components used to execute model transformations, action code, and simulations. MvK is a pure modeling system where all models are treated uniformly. The unique aspects of certain models are handled by the outside systems that use these models. For example, the concrete syntax model is merely a model conforming to the concrete syntax metamodel when considered by MvK. However, a client UI uses a concrete syntax model when generating the graphical components that will visually represent elements of the current model.

3 Conclusion

We described the cloud-based architecture of AToMPM (A Tool for Multi-Paradigm Modeling). The cloud architecture makes AToMPM independent from any operating system, platform, or device it may execute on. AToMPM's cloud architecture provides a scalable back-end for potentially many heterogeneous front-end environments including text-based command-line applications, graphical web-based applications, and everything in between. The architecture consists of several key components:

- A front-end server which manages all direct communication with client systems.
- An MVC structure composed of three basic controllers.
  - Supercontroller which manages model controllers. Ensuring only a single model controller exists for each active model.
  - Model Controller which manages a given abstract model and all related view models. Ensuring consistency of the various models for all users.
  - Package controller which manages requests not related to a specific model.
- The MvK which manages all access to the model-verse (the collection of all models).

Together these components build a scalable and flexible modeling system supporting development of heterogeneous, and potentially globally distributed, modeling environments.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A View of Cloud Computing. Communications of the ACM **53**(4) (apr 2010) 50–58
2. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. 2nd edn. Addison Wesley Professional (2008)
3. 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
4. Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Van Mierlo, S., Ergin, H.: AToMPM: A Web-based Modeling Environment. In: MODELS'13: Invited Talks, Demos, Posters, and ACM SRC. Volume 1115., Miami FL, USA, CEUR-WS.org (2013)