ALMOsT-Trace: A Web Based Embeddable Tracing Tool for ALMOsT.js

Rocio Nahime Torres and Carlo Bernaschina(✉)

Dipartimento di Elettronica, Informazione e Bioingegneria,
Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milano, Italy
{rocionahime.torres,carlo.bernaschina}@polimi.it
http://www.deib.polimi.it

Abstract. Model Driven Development (MDD) requires model-to-model and/or model-to-text transformations to produce application code from high level descriptions. Debugging and evaluating such transformations is in itself a complex task; complexity which can be mitigated through the usage of advanced developer tools. We demonstrate ALMOsT-Trace, a plug-in for ALMOsT.js, which allows developers to debug and analyze their model transformations from within their applications. In the demo, attendees will be able to experiment with ALMOsT-Trace by evaluating it in IFMLEdit.org, an online tool for the rapid prototyping of web and mobile applications, and by means of examples that can be customized by the attendees themself.

Keywords: Agile development · Model-driven development · Computer aided software engineering

1 Introduction

Model Driven Development (MDD) is the branch of software engineering that advocates the use of *models*, i.e., abstract representations of a system, and of *model transformations* as key ingredients of software development [9]. With MDD, developers use a general purpose (e.g. UML [1]) or domain specific (e.g., IFML [10]) modeling language to portrait the essential aspects of a system, under one or more perspectives, and use (or build) suitable chains of *transformations* to progressively refine the models into executable code.

Online platforms like [4] can help reduce the complexity of models and model transformations management, execution and validation.

Traceability of artifacts during the evolution of a software project is a key aspect for regression testing and monitoring. In model transformations the ability to trace models between transformation steps [2,3] enables advanced analysis and rapid error detection.

In a previous work [5] we have presented ALMOsT.js [1] (AgiLe MOdel Transformations for JavaScript), an agile, in-browser framework for the rapid prototyping of MDD transformations, which lowers the technical skills required for

[1] www.npmjs.com/package/almost/.

DOI: 10.1007/978-3-319-60131-1_42

Web and Mobile developers to start be proficient with modeling and code generation. The design philosophy of ALMOsT.js stemmed from 6 requirements (#1 *No installation*, #2 *No new language*, #3 *Fast start-up*, #4 *Parallel development*, #5 *Customized output*, #6 *Customized generation*), which could be summarized in one: keep it simple. It allows the developer to rapidly bootstrap and evolve an MDD project, by storing models in a flexible JSON based format and by defining model transformation rules by means of simple JavaScript functions. The following code shows a template example to create a rule.

```
// Create a rule
createRule(
    // Condition function
    function (element, model) { return /*condition*/;},
    // Action function
    function (element, model) { return /*result*/; }
);

// Create a transformer
var transform = createTransformer(rules, 'm2m');

// Execute transformer;
var output_model = transform(input_model);
```

We present ALMOsT-Trace, a plug-in for ALMOsT.js, able to trace elements and relations during the execution of model transformations. The novelty of ALMOsT-Trace is the ability to trace model during transformations in-browser, by easily integrating the tracing tool inside the final MDD environment.

2 Framework

Following the *keep it simple* and plug-in based nature of ALMOsT.js the introduction of ALMOsT-Trace in a project is completely optional and as for other plug-ins [5] does not require any changes to the existing system.

ALMOsT-Trace is based on two components:

1. A drop-in replacement for the **createTransformer** creator function which enhances the created transformer with tracing abilities, as shown below.

```
// Traced transformer
var transformer = createTracedTransformer(rules, 'm2m');

// Events
transformer.on('begin', function (model) { ... });
transformer.on('skipped', function (rule, input) { ... });
transformer.on('executed', function (rule, input, output) { ... });
transformer.on('end', function (result) { ... });
```

Tracing aware rules can add custom tracing data to the final report, via a **trace** function that is injected as last parameter in each one of them.

2. A **dashboard** which listens for events on a transformer and allows the developer to analyze the traces recorded during every execution. It can be easily integrated inside any web based tool already using ALMOsT.js and enabled on demand.

```
// Create the dashboard
var dashboard = createDashboard(transformer);
// Show the dashboard
dashboard.show();
```

At the end of the execution of each rule the tuple $< rule, input, output >$ is stored. This lookup table allows the dashboard to lookup all the tuples related to each *rule*, *input* element and relation or *output* object components.

While forward tuples lookups (from *rules* and *input* to the *output*) are naïve to implement, backward lookups (from *output* to *rules* and *input*) are not. If during the aggregation phase two or more objects are merged together it is not possible to identify them from their root reference, it is instead required to search for outputs in the tuples collection which contain a particular subpart that survived unaltered till the end of the aggregation.

While reference-based types (*Object*, *Array*, ...) are easy to identify, it is enough to compare the reference, primitive types (*number*, *string*, ...) cannot be identified by means of a simple comparison, it is not possible to distinguish a number from another which contains the same value.

Enhanced rules map all the primitive values to their *Object* wrapper counterpart (*Number*, *String*, ...) the wrappers will generate the same result during aggregation, but will be uniquely identifiable. The enhanced transformer is responsible to restore each one of the wrapper objects to their original form before returning the result to the caller, preserving though the drop-in nature of ALMOsT-Trace.

By analyzing the stored tuples in the lookup table the dashboard of ALMOsT-Trace generates four different reports:

1. **Rule execution statistics**
   All the defined rules are listed and for each one of them it is possible to know: (a) the number of evaluations (b) the number of executions (c) the enabling elements or relations (d) the output related to the execution against each enabling element.
2. **Input Model statistics**
   All the elements and relations defined in the input model are listed and for each of them it is possible to know: (a) the number of enabled rules (b) the output related to the execution of the enabled rules against the element or relation itself.
3. **Output Model statistics**
   For *model to model* and *model to text* transformations all the elements, relations, folders or files defined in the output are listed and for each one of

them it is possible to know: (a) the rules which generated them (b) the input against which the rules have been executed to generate the element, relation, folder or file itself.

4. **Output Object statistics**
   For *custom* transformations (transformations with an output format defined by the developer) the final output JSON Object is presented in an expandable tree-view. For each *Object* attribute or *Array* item it is possible to know if they were generated directly from a rule or if they are the result of an aggregation step. For each attribute or item that can be identified in one of the rules outputs the corresponding $< rule, input >$ pair is shown.

3 Conclusions

This demo presents ALMOsT-Trace, a transformations tracing plug-in for ALMOsT.js [5]. We show how it can be integrated inside an existing project and how it can be used to analyze the model transformations execution. Attendees will be able to experiment with ALMOsT-Trace by seeing its impact inside IFMLEdit.org [6–8], a web based model driven tool, and by means of examples that the can be customized by the attendees themself.

The future work will focus on the experimentation and further assessment of ALMOsT-Trace in order to validate its impact on the development cycle, in both industry and academia.

References

1. UML unified modeling language. www.uml.org/. Accessed 17 Mar 2017
2. MeTAGeM-trace: improving trace generation in model transformation by leveraging the role of transformation models. Sci. Comput. Program. **98**, Part 1, 3–27 (2015). Fifth Issue of Experimental Software and Toolkits (EST): A Special Issue on Academics Modelling with Eclipse (ACME 2012)
3. Aranega, V., Mottu, J.M., Etien, A., Dekeyser, J.L.: Using Trace to Situate Errors in Model Transformations
4. Basciani, F., Di Rocco, J., Di Ruscio, D., Di Salle, A., Iovino, L., Pierantonio, A.: MDEForge: an extensible web-based modeling platform. In: CloudMDE@ MoDELS
5. Bernaschina, C.: ALMOsT.js: an agile model to model and model to text transformation framework. In: International Conference on Web Engineering (ICWE) 2017
6. Bernaschina, C., Brambilla, M., Koga, T., Mauri, A., Umuhoza, E.: Integrating modeling languages and web logs for enhanced user behavior analytics. In: International Conference on World Wide Web (WWW) 2017
7. Bernaschina, C., Brambilla, M., Mauri, A., Umuhoza, E.: A big data analysis framework for model-based web user behavior analytics. In: International Conference on Web Engineering (ICWE) 2017

8. Bernaschina, C., Comai, S., Fraternali, P.: IFMLEdit.org: model driven rapid prototyping of mobile apps. In: International Conference on Mobile Software Engineering and Systems (MOBILESoft) 2017
9. Kleppe, A., Warmer, J., Bast, W.: MDA Explained - The Model Driven Architecture: Practice and Promise. Addison Wesley object technology series. Addison-Wesley, Reading (2003)
10. OMG: Interaction flow modeling language (ifml), version 1.0. (2015). http://www.omg.org/spec/IFML/1.0/