txtUML[⋆]

Gergely Dévai, Tibor Gregorics, Melinda Tóth, Domonkos Asztalos, Dávid János Németh, Gábor Ferenc Kovács, Boldizsár Németh, Zoltán Gera, András Dobreff, Balázs Gregorics, András Nagy, Martin Budai, Zsolt Kulik, and Kristóf Kanyó

Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary,
deva@elte.hu

Abstract. The name *txtUML* stands for textual, executable, translatable UML. It is an Eclipse-based tool built on top of JDT, Xtext/Xbase and Papyrus UML.
The tool is designed for textual model editing. This makes storage, version control, compare and merge processes, editing and searching easier and more efficient. The tool supports two textual syntaxes for modeling: the *standalone syntax*, which is designed to be clean and short, and alternatively, the *txtUML Java API*, which can be used to define models as standard Java programs.
The tool supports the generation of graphical UML diagrams from the textual descriptions: class and state machine diagrams. The layout of the diagrams can be controlled by a simple textual diagram layout language.
Models can be seamlessly integrated into Java programs: they can be executed and debugged. Generated state machine diagrams can be animated during model execution to further enhance comprehension of model dynamics.
Compatibility with other tools is ensured by generating standard UML models in EMF-UML2 format. This representation is the input for our model compiler, which generates C++ code.

Keywords: executable modeling, textual modeling, UML

1 Introduction

This demonstration presents the framework *txtUML*, which stands for *textual, executable, translatable UML*. It is a tool for software development according to the executable UML paradigm [8].

The project was started in 2014, and the development is done by the *Model-Driven Development Research Group* of Eötvös Loránd University, Budapest. It is open source [17] on GitHub under the *Eclipse Public License* [2]. The development is supported by Ericsson Hungary and the tool is currently used in pilot projects at the company. The txtUML framework is distributed as a set of Eclipse plugins. Installation instructions and detailed user documentation is available on the project website [16].

[⋆] This work is supported by Ericsson Hungary.

2 Similar Tools

This subsection presents open source UML tools with model execution capabilities or textual syntaxes.

BridgePoint [13] uses an early fork of UML extended with a proprietary action language. It uses graphical modeling for class and state modeling, and textual modeling on the action code level. It is now maintained by a company named *OneFact* and has been open sourced [4]. In contrast to BridgePoint, txtUML supports textual modeling for all layers of the modeling language and visualizes the model via generated diagrams. Furthermore, txtUML is compatible with the latest UML standard.

Papyrus UML [14] is an Eclipse UML framework that aims at providing customizable graphical editor support for the full UML standard. Papyrus has an extension, called *Moka* [10], which contains an interpreter that executes *fUML* [12] models and animates diagrams. This interpreter is based on the reference implementation of fUML and is not designed for high performance. By contrast, txtUML's model execution is based on on-the-fly translation to Java to provide better performance.

Alf [11] is an OMG standard defining textual syntax for fUML. An Alf editor is being integrated into Papyrus [15] in order to provide a textual model editing alternative. As state machines are not part of fUML and Alf, this modeling layer is currently not handled by the solution.

Moliz [9] is a testing and debugging framework for fUML activities. It defines a test specification language and extends the fUML reference implementation with debugging and tracing capabilities. The execution traces are used to decide if a given test case passes or fails. This project also uses graphical model editors and the aforementioned fUML reference implementation.

Umple [5], *eTrice* [3] and the *Papyrus UML-RT* [1] are modeling environments for both graphical and textual model editing. Unlike txtUML, these tools lack an abstract action language: Umple allows modeling code mixed in Java, PHP, C++ and Ruby and its code generator emits code in these languages, while eTrice allows action code in Java or C written in the models as *string literals* which are propagated to the generated code. UML-RT uses the same strategy with C++. None of the approaches allows execution and debugging on the model level. The diagram generation methodology is also different: The referenced tools use autolayout algorithms, while txtUML allows the user to define the layout of diagrams using a concise DSL.

3 Textual Syntaxes

Textual modeling has many advantages over editing models in graphics [7]. Most importantly, model storage, version control, compare and merge functionalities can be provided by off-the-shelf, reliable tools. Text editors are more advanced and stable compared to currently available open source graphical model editors, and it is often faster to input text than graphics. The txtUML tool provides two textual syntaxes for modeling:

– *X*txtUML: Standalone syntax designed to be short and clean. It is supported by an Xtext-based editor [18].
– *J*txtUML: In this case, the model is a standard Java program, which uses the *txtUML Java API*. Therefore this is an embedded language in Java.

The standalone syntax is easier to use, especially for users not familiar with Java. On the other hand, the Java syntax can be used in any Java development environment so that a project using this syntax will not depend on a new editor and a new language. The two syntactic variants are closely related: In fact, XtxtUML models are on-the-fly translated to JtxtUML models while they are edited in Eclipse. This is the basis for model execution.

```
signal ButtonPress;        class ButtonPress extends Signal{}

class Machine {            class Machine extends ModelClass {
  initial Init;              class Init extends Initial {}
  state Off;                 class Off extends State {}
  state On;                  class On extends State {}

  transition Initialize {    @From(Init.class)
    from Init;               @To(Off.class)
    to Off;                  class Initialize extends
  }                              Transition {}

  transition SwitchOn {      @From(Off.class)
    from Off;                @To(On.class)
    to On;                   @Trigger(ButtonPress.class)
    trigger ButtonPress;     class SwitchOn extends
  }                              Transition {}

  transition SwitchOff {     @From(On.class)
    from On;                 @To(Off.class)
    to Off;                  @Trigger(ButtonPress.class)
    trigger ButtonPress;     class SwitchOff extends
  }                              Transition {}
}                          }
```

Fig. 1. Two syntactic variants of the txtUML language
(Left: Standalone syntax, Right: Embedded language in Java)

Figure 1 shows an example model both in standalone and Java syntax. The model defines one signal (*ButtonPress*) which will trigger transitions in the state machine of the class *Machine*. Inside the class, its states and transitions are defined. For example, the *ButtonPress* signal triggers a transition from the *Off* state to the *On* state.

The design of the two language variants follows a pattern: Kinds of the model elements are shown by keywords (*signal*, *class*, *transition*) in the standalone syntax, while the Java version uses inheritance from *Signal*, *ModelClass* and *Transition*. These classes are provided by the txtUML Java API. Properties of the transitions are expressed by Java annotations (e.g. *@From*) in Java, while attribute-like syntax with keywords (e.g. *from*) is used in the standalone version.

The txtUML language covers a subset of UML. We summarize the supported elements below:

– *Component modeling*: Interfaces containing signals; ports; connectors
– *Class modeling*: Classes with attributes and operations; simple binary associations; compositions; (single) inheritance
– *State modeling*: Simple and composite states; transitions triggered by signals; guards; choice states
– *Behavior modeling*: Action code can be written in operations of classes, entry and exit actions of states and effects of transitions. Supported base types are *int*, *double*, *boolean* and *String* with the usual arithmetic and logic expressions, variables and assignment. Control structures (loops, branches), attribute access and operation calls are supported. UML-specific actions: creation and deletion of objects; linkage and unlinkage via associations and connectors; reading links; sending signals; accessing signal data in entries, exits and effects.

4 Diagram Generation

While textual modeling is practical for *editing* models, graphical diagrams are better for *understanding* them. For this reason txtUML can generate standard UML diagrams from the models in the format of the Papyrus UML framework [14]. As of release *0.4.1*, *class diagrams* and *state machine diagrams* are supported. We are working on *composite structure diagrams* at the time of writing this paper.

txtUML allows users to influence the layout of the generated diagrams by textual *diagram descriptions*. A simple DSL, realized by Java annotations, is defined for this purpose. The most important layout statements are the following:

– *@Above*, *@Below*, *@Left* and *@Right* define that two given diagram elements are placed next to each other vertically or horizontally.
– *@North*, *@East*, *@South* and *@West* place the two given diagram elements in two different half-plains of the diagram, but gives no further constrains on the distance of the elements.
– *@TopMost*, *@BottomMost*, *@EastMost* and *@WestMost* place the given elements on the corresponding side of the diagram.
– *@Row*, *@Column* and *@Diamond* can layout multiple elements at once in the given shape.
– *@Show* declares that the given elements have to appear on the diagram, but gives no further constraint on their placement.

```
class MicrowaveOven;
class Lamp;
class Magnetron;

composition LampsOfOven {
  container MicrowaveOven oven;
  1..* Lamp lamp;
}

composition MagnetronOfOven {
  container MicrowaveOven oven;
  1 Magnetron magnetron;
}
```

Fig. 2. Example model with classes and composite associations

```
class MicrowaveDiagram
  extends Diagram {
 @TopMost(
  MicrowaveOven.class)
 @Row({
  Lamp.class,
  Magnetron.class})
 class MicrowaveLayout
  extends Layout {}
}
```

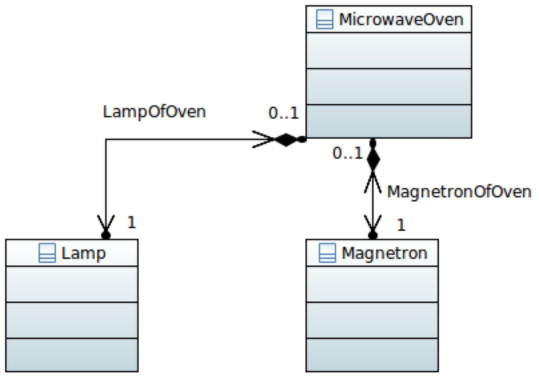

Fig. 3. Diagram layout description and generated diagram of the model in Figure 2

In order to produce a diagram layout according to the given constraints, txtUML applies different graph algorithms and heuristics. We refer the reader to [6] for the details of this solution.

The left side of Figure 3 shows a diagram layout description for the class model in Figure 2. It requests the *MicrowaveOven* class to be on the top of the diagram and that the other two classes form a row. This setup is realized by the generated diagram in the right of the Figure.

5 Execution and Model Compilation

No matter which syntactic variant of txtUML is selected to implement a model, it can be seamlessly integrated into Java programs. The txtUML Java API can be used to create objects of classes in the model, link them via the associations (if needed) and send signals to them. Such a Java *main* function, executing the (completed) model of Figure 2, is shown in Figure 4.

```
public static void main(String[] args) {
  ModelExecutor.Settings.setExecutorLog(true);
  MicrowaveOven oven = Action.create(MicrowaveOven.class);
  Action.start(oven);
  Action.send(new DoorOpened(), oven);
  Action.send(new DoorClosed(), oven);
  Action.send(new Start(), oven);
  Action.send(new DoorOpened(), oven);
  ModelExecutor.shutdown();
}
```

Fig. 4. Executing a model from Java code

The usual Java debugging functionalities of Eclipse (breakpoints, stepping, viewing values of variables etc.) are available for txtUML model code as well, also for the standalone syntax. While the model is executed or debugged, it is possible to animate the generated state machine diagrams, as shown in Figure 5.

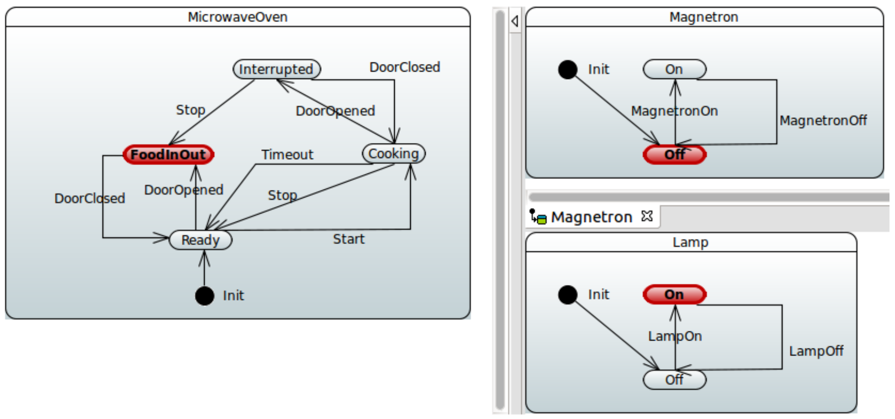

Fig. 5. Animated state machine diagrams

If the target platform is not compatible with Java, then model compilers have to be used. As the first step of the transformation, we turn txtUML models into EMF-UML2 models, which is the de facto standard format of UML in the Eclipse environment. This can be used as platform-independent input for code generators.

The txtUML project includes a work-in-progress code generator for C++. It can generate single and multithreaded C++ programs, and users can configure which classes of the model should run in which thread.

References

1. Mixed textual/graphical modelling in Papyrus-RT. `https://wiki.polarsys.org/images/0/04/TextualModelingForModelingDays.v2.pdf`
2. Eclipse Public License - v 1.0. `https://www.eclipse.org/legal/epl-v10.html`
3. eTrice. `http://www.eclipse.org/etrice/`
4. Executable Translatable UML Open Source Editor. `https://www.xtuml.org`
5. Forward, A., Badreddin, O., Lethbridge, T.C.: Umple: Towards Combining Model Driven with Prototype Driven System Development. In: Rapid System Prototyping (RSP), 2010 21st IEEE International Symposium on. pp. 1–7. IEEE (2010)
6. Gregorics, B., Gregorics, T., Kovács, G.F., Dobreff, A., Dévai, G.: Textual Diagram Layout Language and Visualization Algorithm. In: Model Driven Engineering Languages and Systems (MODELS), 2015 ACM/IEEE 18th International Conference on. pp. 196–205. IEEE (2015)
7. Grönniger, H., Krahn, H., Rumpe, B., Schindler, M., Völkel, S.: Textbased Modeling. In: 4th International Workshop on Software Language Engineering (2007)
8. Mellor, S.J., Balcer, M., Jacoboson, I.: Executable UML: A foundation for model-driven architectures. Addison-Wesley Longman Publishing Co., Inc. (2002)
9. Mijatov, S., Langer, P., Mayerhofer, T., Kappel, G.: A Framework for Testing UML Activities Based on fUML. In: Proceedings of the 10th International Workshop on Model Driven Engineering, Verification and Validation (MoDeVVa) co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013). pp. 1–10 (2013)
10. Moka. `http://wiki.eclipse.org/Papyrus/UserGuide/ModelExecution`
11. Object Management Group: Action Language for Foundational UML (ALF), standard, version 1.0.1. `http://www.omg.org/spec/ALF/` (2013)
12. Object Management Group: Semantics of a Foundational Subset for Executable UML Models (fUML), standard, version 1.1. `http://www.omg.org/spec/FUML/1.1/` (2013)
13. OneFact: BridgePoint xtUML tool. `http://onefact.net`
14. Papyrus. `https://www.eclipse.org/papyrus/`
15. Seidewitz, E., Tatibouet, J.: Tool Paper: Combining Alf and UML in Modeling Tools – An Example with Papyrus –. In: OCL 2015–15th International Workshop on OCL and Textual Modeling: Tools and Textual Model Transformations Workshop Proceedings. p. 105 (2015)
16. txtUML: Textual Executable Translatable UML. `http://txtuml.inf.elte.hu/`
17. txtUML: Textual Executable Translatable UML – Open source repository. `https://github.com/ELTE-Soft/txtUML`
18. Xtext. `http://www.eclipse.org/Xtext/`