Round-Trip Modeler for Java using Spoon
# Table of contents

Introduction .......................................................................................................................... 3

I. Background ......................................................................................................................... 4
   A. AST ................................................................................................................................. 4
   B. UML Modelers ............................................................................................................... 4

II. Project ................................................................................................................................. 5
   A. Requirements .................................................................................................................. 5
      1. GUI ............................................................................................................................ 5
      2. Features ..................................................................................................................... 6
   B. Technologies .................................................................................................................. 6
      1. Spoon ......................................................................................................................... 6
      2. Round Trip Modeler ................................................................................................. 6
   C. Implementation .............................................................................................................. 7
      1. Spoon Readers ............................................................................................................ 7
      2. Spoon Writers .......................................................................................................... 7

III. Analysis ............................................................................................................................. 8
   A. Results ............................................................................................................................ 8
   B. Limitations ....................................................................................................................... 9
   C. Perspectives .................................................................................................................. 9

IV. Conclusion ......................................................................................................................... 11

V. References .......................................................................................................................... 12
Introduction

UML class diagram is a graphical way to represent object-oriented software projects, which are mainly composed of classes, interfaces, etc. Each component in a Java project can be depicted as an UML element, so that it makes it easier to visualize the global structure of the project. Currently, even if a diagram describes the implementation of a particular project, it is not necessarily linked to the source code. In case of changes, we can find differences and even contradictions between the code and the diagram.

Round Trip Modeling offers to always synchronize the UML diagram and the source code, by impacting one as soon as the other is changed. It means that we can regenerate the UML representation to always see the real project, but we can also add Java elements directly in the UML editor and generate empty templates in the source code.

Our goal is to make a little application which shows that Round Trip Modeling is possible. We make a proof of concept to demonstrate the feasibility of our idea using Spoon.
I. Background

A. AST

An Abstract Syntax Tree (AST) is a tree representation of the source code, where each node is an element of the language. AST is a hierarchical view where the parent node contains its children. For example, a Class element can have several Fields and Methods. Each Method can have several Parameters. AST are particularly used by parsers and compilers, as they give an abstraction of the source code that can be easily used for syntax analysis, and then compiling.

A Java AST is already written and used by the Java compiler. However, this kind of AST is optimized for compiling, but the level of abstraction can be improved. Some other ASTs exist, like JDT for Eclipse or SrcML, which provide a better abstraction.

B. UML Modelers

As UML is widely used by software engineers, there are many existing UML modelers. They can be used for different goals, such as reverse engineering: a UML diagram can be generated from an existing project to make it easier to understand its mechanism; code generation: a UML diagram is created and then empty templates are generated in the source code; modeling: before starting the development phase, a UML model can be suggested, but it’s just an image that can’t generate source code; or round-trip modeling: only few modelers enable the two first points, which means synchronization of UML diagram and source code.

Most IDEs include at least one UML modeler as a plugin, which sometimes enables Round Trip Modeling, but this part is still in development and the modelers are based on the AST we’ve just seen, with all the difficulties we’ve expressed. Moreover, they are actually barely used throughout the development phase.
II. Project

A.Requirements

The purpose of our project is to implement critical features of the Round Trip Modeling. Each change in the source code should be displayed in the UML modeler, and each change in UML should be transmitted to the source code and generate code templates (which always compile).

1. GUI

We want to display our project in a UML modeler but we wish to keep all the existing and useful functionalities we can find in IDE. We also would like the UML modeler to be independent from any IDE. That's why the project source folder path is given as a parameter to the UML modeler, and not written directly in our application.

So we would like our POC to look like this:
2. Features

As we don’t have the time to implement all the functionalities we have in mind, we decided to focus on the fundamental features that demonstrate the principle of Round Trip Modeling. Our POC hence implements the CRUD (Create, Read, Update, Delete) operations, for compilation units (class, interface, enumeration), fields and methods.

Once the project is loaded, each existing compilation unit is depicted in the UML modeler. From the main screen, it is possible to add a class, an interface or an enumeration. Once a compilation unit is selected, it can be deleted but only if it’s not used elsewhere in the source code, in order to keep the source code safe from compiling errors. The compilation unit’s name can also be changed.

In the UML representation of a compilation unit, fields and methods are displayed in the UML way. Once a compilation unit is selected, its properties are shown in the content panel, like the list of fields and methods. Each one can be deleted, but only if it’s unused, for the same reasons as said before. It’s also possible to change the name and the type (or return type for a method).

To get the expected results, we lean on the following technologies.

B. Technologies

1. Spoon

Spoon is an Inria tool which enables to analyze and transform Java source code. It provides a new complete Java metamodel, including each element (classes, interfaces, methods...). It is far easier to understand than the Java Compiler AST. In addition, Spoon also exposes a full API to manipulate this AST easily, that’s why we decided to directly use this AST in our POC.

In order to get the Round Trip Modeling to work, we have to read data from Spoon AST and display it in the UML diagram, and write or update existing data into AST when a user changes the UML Diagram.

2. Round Trip Modeler

As we said before, there are many existing UML modelers. However, we decided not to use them because only few are open sources and written in Java language. Furthermore, it may be a time-consuming activity to learn how they work. Without taking into account the fact that it may be more complicated to use Spoon with them; or they might already use another AST. Finally, they often implement many advanced features that we can't reuse due to the fixed limits of this project.

That’s why we decided to create a new UML modeler, named “Round Trip Modeler”, which directly uses Spoon AST as model, and displays each element as simply as possible to show the entire UML model. We followed the UML way to depict the classes, interfaces and enum classes, including fields and methods.
C. Implementation

1. Spoon Readers

In order to read correctly data in the analyzed projects, we implemented some Spoon processors able to find each kind of compilation unit and each reference to a field or a method, in order to know if it’s called in at least one constructor or method. In this case, we can’t remove or rename it without creating compiling errors.

2. Spoon Writers

Instead of injecting some source code directly in the project, we add data in the AST and make Spoon generate the whole project again. Furthermore, the Spoon AST throws an Error if it can’t accept the changes in AST for any reason (if two fields have the same name for example). In this case, the change is just cancelled and doesn’t affect the source code.
III. Analysis

A. Results

Our POC is working and demonstrates the Round Trip Modeling principles.
This project is able to run several refactorings on the analyzed project:

<table>
<thead>
<tr>
<th>Element affected</th>
<th>Refactoring</th>
<th>Effective ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation Unit</td>
<td>Create a new C.U.</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Remove an existing C.U.</td>
<td>Yes, but generated again later</td>
</tr>
<tr>
<td></td>
<td>Refactor the C.U. name</td>
<td>Yes</td>
</tr>
<tr>
<td>Field</td>
<td>Create a new Field</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Remove an existing Field</td>
<td>Only for unused fields</td>
</tr>
<tr>
<td></td>
<td>Refactor field name</td>
<td>Only for unused fields</td>
</tr>
<tr>
<td></td>
<td>Refactor field type</td>
<td>Only for unused fields</td>
</tr>
<tr>
<td>Method</td>
<td>Create a new Method</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Remove an existing Method</td>
<td>Only for unused methods</td>
</tr>
<tr>
<td></td>
<td>Refactor method name</td>
<td>Only for unused methods</td>
</tr>
<tr>
<td></td>
<td>Refactor method return type</td>
<td>Only for unused methods</td>
</tr>
</tbody>
</table>

Summary of implemented features

Our application shows well that Round Trip Modeling is effective as the two views are always synchronized. But you can't do everything with the current version, as there are some functionalities that could be useful but are not yet implemented.

**B. Limitations**

We found out that renaming a compilation unit, a field or a method which is used elsewhere in the source code would add compiling errors, just like deleting these elements. It would have been difficult to change all the references, so we decided to check if it can be safely refactored, and to forbid the action if not.

Another limitation comes from Spoon API. The way to create classes and other elements isn't the same, so we had to use a little trick to generate interfaces, fields, etc. It's working but it's a bit tricky. It may be corrected in a future version of the API.

So there are some limitations in our application, but more important are the improvements we could add.

**C. Perspectives**

First, as the project doesn’t cover the entire Java syntax, it can’t display all project concepts. It could be a great enhancement to add notions of hierarchy, packages, etc, to have as much information as possible.
Currently, when user gives a type (for a field, or the return type of a method), he has to set its qualified name. It would be better to just give the simple name (ex: String) and then the application could find the qualified name (ex: java.lang.String), so that it could be easier for the user.

Even if primitive types in source code are understood by Spoon AST, they are not accepted as a type when user wants to add a new field or method from the UML modeler.

Moreover, at the moment, user can only add methods with no parameters from the UML modeler. The best would be to have a variable number of parameters for a method.

Though we only implement the simplest refactorings, we believe that more advanced ones can be developed in this project, by just adding other Spoon processors. We could imagine enabling field refactoring even if it's used elsewhere, by changing every reference in the source code. But it would still be impossible to refactor the type without checking it's not called somewhere else. It’s quite the same for methods.

Another important improvement is to just insert specific code when adding Java elements in the UML modeler, instead of always generate the entire project again. For example, if we want to add a field in a class from the UML modeler, we could just add the code for this field in the class, and not generate the project entirely. So the insertion could be much more efficient.

Furthermore, it could be wonderful to be able to detect modifications in the source code and refresh the UML diagram automatically. It would prevent user from forgetting to reload the code and using the UML modeler, which can provoke errors.

Finally, for user convenience, the modeler part could be improved by adding other features, like reference seeker or displaying only a part of the entire UML diagram. It could also be more ergonomic and easier on the eye.
IV. Conclusion

Our goal was to make a proof of concept to show the feasibility of Round Trip Modeling. As we can see in our application, an update in the source code changes the UML diagram, and a modification in the UML modeler generates the associated code in the project. This way, the two parts are always synchronized, which means Round Trip Modeling is effective. Spoon is a very efficient tool, thanks to its good abstraction of the Java syntax, its generation ability and its API which is easy to use and understand.

We didn't have the time to implement all the features we wanted to develop in our Round Trip Modeler, so we chose the fundamental ones, which show the most how Round Trip Modeling works (adding compilation units, removing fields, etc). But there are a lot of interesting functionalities that could be added and that could make our application more useful throughout the development phase.

Maybe one day, Round Trip Modeling may be widely used by the community during the development phase as it provides new opportunities in software engineering. For example, it could include better templates generator, with intelligent code insertions.


V. References

- code du projet : https://github.com/JonathanGeoffroy/RoundTripModeler