User Manual

For

Blended Modeling Framework
(PSS Use Case)

Table of Contents

1. Introduction

In line with BUMBLE project, our main objective is to develop a generic solution that should support seamless runtime synchronization between graphical and textual syntaxes for multiple Domain Specific Modeling Languages (DSMLs) / Standards. To achieve this, we propose a comprehensive blended modeling framework as shown in **Figure 1**. Particularly, it takes meta-models (in ECORE) and textual DSL / grammar as an input and subsequently, generates graphical and textual notations automatically through *BUMBLE-M2T-Meta-Model* and *BUMBLE-Language-Analyzer* components respectively. The generated graphical and textual notations are saved in XML files and serve as an input for mapping & synchronization activities as shown in **Figure 1**. The *BUMBLE-Mapping-Editor* component is developed to perform mapping between graphical and textual syntaxes. Subsequently, the mapping rules are saved in XML and serve as an input for synchronizer. For runtime synchronization, mapping rules in XML are implemented as an EBNF grammar through *BUMBLE-EBNF-Generator* component as shown in **Figure 1**. Finally, *Blended Modeling Editor* prototype supporting seamless synchronization between graphical and textual syntaxes is generated through Sirius platform.

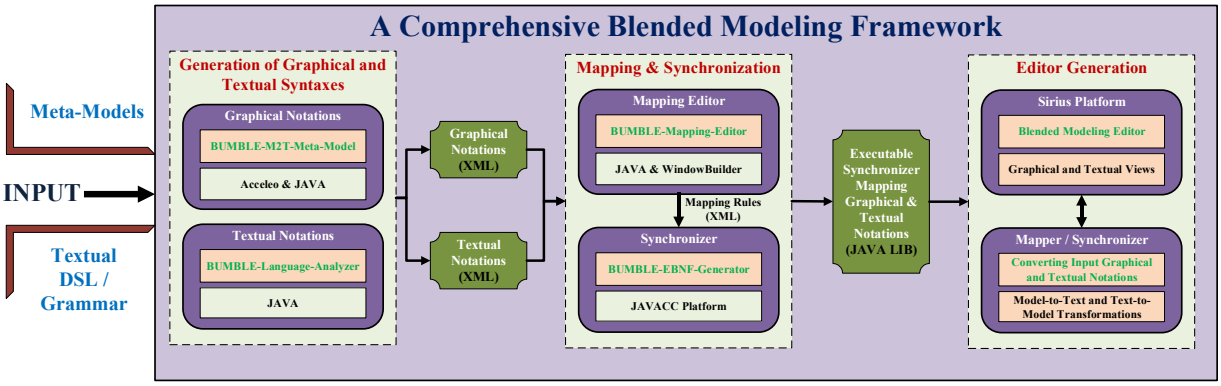

Figure 1: The High Level Architecture of Proposed Framework

In this manual, we demonstrate the usage / working of each component with the help of Portable test and Stimulus Standard (PSS) use case. It is pertinent to mention that all components of proposed framework (along with installation guide and detailed user manuals) are publicly available at GitHub i.e. https://github.com/blended-modeling/PSS or https://github.com/MDH-BUMBLE/PSS/

2. Components Description and PSS Use Case

2.1 PSS.DSL.MetaModel

We utilize PSS use case for the demonstration of different components. In this regard, the modeling environment and meta-model of PSS is not available to the best of our knowledge. Therefore, a basic PSS meta-model is proposed here under PSS.DSL.MetaModel project. It can be seen from **Figure 2** that the main PSS concepts like actions, objects, resources etc. and their semantics are included in the meta-model. This meta-model is given as an input to *BUMBLE-M2T-Meta-Model* component for the generation of graphical notations. Furthermore, it also facilities the editor generation process in Sirius Platform.

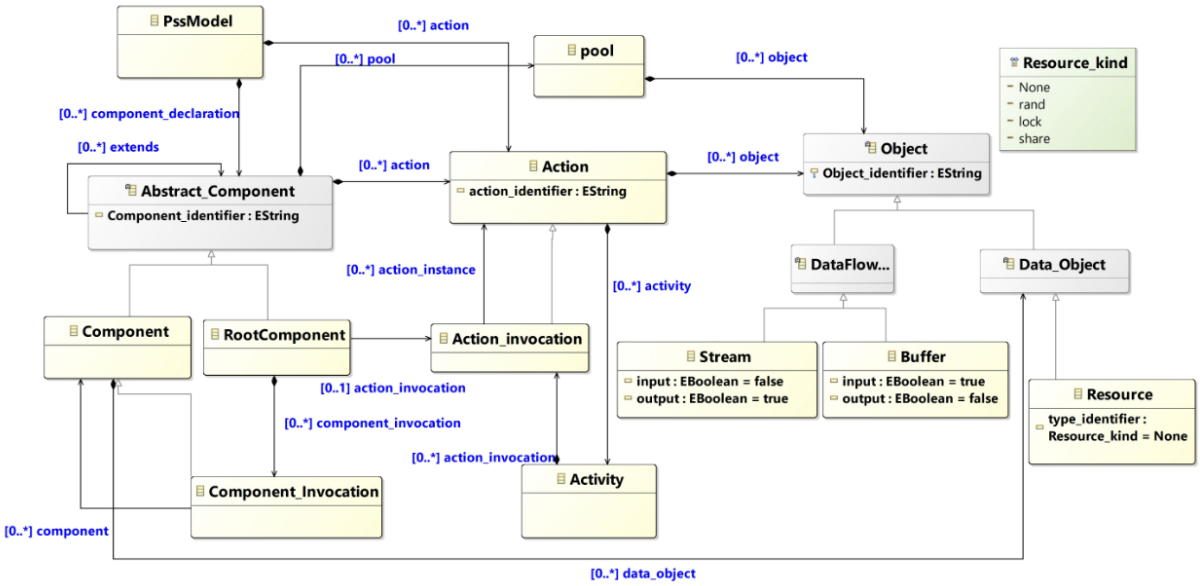

Figure 2: PSS meta-model Comprises Major Concepts

2.2 BUMBLE-M2T-Meta-Model

This component takes meta-model as an input and generate required graphical syntaxes. Subsequently, it stores all information in XML file. The address of meta-model and target folder can be specified through run configurations as shown in **Figure 3**.

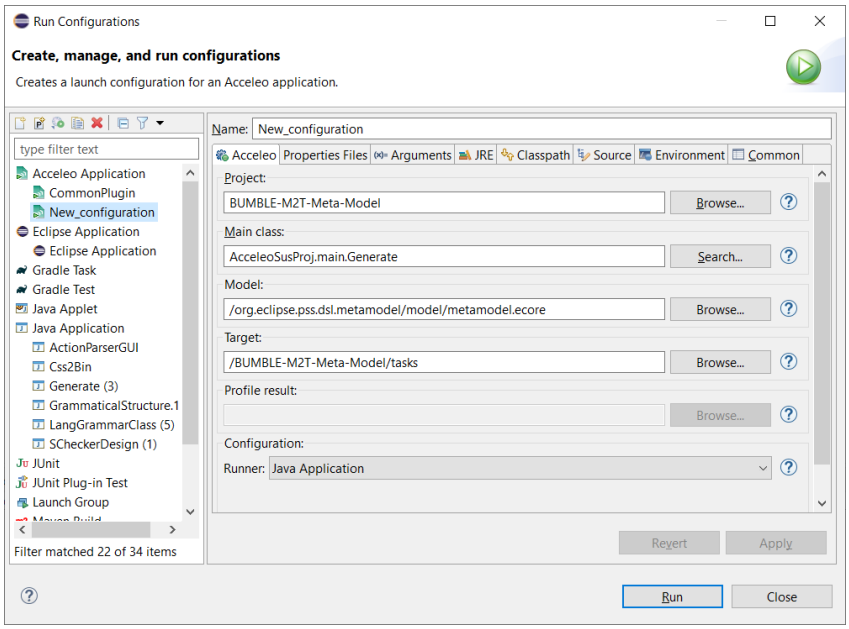

Figure 3: Specification of Meta-Model and Target Folder

The implementation of BUMBLE-M2T-Meta-Model component is achieved in Acceleo and Java as shown in **Figure 4**.

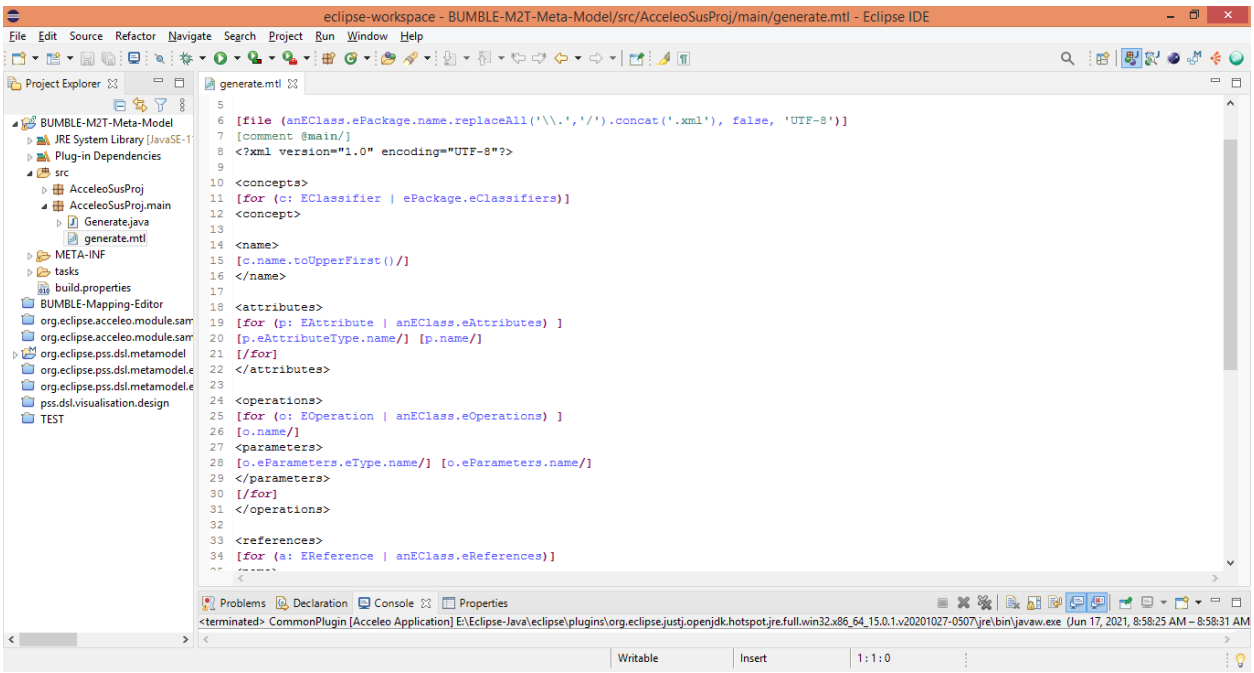

Figure 4: implementation of BUMBLE-M2T-Meta-Model component

The PSS meta-model (**Figure 2**) is given as an input and generated XML file is shown in **Figure 5**.

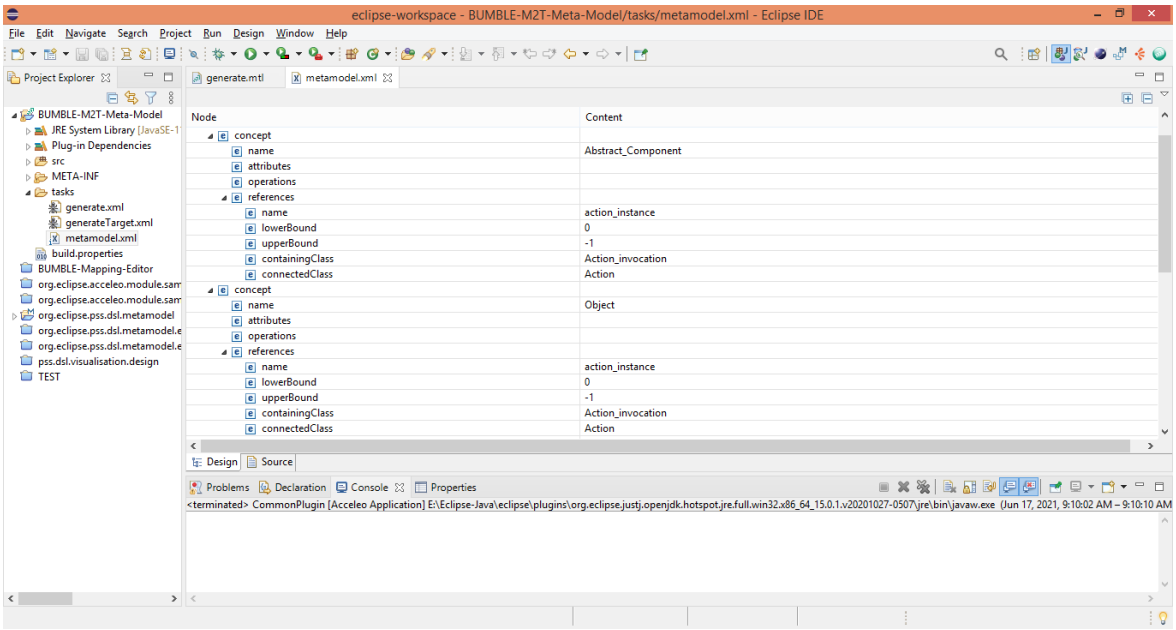

Figure 5: Graphical Syntaxes Saved in XML File

2.3 BUMBLE-Language-Analyzer

This component is responsible to effectively generate the textual notations from given DSL grammar / syntax samples. Particularly, it learns patterns and tokens of given DSL / grammar and subsequently, generates textual syntaxes. This component is developed in java and its interface is shown in **Figure 6**. A sample PSS file is given as an input and grammar analyzer generate textual syntaxes (tokens) through parse button. Subsequently, grammar analyzer is capable to save all textual syntaxes in XML file as shown in **Figure 6**. Similarly, stakeholders / users can generate textual syntaxes for any language.

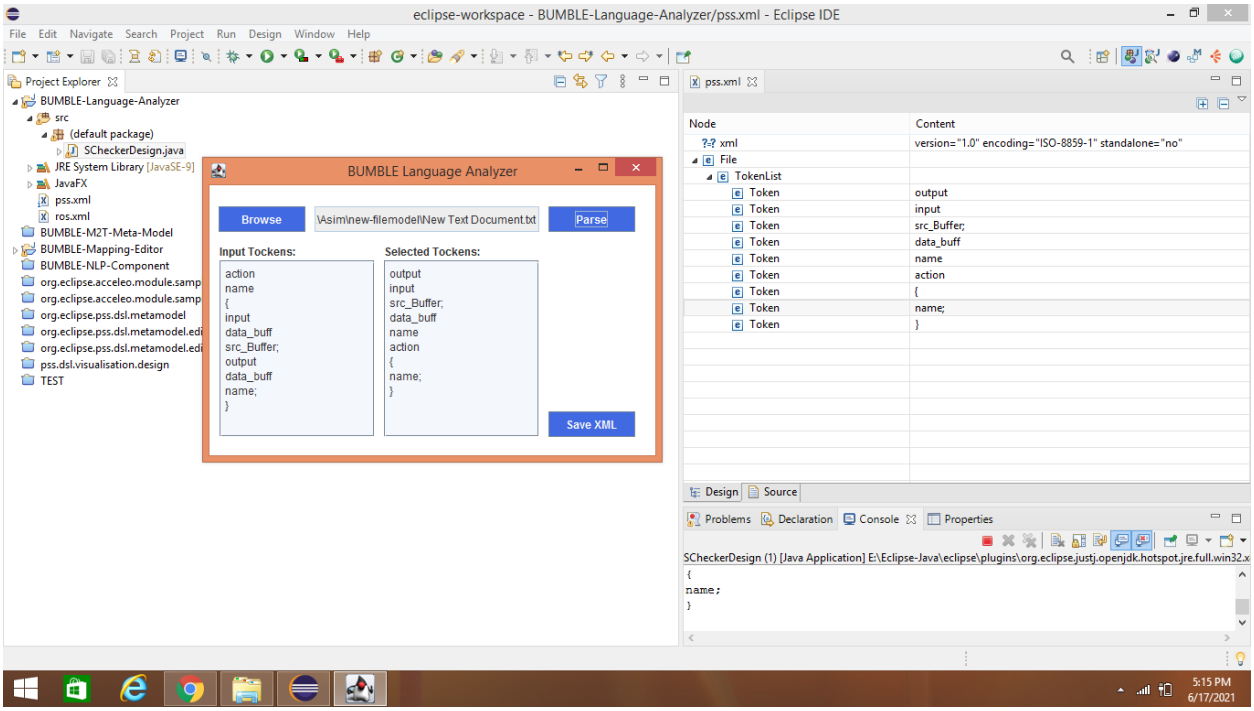

Figure 6: Implementation of Language Analyzer Component

2.4 BUMBLE-Mapping-Editor

A mapping editor is developed with high level of flexibility in order to be utilized in different domains broadly. Particularly, all the interface components are generated dynamically through XML files as shown in **Figure 7**. The "graphical-tree" XML is used to generate graphical mapping portion. Here, graphical concepts are displayed through this XML which was generated in BUMBLE-M2T-Meta-Model component. The symbols are displayed through "forSymbol" XML. Basically, this XML contains repository of symbols which are displayed dynamically. Therefore, the stakeholders / users can provide the symbols of their choice for mapping in XML format. Similarly, textual syntaxes are displayed through "Textual-Syntaxes" XML which was generated through BUMBLE-Language-Analyzer component. To summarize, BUMBLE mapping editor is flexible enough to be utilized in different domains broadly.

A screenshot of "forSymbol" and "graphical-tree" XMLs are shown in **Figure 8**. It can be seen that symbols XML contains three values (i.e. id, name and path) for each symbol and all values are displayed in editor dynamically. Similarly, concepts and sub-concepts names are fetched from graphical-tree XML for the displacement in Editor. The stakeholders / users can provide their own concepts and symbols in XML format for mapping process.

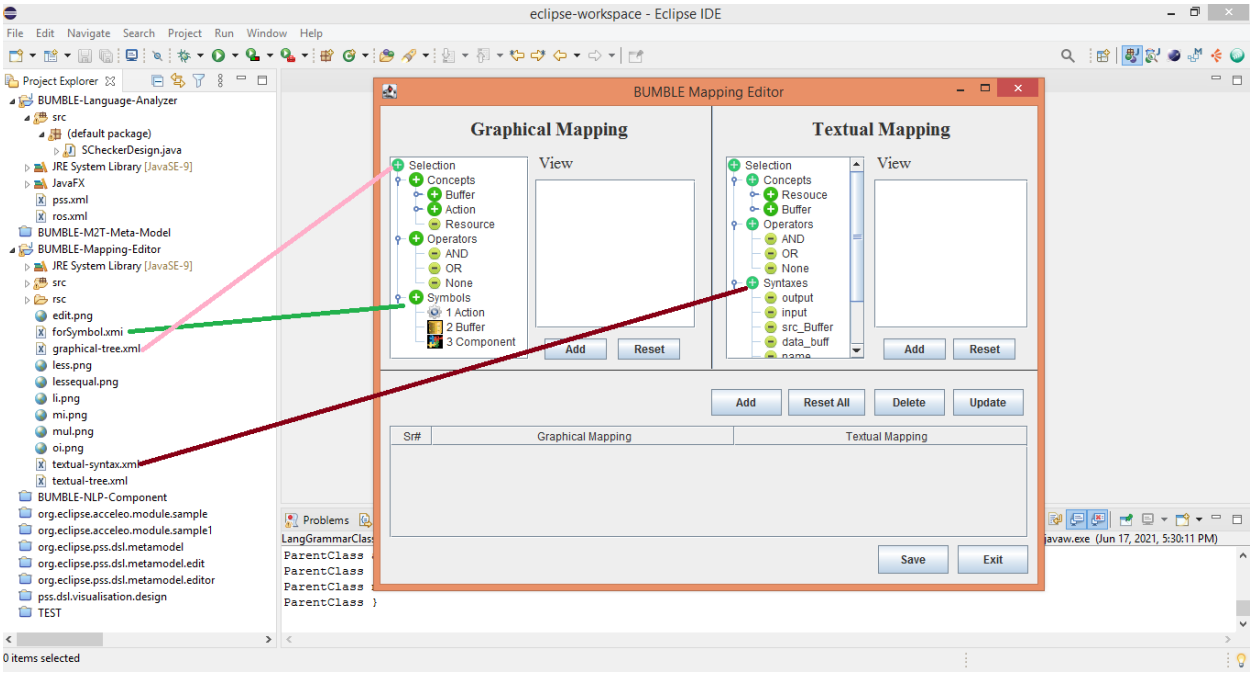

Figure 7: Implementation of Mapping Editor

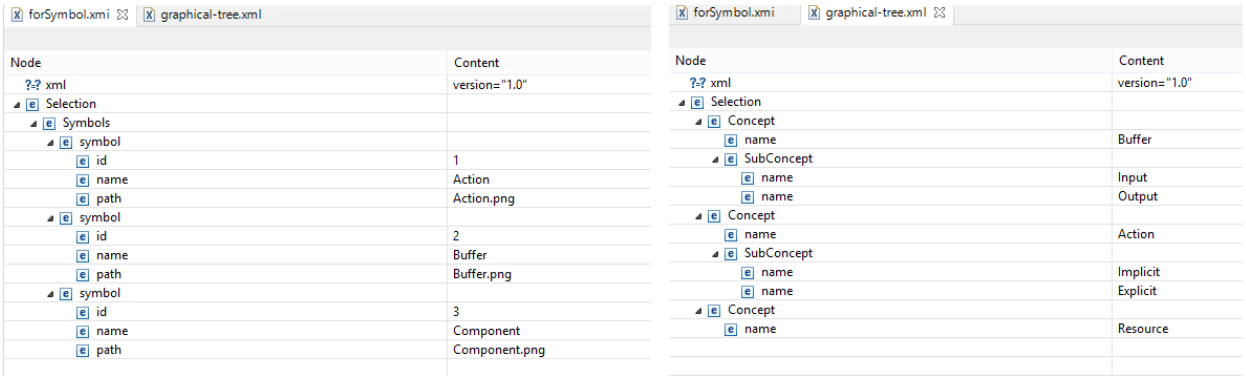

Figure 8: Implementation of Mapping Editor – ForSymbol and graphical-tree XMLs

A comprehensive and accurate mapping between graphical and textual syntaxes can be performed through editor as shown in **Figure 9**. Here, PSS concept "action" and its symbol is selected in graphical mapping. On the other hand, "action name {" is selected as an equivalent textual syntax. This mapping can be added in grid through add button. Similarly, detailed mapping can be performed, and all the mapping information can be saved in XML file as shown in **Figure 10**. This XML can be utilized further for the implementation of EBNF grammar.

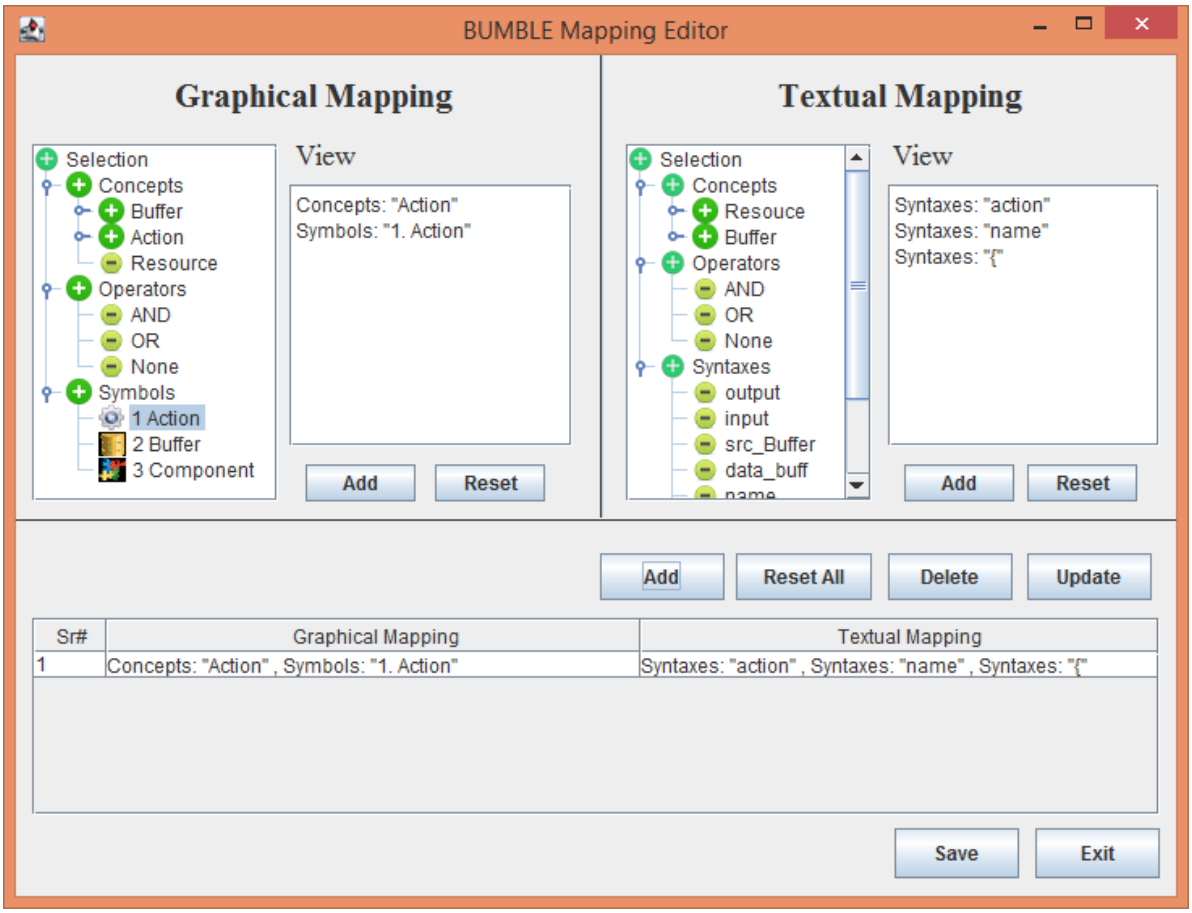

Figure 9: Implementation of Mapping Editor – PSS Action Mapping Process Screenshot

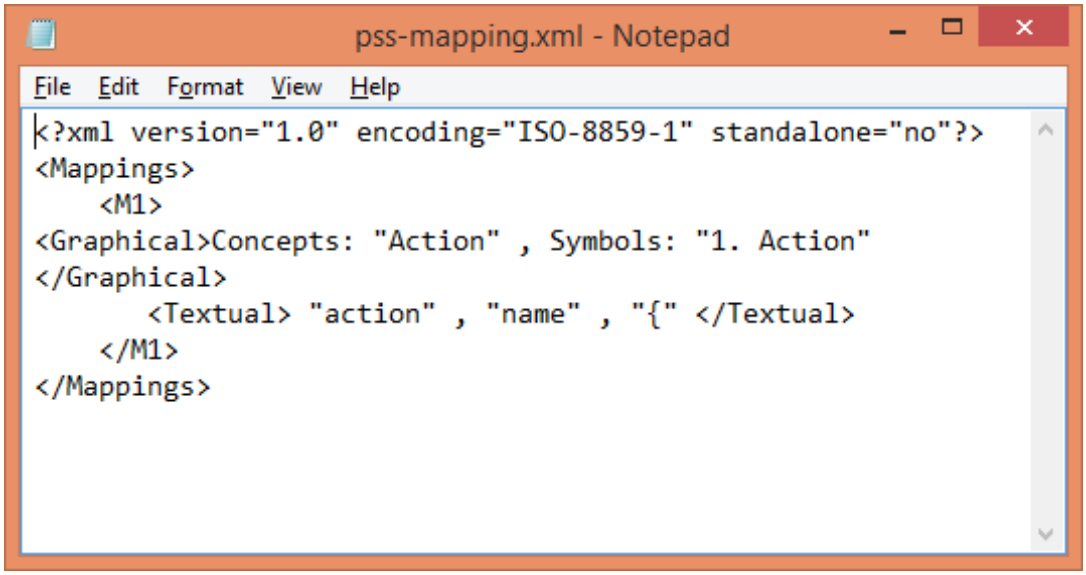

Figure 10: Mapping Information XML

2.5 BUMBLE-EBNF-Generator

To achieve seamless runtime synchronization between graphical and textual syntaxes, EBNF grammar is developed by utilizing Editor's mapping information (in XML). Few EBNF rules for the PSS action and buffer concepts are given here for demonstration purposes:

Rule 1: <Action> : := <Graphical-Action> | <Textual-Action>

Rule 2: <Graphical-Action> : := <Name><Symbol> |
<Name><Symbol><Relationship>(<Graphical-Data Buffer>)*

Rule 3: <Textual-Action> : := action <Name> { } | action <Name> { (<Textual-Data Buffer>)* }

Rule 4: <Data Buffer> : := <Graphical-Data Buffer> | <Textual-Data Buffer>

Rule 5: <Graphical-Data Buffer> : := <Type><Name><Symbol>

Rule 6: <Textual-Data Buffer> : := <Type> data_buff <Name>;

Rule 7: <Relationship> : := Containment <Symbol>

Rule 8: <Name> : := ([a-z][A-Z][0-9])*

Rule 9: <Symbol> : := ([a-z][/][\][A-Z][0-9])*

Rule 10: <Type> : := input | output

For better understanding, consider a simple PSS DSL example where an action having two buffers is defined as: *action mem2mem { input data_buff src Buffer;*
output data_buff dst Buffer;}

Based on aforementioned EBNF rules, a parsing tree of given example is shown in **Figure 11** to achieve seamless synchronization and switching between graphical and textual notations. Please note that terminal symbols are displayed in orange boxes.

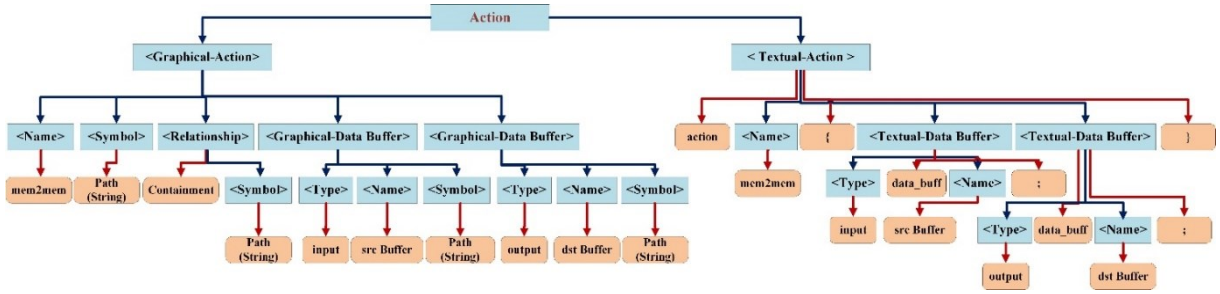

Figure 11: Parsing tree of EBNF rules for PSS action concept

The implementation of EBNF rules is accomplished through JAVACC as shown in **Figure 12**. Particularly, all rules are implemented through JAVACC template file (grammar). The implementation of EBNF is done as a service, so that, it can be easily imported in other tools for editor generation like Sirius. It can be seen from **Figure 12** that "MyNewGrammar.jj" implements the rules while equivalent parser is automatically generated by JAVACC in default folder (e.g. multiple files like EG1.java etc.).

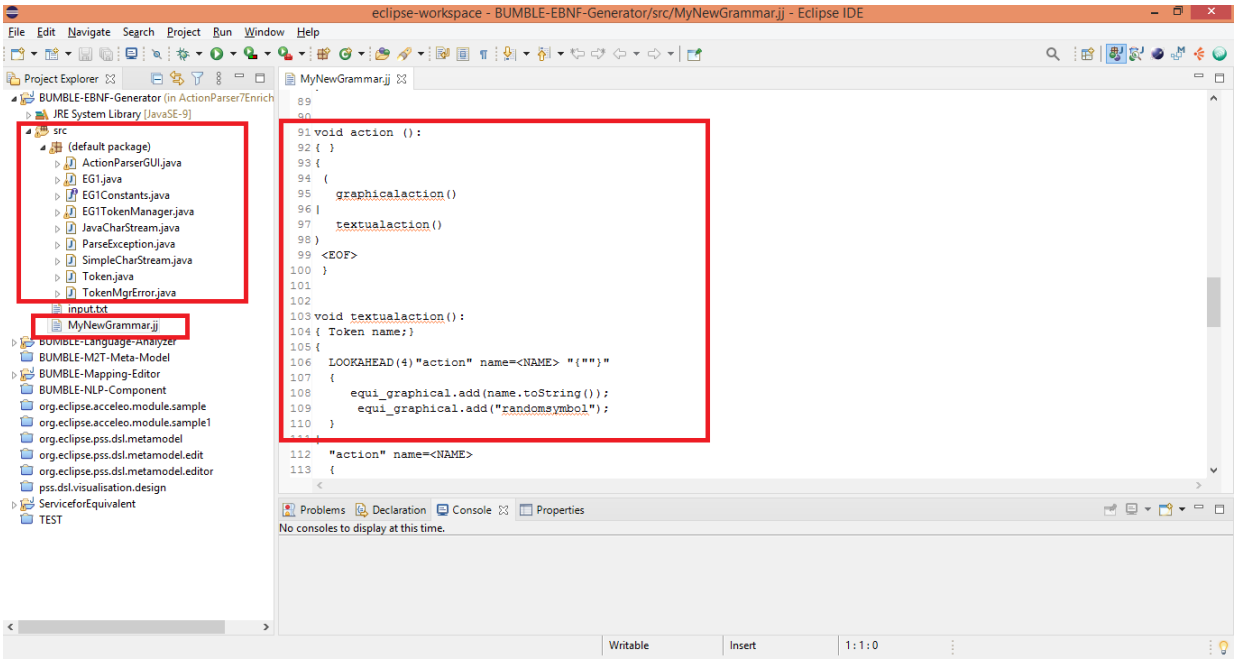

Figure 12: Implementation of EBNF Grammar Generator

To verify EBNF JAVACC service, a test client application with GUI is also developed as shown in **Figure 13**. Particularly, it provides interface to input textual or graphical syntaxes and subsequently, calls EBNF service to receive equivalent textual or graphical syntaxes accordingly as shown in **Figure 13**.

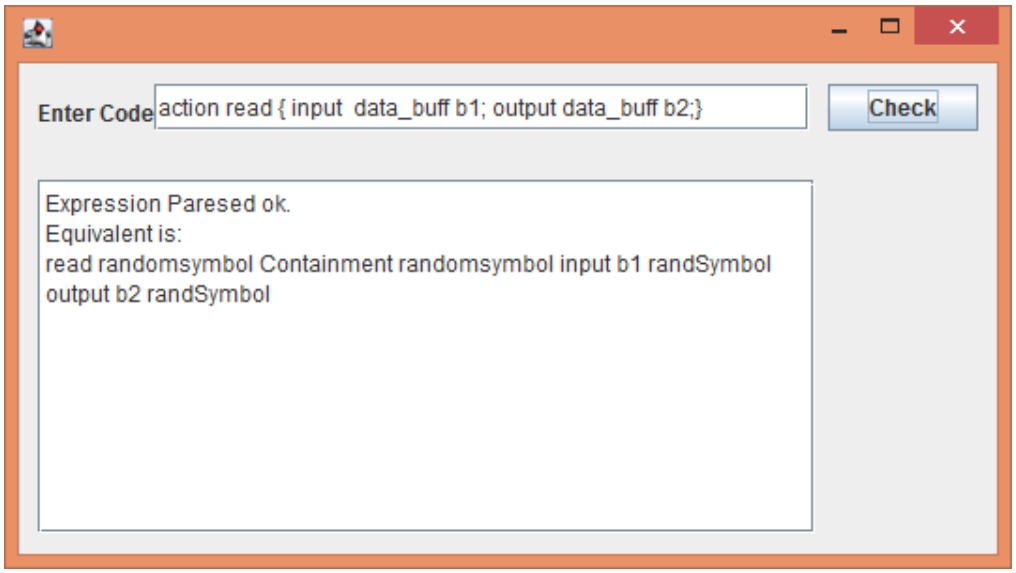

Figure 13: Client Application for the Verification of EBNF Grammar Service

2.6 PSS Blended Modeling Editor

The Sirius platform is utilized to generate the blended modeling editor. The architecture of the editor generation component is shown in **Figure 14**. In the first step, a graphical editor is generated in Sirius by utilizing the PSS meta-model. In the second step, a textual view is incorporated in the graphical editor for the specification of textual PSS. The blended modeling editor supports seamless synchronization and switching between graphical and textual notations using Java services, model-to-text and text-to-model transformations, and the Synchronizer component, as shown in **Figure 14**.

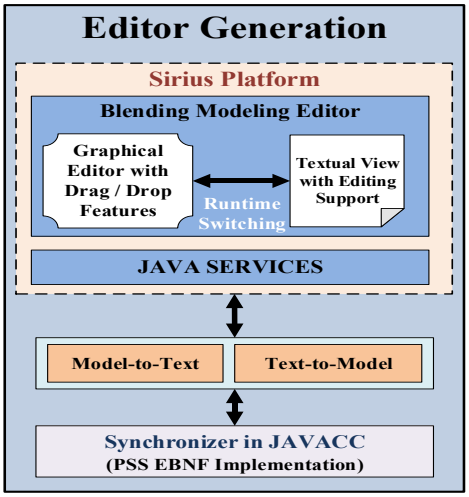

Figure 14: Architecture of editor generation component

The generated PSS blended modeling editor is shown in **Figure 15**. Particularly, it offers drag/drop functionality for different PSS graphical elements (provided in a palette) with suitable symbols that were chosen during the mapping process. The "Update Views" button is provided to synchronize the two notations on-demand. The outcome of the synchronization process (i.e. Successful, Done with Errors and Failed) is displayed.

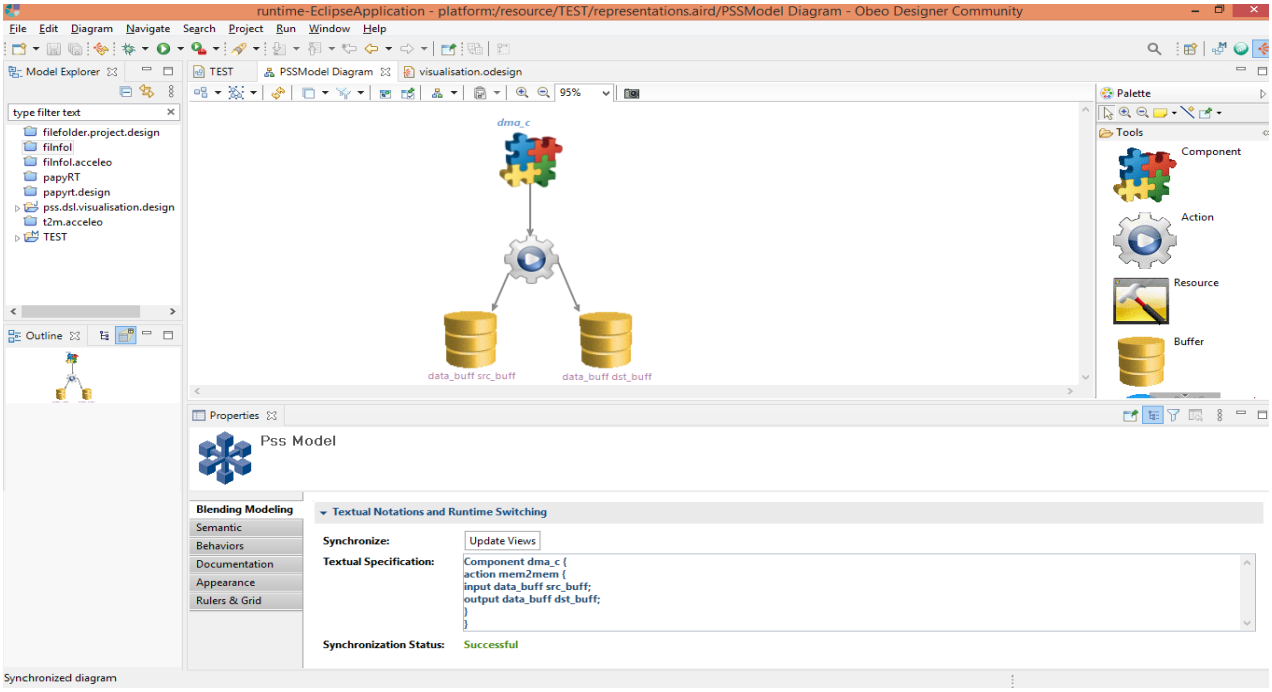

Figure 15: PSS Blended Modeling Editor