Technical Report:
Unresolved Challenges and Potential Features in EATXT

Contributed by:
Dr. Jörg Holtmann
Independent Researcher
Weixing Zhang
PhD student
Chalmers | University of Gothenburg

Edited by:
Weixing Zhang
weixing@chalmers.se

GÖTEBORGS
UNIVERSITET

CHALMERS

Computer Science and Engineering
Chalmers | University of Gothenburg
Sweden
2023

1 INTRODUCTION

The BUMBLE project[1] was launched to provide an innovative system and software development framework based on blended modeling notations (or languages, such as textual and graphical). In this project, the team from the University of Gothenburg ("We" or "GU" in short) conducted technical experiments/implementation on blended modeling[2] using EAST-ADL as a case language to explore and improve blended modeling technology. EAST-ADL[3] is a language used to describe the architecture of automotive embedded systems. EATOP[4] is a specialized tool that supports EAST-ADL modeling. Like some of the tools we reported in [1], it supports blended modeling by supporting part of the common notations but does not support the textual notation. As a start of the work, we used Xtext technology[5] to develop a textual syntax (and a textual editor that supports it) for EAST-ADL based on the EAST-ADL metamodel. We named this textual syntax and the textual editor **EATXT**.

We have implemented multiple features in EATXT and have identified some potential features that may be added to EATXT in the future and the challenges involved in implementing them. This document focuses on describing potential advanced features and challenges that have not yet been implemented that can be added to EATXT. This paper does not discuss in detail the features that have been implemented, the use of the software, and the research involved.

2 SOFTWARE DEVELOPMENT OVERVIEW

As mentioned before, the software we developed is a textual syntax and a textual editor that supports it, we named it EATXT. It supports building, editing, and viewing EAST-ADL models in the form of text, thus filling the gap in textual notation among existing modeling tools that support EAST-ADL.

Figure 1 depicts an overview of our solution to developing EATXT. We first used Xtext to generate the grammar from the EAST-ADL metamodel. We then designed and developed the grammar by adapting the generated grammar, including completing the definition of missing grammar rules, adjusting the grammar style, etc. When the grammar is ready, we run the MWE2 workflow [6] to generate the Xtext artifacts, which contains the Xtext's generator used to generate the textual editor that supports the grammar. To customize and enhance the features of the textual editor, we extended this Xtext generator.

3 SOLVED TECHNICAL ISSUES

We have implemented EATXT's grammar by directly adapting the text of the grammar definition, including 1) completing the incomplete grammar rules, 2) creating primitive types, 3) repositioning the attribute `shortName`, 4) removing redundant elements (such as curly braces), 5) adding symbol ";", and 6) supporting empty elements. This adaptation also makes the grammar style tend to be Python style. This style change refers to the work of [3], but retains the use of curly braces to separate code blocks. For information on how to convert a language with a generated grammar into a Python-style language, please refer to [3]. We automated this adaptation using the tool GrammarOptimizer [4].

Moreover, we have added or enhanced some advanced features of the textual editor, including 1) supporting code templates, 2) supporting automatic formatting, 3) supporting content-assist for a new model element with a unique name, and 4) supporting more accurate cross-references. We implemented those features by customizing and extending Xtext's generator capabilities and its implementation which we have introduced in [2]. We also implemented conversion (i.e., serialization and deserialization) between textual models (.eatxt model file) and tree-based models (.eaxml model files) on small model instances.

[1] https://itea4.org/project/bumble.html
[2] https://blended-modeling.github.io/
[3] https://www.east-adl.info/Specification.html
[4] https://projects.eclipse.org/projects/modeling.eatop
[5] https://eclipse.dev/Xtext/index.html
[6] https://wiki.eclipse.org/Modeling_Workflow_Engine_(MWE)

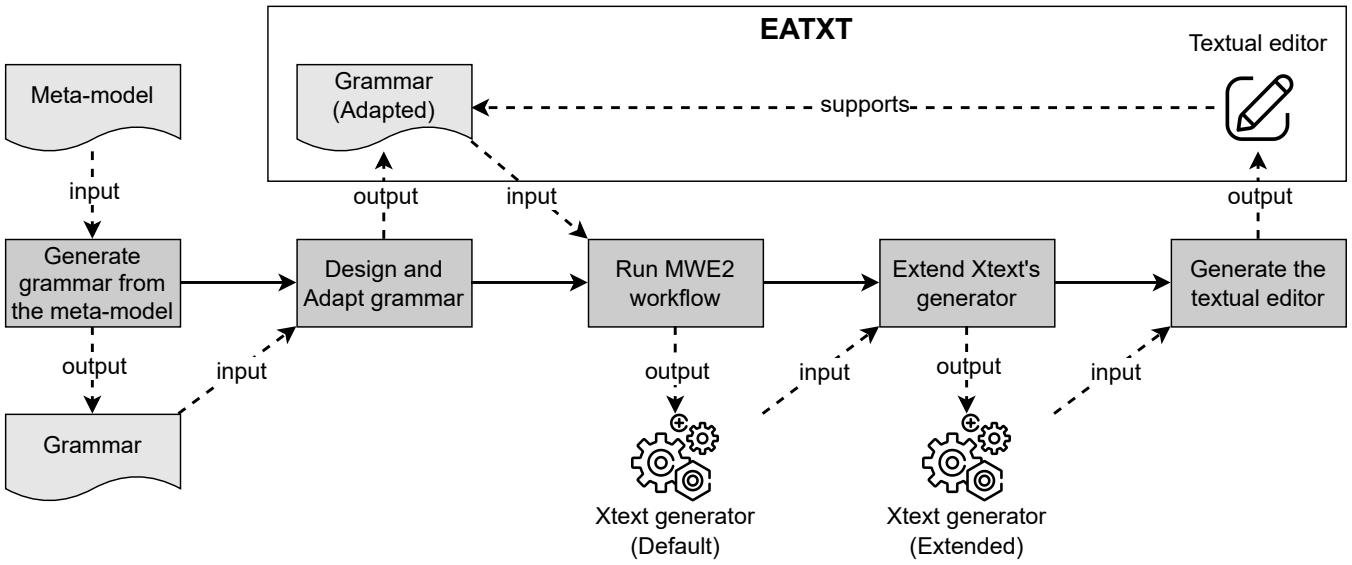

Figure 1: Overview of solution to developing EATXT.

4 UNSOLVED TECHNICAL ISSUES

Unresolved technical issues in EATXT hinder the realization of potential advanced features. Below we will describe the unresolved technical issues by describing these unimplemented advanced features.

4.1 Blended Modeling of EAST-ADL

Figure 2 depicts how EATXT will work with EATOP to implement blended modeling of EAST-ADL. The current EATXT we implemented can serialize .eatxt model files into .eaxml model files and vice versa, thus achieving blended modeling. The unsolved challenges here are two: 1) This blended modeling feature only works with smaller models, but not larger ones. 2) The current EATXT and EATOP are two independent software, so when performing model conversion, users have to manually import the model file into the software.

4.2 Flexible Schema Version

In the second line of the .eaxml file, the field “xmlns" contains the version of the EAST-ADL metamodel that this file complies with. The version of the metamodel is derived from the EAST-ADL specification it adheres to. Since the initial release of the EAST-ADL specification in 2001, the EAST-ADL metamodel has been evolving. In practice, there are differences between the schemas of consecutive versions, while containing similar essential information. EATOP can tolerate these differences and support multiple versions. However, Xtext is more stringent in this regard. As a result, a textual editor for EAST-ADL developed using Xtext (i.e., EATXT) may not support models adhering to schemas of other EAST-ADL versions. This affects the efficiency of blended modeling in EAST-ADL.

For instance, our industrial partner extended EATOP by developing a graphical view plugin to support the graphical view of models. Although EATOP can accommodate different versions, the graphical view plugin is specific to schema version 2.1.12, and therefore, they model EAST-ADL based on version 2.1.12. Yet, we developed the EATXT based on the latest metamodel version (i.e., 2.2), which does not support loading and resolving models obtained from EATOP due to version mismatch. Consequently, users must manually adapt the models in the .eaxml files before importing them into EATXT. This manual effort adds extra steps in switching between tree-based and textual notation, thereby burdening the efficiency of blended modeling.

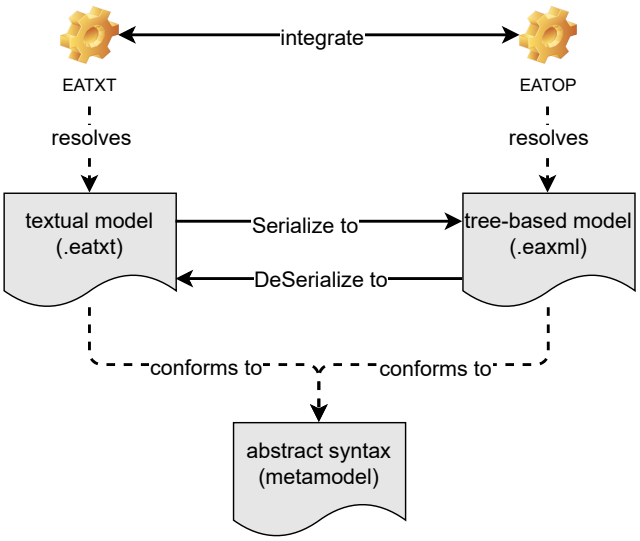

Figure 2: Schematic diagram of EAST-ADL's blended modeling, including serialization and ensemble..

To address this issue, EATXT needs to tolerate differences between different versions, enabling users to seamlessly and freely load models from various versions. One possible solution is to have EATXT assess the version of the .eaxml file upon loading. If the version is inconsistent with the EAST-ADL version EATXT is based on, the file should undergo an XSLT transformation to convert it to a format compliant with the version supported by EATXT. By doing so, EATXT ensures compatibility between different EAST-ADL versions and facilitates efficient modeling for users.

4.3 Error Reporting

EATXT was built by automatically generating Xtext artifacts from the grammar. It already includes basic error reporting functionality, with error messages displayed in the "Problems" window. However, these error reports are generic and may not provide specific details in certain cases. For example, when a user inputs content that does not match the expected text of the editor, such as typing `shortName` instead of the required keyword `EAPackage`, the error message only informs the user of the mismatch without providing further context.

Currently, when EATXT attempts to load an incompatible version of an .eaxml file, it rejects the model without providing helpful error messages. We envision that the EATXT in the future will point out the exact issue in such scenarios. For instance, before EATXT loads the .eaxml model file, it could automatically check the version of the EAST-ADL schema that the .eaxml and the EATXT are based on respectively. If there is a version mismatch, EATXT will clearly indicate this in the "Problems" window.

It is important to note that different schema versions may have different elements. For example, in version 2.1.12 of EAST-ADL metamodel, there is an attribute in the class `HardwareFunctionType` that references to class `HardwareComponent`, while in version 2.2, such an attribute does not exist. Consequently, a model file (.eaxml) that conforms to version 2.1.12 and contains this attribute would not be loadable by EATXT based on version 2.2. In the envisioned future of EATXT, it would be capable of loading such a file, as described in the previous section, while also providing an accurate error message that the attribute is not necessary for the current version.

4.4 Precise Outline

The editor, composed of automatically generated Xtext artifacts, also includes an outline feature, which provides a tree-based representation of model instances, as shown in Figure 3. The current version of EATXT includes this

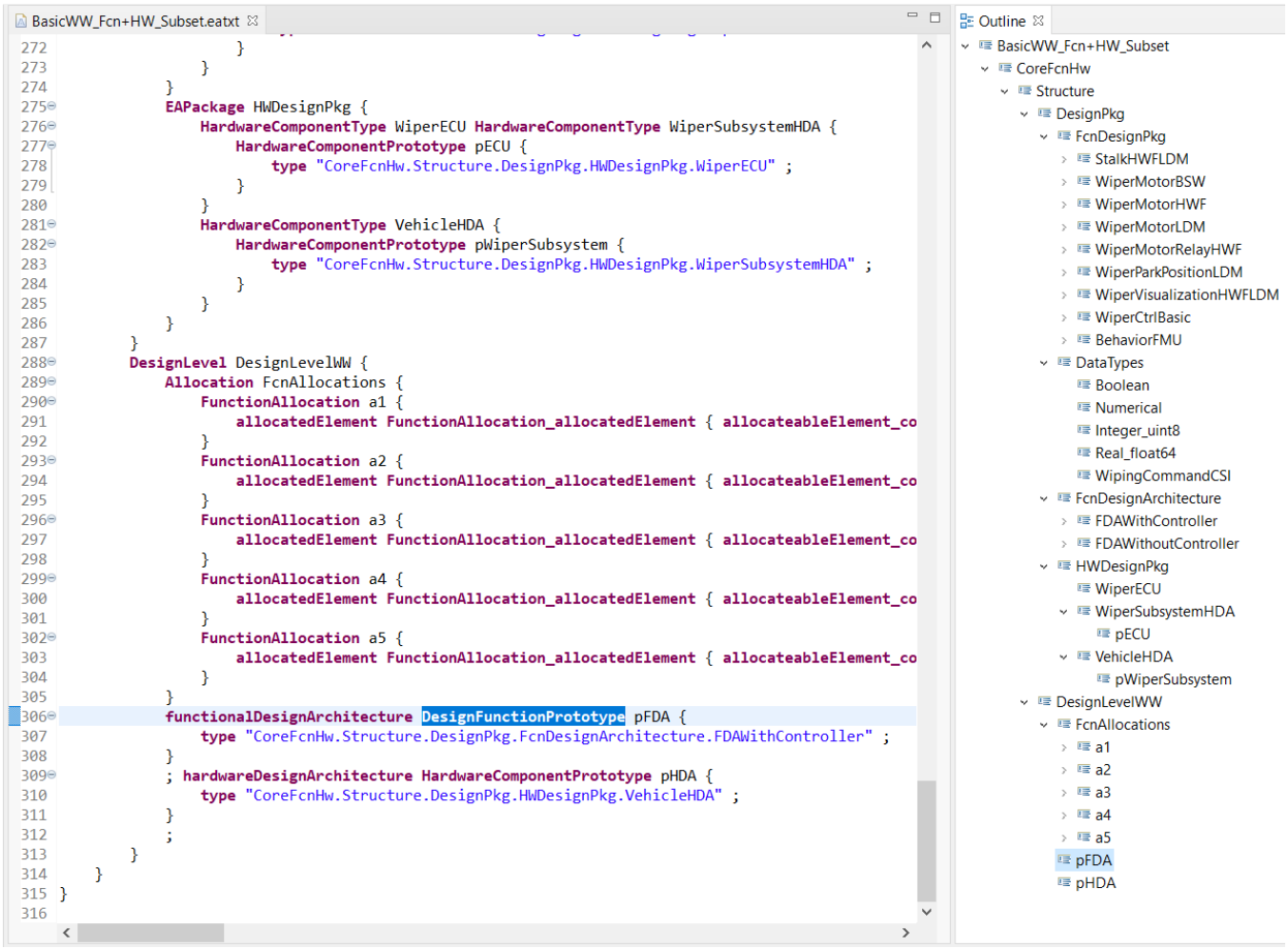

Figure 3: An example of outline in EATXT: *pFDA* stops expanding downwards.

default and generic outline view. However, this outline lacks important details, hindering users' basic understanding of the model structure. For example, Figure 4 is a screenshot from EATOP and it illustrates an instance of type `DesignFunctionProtoType` named `pFDA`, which contains several sub-instances that are cross-referenced to other types. In EATOP's tree-based view, these contents are displayed. However, in the default outline view of the textual editor, the contents below `pFDA` are hidden, and shown hierarchy stops at `pFDA`.

4.5 Integration of EATOP and EATXT

EATXT facilitates textual modeling of EAST-ADL, with its models capable of being transformed into ".eaxml" model files that can be edited within EATOP, and vice versa. This is the way to implement blended modeling for EAST-ADL. Yet, EATXT is an independent software application separate from EATOP. This signifies that during the blended modeling of EAST-ADL, users are required to run both software applications, necessitating toggling between the two. The challenge here is the integration of one of these applications into the other, with the transformation between ".eatxt" and ".eaxml" files being executed in the background without user intervention. Text-based modeling and tree-based modeling would manifest as two distinct windows within the same software. When a user modifies

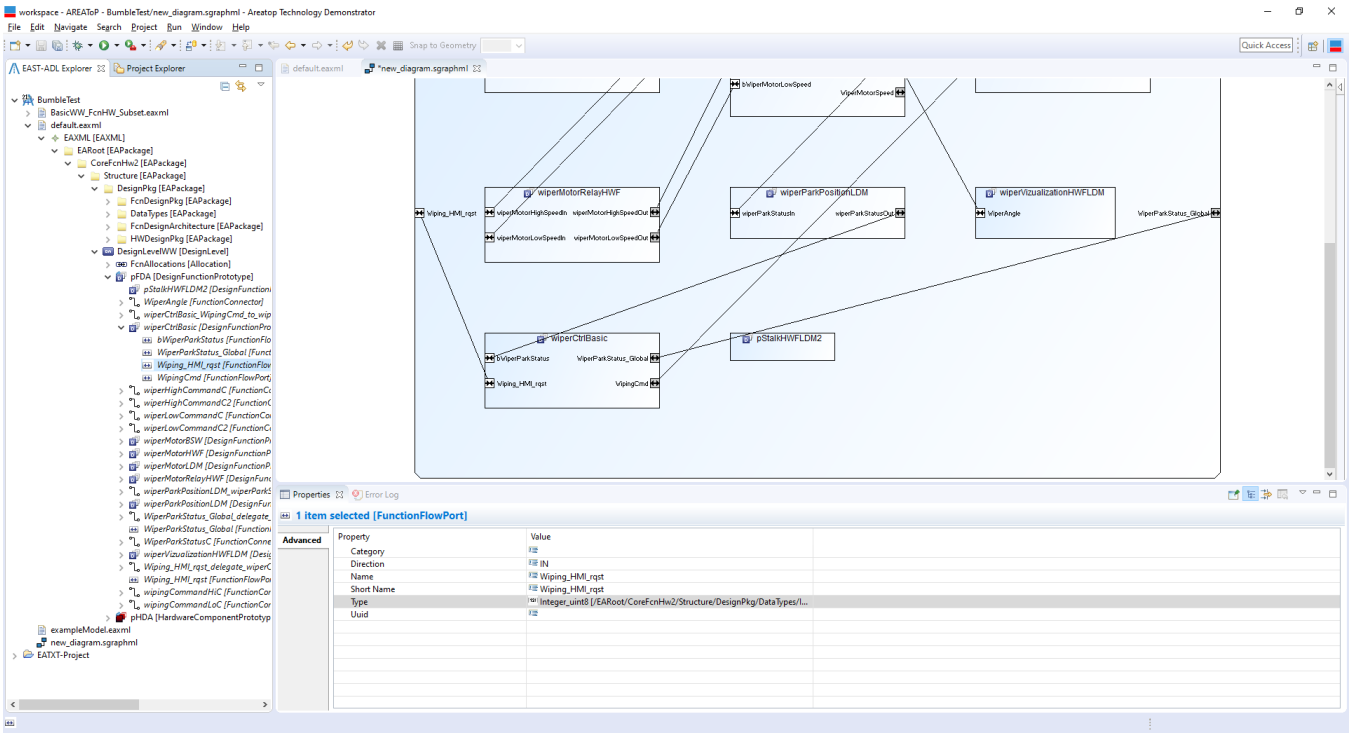

Figure 4: An example of outline in EATOP: *pFDA* continues to expand downwards.

and saves a model in one window, the software automatically converts the model file, enabling the user to directly open the model in the other window and continue editing in an alternative notation.

5 TECHNICAL CHALLENGES AND FUTURE WORK

The challenges we need to address include: realizing schema version identification and tolerance of element differences between different versions of EAST-ADL, customizing error reporting features, rendering deeper hierarchies in outline view, and implementing automatically synchronizing model files of different formats within EATOP. And these will be our future work.

6 CONCLUSION

We report in this document potential and unimplemented advanced features/challenges that could be added to EATXT. Peers working on improving the blended modeling capabilities of the EATOP and those working on developing textual editors with Xtext may benefit from this document.

REFERENCES

[1] Istvan David, Malvina Latifaj, Jakob Pietron, Weixing Zhang, Federico Ciccozzi, Ivano Malavolta, Alexander Raschke, Jan-Philipp Steghöfer, and Regina Hebig. 2023. Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study. *Software and Systems Modeling* 22, 1 (2023), 415–447.

[2] Jörg Holtmann, Jan-Philipp Steghöfer, and Weixing Zhang. 2023. Exploiting Meta-Model Structures in the Generation of Xtext Editors. In *11th Intl. Conf. on Model-Based Software and Systems Engineering (MODELSWARD)*. 218–225. https://doi.org/10.5220/0011745900003402

[3] Weixing Zhang, Regina Hebig, Jan-Philipp Steghöfer, and Jörg Holtmann. 2023. Creating Python-style Domain Specific Languages: A Semi-automated Approach and Intermediate Results. In *11th Intl. Conf. on Model-Based Software and Systems Engineering (MODELSWARD)*. 210–217. https://doi.org/10.5220/0011744900003402 (in press).

[4] Weixing Zhang, Jörg Holtmann, Daniel Strüber, Regina Hebig, and Jan-Philipp Steghöfer. 2023. Meta-model-based Language Evolution and Rapid Prototyping with Automate Grammar Optimization.