




# Developing Reflex IDE Kernel with Xtext Framework

Alena Bastrykina   
*Institute of Automation and  
 Electrometry SB RAS  
 Novosibirsk, Russia*

Vladimir Zyubin   
*Institute of Automation and  
 Electrometry SB RAS  
 Novosibirsk, Russia*

Andrey Rozov   
*Institute of Automation and  
 Electrometry SB RAS  
 Novosibirsk, Russia*

**Abstract**—In this paper, we describe the technology of the process-oriented language Reflex IDE kernel development. The Reflex language, which is being maintained at the Institute of Automation and Electrometry, is a language for cyber-physical systems software specification. In the paper, we assume that the cyber-physical system is a computational core that interacts with the physical world. In the case of Reflex, the computation platform is an industrial PC. Reflex IDE (RIDE) includes a language-based editor, syntax and semantics analyzers as well as an abstract syntax tree (AST) generator, and a class library for working with the generated AST. In this work, we explain our motivation for the research, formulate the requirements for the development, and present the RIDE architecture. We describe the RIDE development process using Eclipse/Xtext tools and its user interface. We also provide an example of extending the Reflex IDE kernel with a code generator for the AVR platform. In the conclusion, we discuss the possibility of using the obtained result to create a web-version of RIDE.

**Keywords**—*process-oriented programming, Reflex, Xtext, parser, translator, DSL.*

## I. INTRODUCTION

The use of general-purpose languages for the implementation of cyber-physical systems control algorithms leads to an increase in the complexity of the software architecture and makes it difficult to develop, debug and maintain such systems.

The process-oriented language Reflex, developed at the Institute of Automation and Electrometry, has been successfully used in multiple industrial applications and demonstrated promising qualities, producing easily readable, maintainable code and overall robust and dependable software [1].

The existing toolkit for the Reflex language was created “manually”, without the use of automated DSL development tools. The technologies that were used for developing each tool are heterogeneous, thus, the toolkit is difficult to integrate into a single development environment, and it is hard to support. Therefore, the acute task is to refactor the existing tools in order to create an integrated development environment for the Reflex language. The IDE development should be based on modern technologies for creating DSL tools. Modern trends in the field of web services and their advantages [2] also should be taken into account when

formulating the requirements for the IDE.

The article structure is the following. In the second section, we state the requirements for the development. In the third section, we describe the RIDE architecture and the DSL tools we have chosen to create the IDE. The fourth section depicts the language-based editor and parser development process using the Eclipse/Xtext tool. The fifth section describes the semantic checks’ implementation using standard Xtext methods. In the sixth section, we describe the IDE extension process and provide an example of extending the Reflex IDE kernel with a code generator. Finally, we summarize the results and consider that the developed Reflex language toolkit can be ported into a web-IDE platform.

## II. REQUIREMENTS FOR THE REFLEX IDE

The requirements were based on the following:

1. Text format for representing DSL source codes is preferred since it allows to view and edit source codes in all the standard text editing programs.
2. Parser-generation method simplifies language tools maintenance, particularly by decreasing an effort for changing or expanding the language syntax.
3. The Reflex project itself is a research project. There are already several tools for processing Reflex programs that have been developed as a result of the research, e. g. a dynamic code verification system, a C-code generator for PC-based control, etc. Moreover, some language tools are in progress (static code verification tools, debugging toolkit). We want to have the possibility to integrate all these tools into one single IDE.
4. Modern language tools tend to have a language-based editor and provide features like autocompletion, code navigation, static code analysis, syntax and semantics errors reporting [3].
5. The increasing popularity of web-IDEs. A web-IDE lacks deployment on the user’s PC, simplifying the process of getting started with the new language. This stimulates new users to learn and use the language.

As a result of the analysis, the following requirements were formulated for the IDE project:

- (1) text format should be used for representing Reflex program source codes;

---

This work was supported by the Russian Ministry of Education and Science, project no. AAAA-A19-119120290056-0.

(2) the development should be based on an automated parser generation method;

(3) the IDE should be extensible with problem-oriented modules;

(4) the IDE kernel should include an editor, which provides syntax highlighting, interactive diagnostics based on syntax and semantics analysis;

(5) chosen DSL technologies should allow developers to port the IDE to different platforms in order to use it in both desktop and web modes.

### III. SYSTEM ARCHITECTURE AND DEVELOPMENT TECHNOLOGIES

The developed IDE architecture and workflow is shown in Fig. 1. RIDE kernel consists of the following components:

1. Language-based editor.
2. Parser, which is automatically generated from the description of the Reflex grammar. Parser is triggered each time when the user changes program code in the editor. Parser performs syntax analysis and produces an AST.
3. Semantic analysis components, which perform a set of semantic checks on the generated AST.
4. Delegating code generator, which controls the operation of problem-oriented extension modules. The delegating code generator is fired each time the user saves program source code in the editor. It takes an AST as input and redirects it to the loaded extension modules.
5. Problem-oriented IDE extension modules, which take the program AST as input and produce the set of particular artifacts (e.g. generated code).

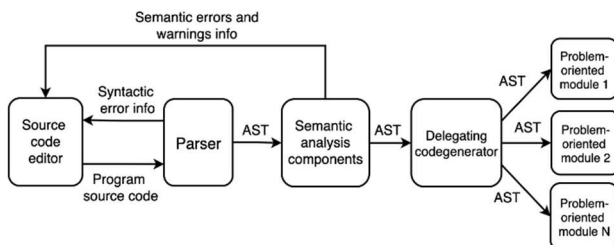


Fig. 1. Reflex IDE architecture

The following tools for domain-specific languages development were analyzed:

- GNU Bison [4] and ANTLR4 [5] parser-generators;
- JetBrains MPS [6] and Xtext [7] DSL development frameworks.

The DSL development framework usage was preferred, hence to reduced effort on building complex IDE architecture and automatic source code editor generation.

Based on the analysis results, it was decided to choose Eclipse/Xtext technologies to build a RIDE kernel.

Xtext is a powerful DSL workbench based on the Eclipse Platform. The main advantages of the framework for building RIDE kernel are:

1. Automatic parser, editor and generation based on DSL definition.

2. API's for defining different types of semantic checks for DSL.

3. Integration with the Xtend language, a JVM language, which translates to Java [8]. Xtend expands Java with useful 'syntactic sugar', making it more flexible and convenient to write a code generator for the DSL.

4. Integration with the Language Server Protocol (LSP). Thus, although Xtext uses Eclipse as a target platform, a part of Reflex language artifacts generated by Xtext can be ported into various IDE's and text editors, including Web-IDEs, which support LSP [9]. The full list of such tools is introduced in [10], the most popular are: IntelliJ, VSCode, Sublime.

JetBrains MPS was considered almost as powerful as Xtext, but unfortunately, it offers only a projectional way of editing programs. This means that the program semantic mode, represented by the AST, is being edited directly from the projectional editor. Therefore, the format used by MPS to represent DSL source code is not textual. Moreover, there is no technology for migrating MPS-based language projects into other IDE platforms, including web-IDE engines. All these disadvantages violate the requirements for the Reflex IDE.

### IV. LANGUAGE-BASED EDITOR AND PARSER

Xtext offers a special language, called the Grammar Language, to describe languages with LL-grammars. The Grammar Language is based on the Extended Backus-Naur Form (EBNF) [11, 12] and provides a possibility to describe semantic relations in an abstract syntax tree (AST).

Xtext relies on Eclipse Modeling Framework (EMF) [13] to build Abstract Syntax Trees defined using the Grammar Language. Based on the grammar, Xtext creates an EMF metamodel description for the defined language. In these terms, the Grammar Language rules are used as a definition for the EMF language model entities. The grammar rules describe entities names, properties, and also references between the entities. The parser generated by Xtext creates an abstract syntax tree as an EMF model — which itself is an instance of the described EMF metamodel.

The grammar of the Reflex language was defined using the Grammar Language. Given the definition of the Reflex grammar, the Xtext workbench automatically generated the Eclipse-based IDE UI, which includes the Reflex language editor, a parser, and a Java class library for working with AST.

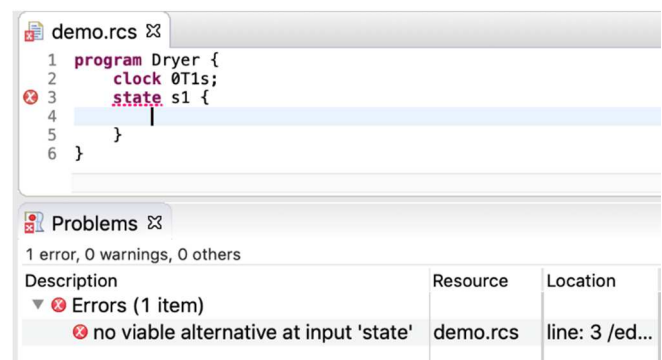


Fig. 2. An example of Reflex language syntax violation report

Syntax errors, which are detected as a result of the parser's work are interactively displayed in the IDE, as shown in Fig. 2. Words containing syntax errors are underlined in the editor, the line containing syntax violation becomes annotated with error marker, and also an error message with the full location information appears in the Problems IDE view.

## V. SEMANTIC CHECKS IMPLEMENTATION

The implementation of the semantic analysis components was based on the customization of the stubs, automatically generated by Xtext. Xtext offers two methods of specifying semantic checks for the developed DSL: a technology for the mechanisms for the specification of semantic rules (validation) and a technology for the specification of contextual links between program identifiers and AST objects (linking and scoping) [14].

A semantic analysis of the reachability of processes and states, the use of variables and other rules typical for the Reflex language was provided through the validation mechanism. The work of validation mechanism was achieved by customizing an AbstractDeclarativeValidator Java class stub instance with a set of methods, which perform semantic checks on the particular types of entities in the AST (processes, states, variables, expressions, etc.).

Checking the availability of variables, states, etc., used inside the expressions, and also contextual autocompletion and navigation through the program text was implemented using the Xtext linking and scoping mechanism. The work of the mechanism was achieved with the implementation stages, described below.

First, we defined the cross-references between the language concepts at the grammar level. The example introduced in Fig. 3 shows simplified grammar rules for the State Reflex language concept, and for the 'set state' expression. In the State rule, we use a 'name' keyword to establish the place of State identifier inside the syntactic construction. Accordingly, we use the State rule name in square braces inside the 'set state' definition. This way, the identifier of State will be recognized inside the 'set state' expression and will be automatically resolved into the existing AST State object as a result of the parsing.

```
State:
    "state" name=ID "{"
    stateFunction=StatementSequence
    "}";

SetStateStat:
    "set" "state" state=[State] ";;";
```

Fig. 3. Links in the Reflex grammar example

Secondly, we described the semantics of link resolution through the scoping API. The scoping API is introduced by the Xtext IScopeProvider Java interface, which is intended to provide an algorithm for selecting scope for each program identifier based on its context. In the terms of scoping, the scope is the set of AST objects, to which a particular identifier can be potentially resolved. The default implementation of this API provided by Xtext was

customized in order to implement Reflex link resolution semantics.

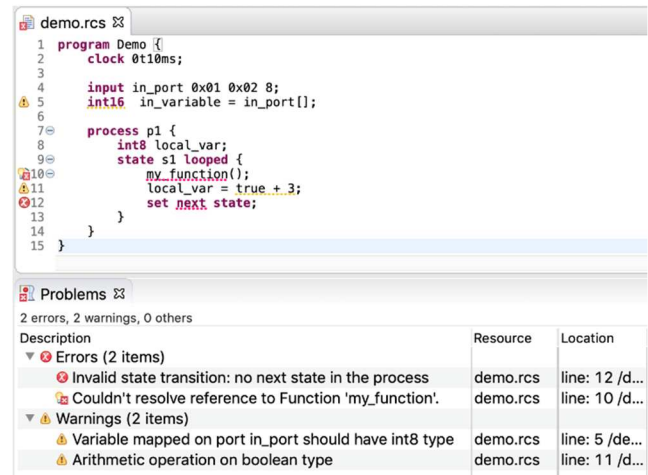


Fig. 4. An example of Reflex language semantic violations (errors and warnings), shown in the editor

All the detected violations of the Reflex language semantics are also reported in the RIDE UI, as well as the syntax errors (Fig. 4).

## VI. IDE KERNEL EXTENSION

Kernel extensibility of the desktop RIDE version was implemented using the standard Eclipse OSGI-bundle extensions concept [15].

The main kernel bundle declares an extension point (problem-oriented module extension point). Meanwhile, a problem-oriented module is represented with an OSGI-bundle, which simply implements this extension point, and the delegating code generator is a Java object of a class that is contained in the main kernel bundle. Once triggered, the delegating code generator finds all the registered extensions, which contribute into the problem-oriented module extension point and invokes the specific method to activate these modules. The delegating code-generator provides an AST of the program as the modules' input by passing it as the argument of the modules' activation method.

The extensibility of the Reflex IDE kernel was verified by implementing a problem-oriented C-code generation module for microcontrollers of the AVR family. The module was written using Xtend language. Given an AST as the input, the module produces a set of \*.c-files, which can be compiled and debugged into a PLC.

## VII. CONCLUSION

As a result of this research, an extensible IDE kernel for the process-oriented language Reflex was created using Xtext/Eclipse technologies. The kernel is based on the parser and semantic analysis module. Kernel's architecture provides an easy way of extending the development environment with additional problem-oriented modules, e.g. executable code generators for different hardware platforms, various Reflex program verifiers, source code analyzers, etc.

The Xtext framework, chosen as a base of the RIDE kernel, provided:

- A language-based code editor with the syntax highlighting,

- interactive reporting of the diagnostic messages, contextual auto-completion and code navigation,
- an automated generation of AST with Java libraries to work with it.

Kernel extensibility technology, based on the Eclipse OSGI-bundle extension mechanism, was developed.

The extensibility of the Reflex IDE kernel was confirmed by implementing a C-code generation module for the AVR microcontroller platform.

The described result was used to create a web version of the IDE, based on the Eclipse Theia technology [16].

Reflex IDE kernel is developed as an open-source project, the source codes are available at <https://github.com/a-bastrykina/reflex-translator-diploma>.

#### ACKNOWLEDGMENT

Authors are very grateful for the charitable support their activity from the JetBrains Foundation. We also would like to thank the anonymous reviewers for their work and constructive comments.

#### REFERENCES

- [1] I. Anureev, N. Garanina, T. Liakh, A. Rozov, V. Zyubin "Towards safe cyber-physical systems: the Reflex language and its transformational semantics," in IEEE International Siberian Conference on Control and Communications (SIBCON-2019), April 2019, Tomsk, Russia, 2019, pp. 1–6.
- [2] M. Walterbusch, B. Martens, F. Teuteberg, "Evaluating Cloud Computing Services from a Total Cost of Ownership Perspective," *Management Research Review*, 36(6), pp. 613–638, 2013. URL: [https://www.researchgate.net/publication/237078103\\_Evaluating\\_Cloud\\_Computing\\_Services\\_from\\_a\\_Total\\_Cost\\_of\\_Ownership\\_Perspective](https://www.researchgate.net/publication/237078103_Evaluating_Cloud_Computing_Services_from_a_Total_Cost_of_Ownership_Perspective)
- [3] A. Jung, J. Carbonell, L. Marchezan, et al. "Systematic mapping study on domain-specific language development tools," in *Empirical Software Engineering*, vol. 25, pp. 4205–4249, August 2020.
- [4] J. Levine, *Flex & Bison: Text Processing Tools*, O'Reilly Media, Inc., 2009.
- [5] T. J. Parr, and R. W. Quong, "ANTLR: A predicated-LL (k) parser generator," in *Software: Practice and Experience*, 25(7), pp.789–810, 1995.
- [6] F. Campagne, *The MPS language workbench: vol. 1*, CreateSpace Independent Publishing Platform, North Charleston SC, United States, 2014.
- [7] L. Bettini, *Implementing domain-specific languages with Xtext and Xtend*, Packt Publishing Ltd, Birmingham B3 2PB, UK, 2016.
- [8] Eclipse – Xtend. Documentation. URL: <https://www.eclipse.org/xtend/documentation/index.html>
- [9] H. Bündler, "Decoupling Language and Editor-The Impact of the Language Server Protocol on Textual Domain-Specific Languages," in *MODELSWARD*, 2019, pp. 129–140.
- [10] A community-driven source of knowledge for Language Server Protocol implementations, URL: <https://langserver.org/>
- [11] R. Pattis, "EBNF: A Notation to Describe Syntax," pp. 1–19, 2013.
- [12] Y. Jianan "Transition from EBNF to Xtext," in *Poster Session and ACM SRC of MODELS*, Valencia, Spain, 2014, pp. 75–80.
- [13] D. Steinberg, F. Budinsky, E. Merks, M. Paternostro, *EMF: Eclipse modeling framework*, Pearson Education, 2008.
- [14] H. Behrens et al. *Xtext user guide*, pp. 43–56. URL: [https://www.eclipse.org/Xtext/documentation/1\\_0\\_1/xtext.pdf](https://www.eclipse.org/Xtext/documentation/1_0_1/xtext.pdf)
- [15] L. Vogel, *Eclipse Rich Client Platform (Vogella Series)*, Lars Vogel, 2015, pp. 145–173.
- [16] K. V. Marchenko, "Using Eclipse Theia to develop an IDE for the process-oriented language Reflex," [Ispolzovaniye Eclipse Theia dlya sozdania integririvannoy sredy razrabotki prorgam na process-orientirovannom yazyke Reflex] (*In Russian*) Data of 58-th