RIDE: Theia-based web IDE for the Reflex language

Ilya Gornev
Institute of Automation and Electrometry of the Siberian Branch of the RAS
Novosibirsk, Russia

Tatiana Liakh
Institute of Automation and Electrometry of the Siberian Branch of the RAS
Novosibirsk, Russia

***Abstract*—The process-oriented programming language Reflex is a programming language for cyber-physical systems' (CPS) control software. It is based on the formal hyperprocess model. Reflex has proved effective in industrial projects. But using Reflex is difficult due to the lack of IDE for Reflex programs. In this paper, we develop cloud desktop IDE for the Reflex language — RIDE. Modularity is the main principle of the RIDE architecture. It meets the needs of CPS software development process and allows extending the IDE functionality. Reflex IDE extensions may provide various functionalities, such as a graphical representation of the Reflex code, translators to other programming languages, debugging and verification techniques.**

Keywords—process-oriented programming, cloud IDE, web, cyber-physical systems, control software

I. Introduction

The programming language Reflex [1] suggests to develop control CPS software in process-oriented style. A CPS implies interaction of software with the physical environment [2]. Such systems are ubiquitous; for example, industrial control systems and the industrial internet of things. CPS control programs have specific properties, such as:

1. *Openness*. CPS control software interacts with an environment (external physical world) through sensors and actuators.
2. *Event-driven behaviour*. Control software reactions depends on physical environment events (facility events and operators commands).
3. System operation time is not limited.
4. *Synchronism*. Control software reacts dynamically on events on a facility.
5. *Logical parallelism*. Control software structure reflects physically parallel processes at a facility.

The Reflex language allows users to take into account the specifics of cyber-physical systems and speed up the development of CPS control software. The C-like syntax promises easy learning for programmers. Reflex has proved effective in industrial projects [3].

Despite advantages, no specialized IDE for the Reflex language is provided. Thus, the paper describes the development of the Reflex specialized IDE named RIDE.

Modern IDEs provide functionality to decrease the coding required time. Such functionality simplifies coding procedure, for example, syntax highlighting, autocompletion, code generation, etc. RIDE must include code editor functionality oriented on the Reflex syntax.

Modern IDEs integrate different tools, for example compilers and translators for different platforms, or tools for collaborative development (Visual Studio Live Share extension [4], Teletype package for Atom [5]). Current trend is to support extensibility and allow developers to create and integrate custom extensions. For this purpose, IDEs often incorporate modularity principle in their architecture, distinguishing core modules from plug-ins, and provide API for custom users' modules. Such IDEs are Visual Studio [6] and Visual Studio Code [7], Eclipse IDE [8], IntelliJ IDEA [9], etc. Plug-ins must have access to communication channels or information about the user's code. The best way to package this information is to build an abstract syntax tree (AST) of the code. Thus, the RIDE architecture must be extensible, and for this purpose it must divide core modules from the domain-specific modules (DSMs), and provide the ability to add new DSMs.

At last, modern IDEs and other applications often use web-format and allow browser access, for example GitLab Web IDE [10] and GitPod [11]. In this way, new users can try them without long download and installation. To attract new programmers to process-oriented programming and the Reflex language, we decided to develop RIDE in two versions: desktop version and web version. The web version will simplify access, and the desktop version ensures offline usage for users with slow internet connection.

Therefore, Reflex IDE must meet the following requirements:

1. The basic RIDE functionality must include the code editor oriented for the Reflex syntax and the parser that generates an abstract syntax tree (AST) of the Reflex code.
2. The RIDE architecture must divide the core module from the domain-specific modules (DSMs). The DSM support ensures the extensibility of the RIDE functionality.
3. The RIDE core must provide access to the AST for all DSMs.
4. Including new DSMs to RIDE must be possible without changing the source code of the RIDE core module.
5. RIDE must be provided in two versions: desktop version and web version.

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

978-1-6654-1498-2/21/$31.00 ©2021 IEEE

6. To test the RIDE functionality and the DSM support, the DSM that translated Reflex code to C code for the Arduino platform must be used.

The section II covers frameworks analysis; the section III describes the implemented architecture and the result user interface; the section IV outlines further development.

II. IDE FRAMEWORKS SURVEY

We analysed the IDE development frameworks that allow implementing the Reflex language as a DSL (domain-specific language). Such frameworks provide some features for DSLs, for example, syntax highlighting and autocompletion. Also, the RIDE requirements include both desktop and web versions and modular architecture. Therefore, we were looking for frameworks that provide ability to create both versions from the same source code and simplify creating modular structure.

The most widespread frameworks for DSL development are JetBrains MPS [12] and Xtext [13]. Although both are DSL development tools, their usage differs. MPS provides a projection tool, so it saves the code not as a text, but as some model instead. Therefore, DSL implemented via JetBrains MPS is not compatible with other text editors and tools. On the other hand, Xtext framework is well documented and makes it possible to incorporate implemented DSL to other tools. Moreover, Xtext artifacts are compatible with the Language Server Protocol (LSP), so it can be used in web applications. To implement the Reflex language, the Xtext framework was chosen. The Eclipse Foundation [14] owns Xtext, so the natural choice of means is to use other known Eclipse projects: Eclipse RCP [15] for the desktop version of RIDE and Eclipse RAP [16] for the web version. These projects suggest compatibility with Xtext. Disadvantages of this solution are its complexity and poor documentation. These are old, not developing frameworks with passive community. Therefore, we decided to abandon them.

Another new Eclipse project, Eclipse Theia [17], shows several advantages. Eclipse Theia is a web-IDE development framework, while eclipse RCP is built for general-purpose applications development. Theia better suits our goal and is easier to learn. Theia meets all the requirements and it allows the integration of the Xtext artifacts in the project. The desktop version of the application may be built from the same source code with the Electron framework [18]. Lastly, Theia project continues developing and its community is active.

Due to the advantages, we chose Xtext to implement the Reflex language, Eclipse Theia for the web-IDE development, and Electron for the desktop version.

III. ARCHITECTURE AND IMPLEMENTATION

A. Architecture

Theia architecture consists of extensions [19]. Extensions may be divided by two categories: frontend (run on the client side), and backend (run on the server side). Backend Theia extensions may communicate with the server operating system and other programs.

Earlier the Reflex language was implemented with the Xtext framework in the form of an Eclipse plug-in [20]. From this plug-in, we build the separate language server (LS) using the Gradle system [21]. Theia backend extension communicates with the LS via JSON-RPC protocol [22], and provides the Reflex language functionality to the editor when it edits the file with the `*.rcs` extension.

Different Reflex applications demand translation of Reflex code to different platform languages. To simplify new translators' contributions, we provided a mechanism for integration of simple DSMs (domain specific modules). A simple DSM takes the abstract syntax tree (AST) of the Reflex code, and returns a string to be shown to user. In case of translator, a DSM would generate a file with the target language code and return a path to the file.

Each DSM needs the AST of the user's code. The extension `ast-service` provides AST to other modules. It gets the AST from the language server, serializes the AST, and sends it to the DSM via the `dsm-wrapper`.

The `dsm-wrapper` extension allows the integration of new DSMs. The `dsm-wrapper`'s client part enhances the user interface with buttons for integrated DSMs. Its server part runs and manages DSMs' processes.

Fig. 1 demonstrates how the `dsm-wrapper` is built in the RIDE architecture. Each domain specific module must be implemented as a Spring application [23]. These Spring applications are packed and run in Docker containers [24]. When DSM Wrapper Server gets commands from DSM Wrapper Client, it communicates with DSMs in Docker containers via HTTP protocol. The DSM Wrapper Client is connected to commands in the tool menu of the Theia user interface. For each DSM a new command is contributed.

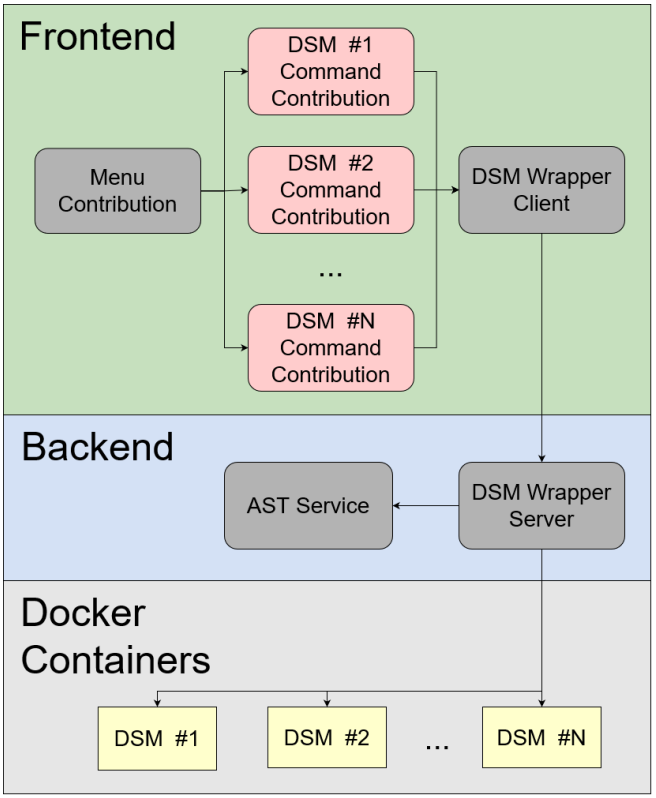

Fig. 1. The DSM integration in the architecture.

B. User Interface

Fig. 2 demonstrates RIDE syntax highlighting for the Reflex language. The implementation of the syntax

highlighting must be quick, for that reason the highlighting is not provided by the language server. Instead, syntax highlighting is implemented on the client side (no HTTP transmission involved). For quick Reflex code analysis, we used TextMate grammars [25].

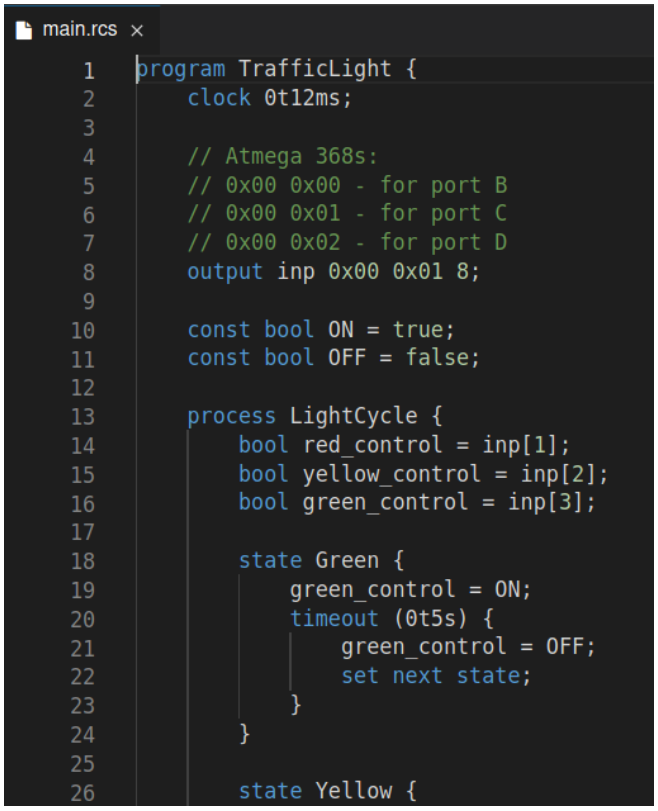

Fig. 2. RIDE syntax highlighting.

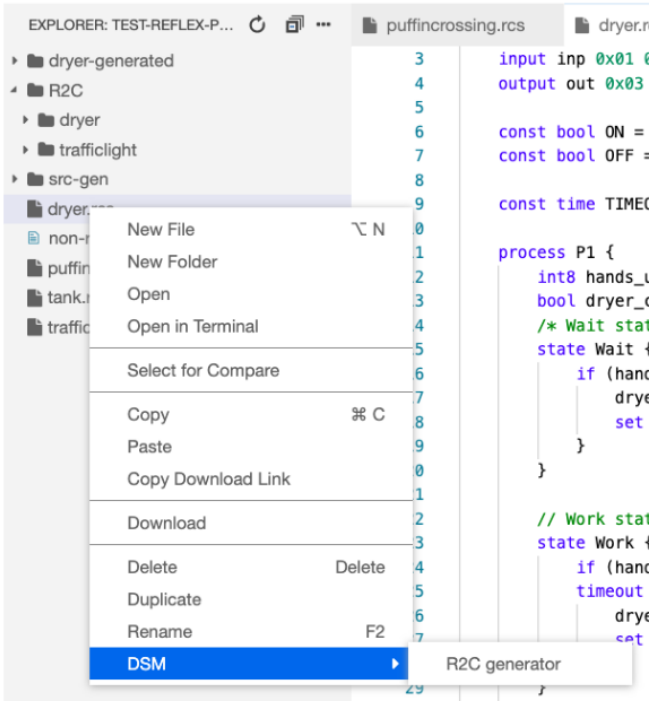

Fig. 3. The project navigation area, and the DSM menu with one DSM command.

Fig. 3 demonstrates the project navigation area, where user can see and search through project files. The DSM menu with the DSM commands is located in the context menu of the files.

Fig. 4 shows how the user sees syntax errors. The information for the code editor functionality, including autocompletion and syntax check, comes from the language server on the backend via the HTTP protocol.

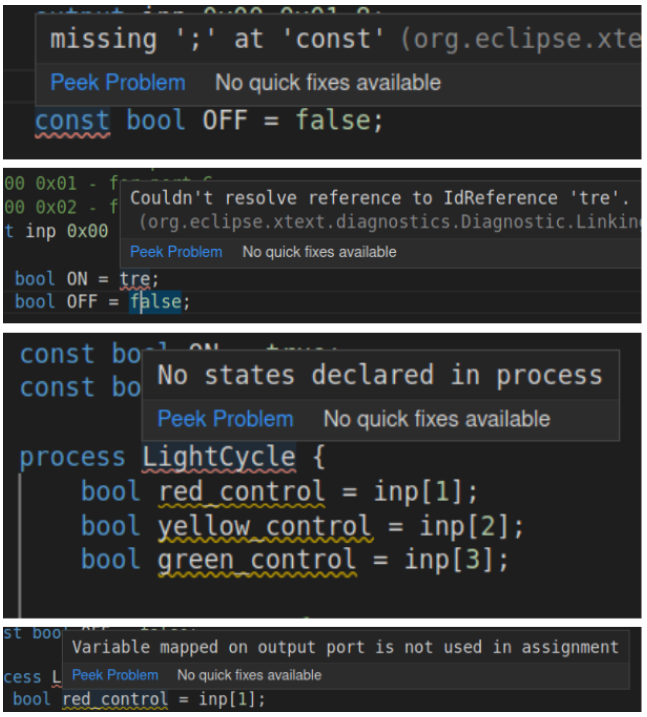

Fig. 4. Messages about syntax error.

IV. Further Development

The further development of RIDE includes three directions:

- Users management, authorization, and organizing users' workspaces using Eclipse Che.
- Development and integration of DSMs for new ways of editing Reflex programs, for example, interactive activity diagrams.
- The functionality for debugging and verification of the Reflex programs, ability to describe and emulate the controlled object.

V. Conclusion

The paper describes the development of the web-IDE specialized for the process-oriented language Reflex. RIDE project represents the result of the work, it may be run as a web-service or a cross-platform desktop application. RIDE allows to write Reflex code with syntax highlighting and autocompletion. The user can refactor Reflex code and analyse it via tree view or the "go to definition" function. RIDE automatically translates Reflex code to C code. The architecture of the project allows implementing more translators or other domain-specific modules in future.

To create RIDE, we analysed the specificity of the CPSs control algorithms, stated the requirements for the Reflex IDE and the requirements for the means. The analysis of the means for the IDE development and the DSL implementation showed the optimal frameworks for the

RIDE project. The results may be used for developing other process-oriented IDEs.

Acknowledgment

We thank the JetBrains Foundation for the charitable support of our research activity.

References

[1] I. Anureev, N. Garanina, T. Liakh, A. Rozov, and V. Zyubin, “Towards safe cyber-physical systems: the Reflex language and its transformational semantics,” IEEE International Siberian Conference on Control and Communications (SIBCON-2019) Tomsk, April 2019

[2] W. M. Taha, AE. M. Taha, and J. Thunberg, Cyber-Physical Systems: A Model-Based Approach. Cham: Springer, 2021.

[3] T. Liakh, V. Zyubin, M. Sizov “The experience of the Reflex language application for the automatization of the Large Solar Vacuum Telescope [Opyt primeneniya yazyka Refleks pri avtomatizatsii Bolshogo solnechnogo vakuumnogo teleskopa],” *(in Russian)*, Industrial control systems and controllers [Promyshlennye ASU i kontrolery], vol. 7, pp. 37-43, 2016.

[4] "Visual Studio Live Share | Visual Studio", Visual Studio. [Online]. Available: https://visualstudio.microsoft.com/ru/services/live-share/. [Accessed: 19- Feb- 2021].

[5] "Code together in real time in Atom", teletype.atom.io. [Online]. Available: https://teletype.atom.io/. [Accessed: 19- Feb- 2021].

[6] "Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio", Visual Studio. [Online]. Available: https://visualstudio.microsoft.com. [Accessed: 19- Feb- 2021].

[7] "Visual Studio Code - Code Editing. Redefined", Code.visualstudio.com. [Online]. Available: https://code.visualstudio.com/. [Accessed: 09- Feb- 2021].

[8] "Eclipse IDE 2020-12 | The Eclipse Foundation", Eclipse.org. [Online]. Available: https://www.eclipse.org/eclipseide/. [Accessed: 09- Feb- 2021].

[9] "IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains", JetBrains. [Online]. Available: https://www.jetbrains.com/idea/. [Accessed: 19- Feb- 2021].

[10] "DevOps Platform Delivered as a Single Application", GitLab. [Online]. Available: https://about.gitlab.com/. [Accessed: 09- Feb- 2021].

[11] "Gitpod - Dev environments built for the cloud", Gitpod.io. [Online]. Available: https://www.gitpod.io/. [Accessed: 09- Feb- 2021].

[12] "MPS: The Domain-Specific Language Creator by JetBrains", JetBrains. [Online]. Available: https://www.jetbrains.com/mps/. [Accessed: 09- Feb- 2021].

[13] S. Efftinge and M. Spoenemann, "Xtext - Language Engineering Made Easy!", Eclipse.org. [Online]. Available: https://www.eclipse.org/Xtext/. [Accessed: 09- Feb- 2021].

[14] M. Milinkovich, "Eclipse Foundation | The Eclipse Foundation", Eclipse.org, 2005. [Online]. Available: https://www.eclipse.org/org/foundation/. [Accessed: 09- Feb- 2021].

[15] "Rich Client Platform - Eclipsepedia", Wiki.eclipse.org. [Online]. Available: https://wiki.eclipse.org/Rich_Client_Platform. [Accessed: 09- Feb- 2021].

[16] "Remote Application Platform (RAP)", Eclipse.org. [Online]. Available: https://www.eclipse.org/rap/. [Accessed: 09- Feb- 2021].

[17] "Theia - Cloud and Desktop IDE Platform", Theia-ide.org. [Online]. Available: https://theia-ide.org/. [Accessed: 09- Feb- 2021].

[18] "Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS.", Electronjs.org. [Online]. Available: https://www.electronjs.org/. [Accessed: 09- Feb- 2021].

[19] "Theia - Cloud and Desktop IDE Platform", Theia-ide.org. [Online]. Available: https://theia-ide.org/docs/authoring_extensions. [Accessed: 09- Feb- 2021].

[20] A. Bastrykina, “Refactoring of the Reflex translator based on the automatic parser generation [Refaktoring translyatora yazyka Refleks na osnove avtomaticheskoy parser-generatsii],” *(in Russian)*, Materials of the 58-th International Scientific Student Conference ISSC-2020 [Materialy pyatdesyat vosmoy Mezhdunarodnoy Nauchnoy Studencheskoy Konferentsii MNSK-2020], Novosibirsk, p. 138, 2020.

[21] "Gradle Build Tool", Gradle. [Online]. Available: https://gradle.org/. [Accessed: 09- Feb- 2021].

[22] "JSON-RPC", Jsonrpc.org. [Online]. Available: https://www.jsonrpc.org/specification. [Accessed: 09- Feb- 2021].

[23] "Spring makes Java simple.", Spring. [Online]. Available: https://spring.io/. [Accessed: 09- Feb- 2021].

[24] "Empowering App Development for Developers | Docker", Docker. [Online]. Available: https://www.docker.com/. [Accessed: 09- Feb- 2021].

[25] "Language Grammars — TextMate 1.x Manual", Macromates.com. [Online]. Available: https://macromates.com/manual/en/language_grammars. [Accessed: 09- Feb- 2021].