2023 IEEE XVI International Scientific and Technical Conference Actual Problems of Electronic Instrument Engineering (APEIE) | 979-8-3503-3088-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/APEIE59731.2023.10347731

Towards Collaborative Coding in RIDE Web IDE

Ilya A. Gornev
Cyber-physical Systems Laboratory
Institute of Automation and Electrometry
of the Siberian Branch of the RAS
Novosibirsk, Russia
i.a.gornev@gmail.com

Vladimir V. Bondarchuk
Cyber-physical Systems Laboratory
Institute of Automation and Electrometry
of the Siberian Branch of the RAS
Novosibirsk, Russia
v.bondarchuk@g.nsu.ru

***Abstract*—RIDE is an integrated development environment (IDE) suited for the Reflex language, specifically designed for creating control algorithms for cyber-physical systems. RIDE is based on Eclipse Theia and a host program running independent instances for each user. However, it does not provide collaborative experience. Eclipse Theia is designed for single-user; therefore, collaborative programming is hampered by security and ease of use issues. This paper outlines a server program architecture to add collaborative abilities to otherwise single-user web applications. The suggested approach is application independent. It is applicable to other single-user applications, making it possible to deploy them in the cloud as web services. The paper provides a detailed analysis of the problem, as well as the tools and frameworks used for implementing this design with RIDE. This implementation makes use of Docker for containerization, Python and Flask for host program, Google Auth for user authentication, and SQLite for the database.**

Keywords—CPS, process-oriented programming, Reflex, cloud IDE, web application, multi-user management

I. Introduction

Increasingly, intelligent systems that use wireless communication and network technologies that provide information exchange between the physical and cybernetic world are penetrating our lives in order to control physical objects using software and sensors. Such systems are called cyber-physical (CPS) [1]. Examples of CPS appear everywhere, for example: smart cities, robotic warehouses and production lines, smart homes, automatic vehicles, etc.

The process-oriented programming paradigm is well suited for creating control algorithms for CPS, which is why the Reflex language was developed [2]. In the heart of the language lies a hyperprocess model based on the concept of Finite State Automaton, and it uses a C-like syntax, making it easy for beginners to learn. Reflex has been proven to accelerate the development of CPS control software while being effective in actual industrial projects [3].

RIDE (Reflex IDE) is an integrated development environment created exclusively for the Reflex language, but it is also flexible enough to potentially support other languages in the future [4]. Currently, RIDE is a single-user web application that has not been publicly available yet.

The article [5] describes a server architecture for allowing multiple users to access RIDE independently at the same time. However, there is a need for implementing collaborative mode for RIDE. This would simplify developing complex algorithms, encourage pair programming and enable new ways of teaching process-oriented programming.

II. Analysis

A. Problem Analysis

RIDE is built on the Eclipse Theia [6]. Theia is a general purpose extensible IDE functioning as a web application. However, Theia developers intended for applications based on Theia to be hosted locally for a team, not for a wide audience. This contradicts RIDE's main goal – to invite new programmers to try process-oriented programming and the Reflex language specifically.

In [5] we proposed a server host program architecture to host RIDE for many users. This solution divides users' workspaces and solves some of security problems. However, while such a host program allows the server to serve many projects at the same time, it does not allow several users to work on the same project. Basically, it creates a workspace for each user. This is an unacceptable practice for programming nowadays, because it is often a teamwork.

Thus, there is a need to implement collaborative mode. The solution must be application-independent because this would allow people to enhance collaborative experience for many other applications. In addition, it must not demand editing original application source code because in some cases the source code may not be available. Alternatively, as in the case with Eclipse Theia, the source code may demand an expert of the specific field to be edited safely.

B. Collaborative Experience in Other Applications

In order to formulate design goals, we performed an analysis of similar applications. We selected collaborative applications mainly by their popularity.

This work was supported by the Russian Ministry of Education and Science, project no. 122031600173-8.

Google Docs [7] is the most popular service for collaborative work on documents. It allows users to create, edit and share documents. When shared, documents may be watched or edited by others. Google Docs allows editing documents simultaneously, while each user immediately sees all changes made. It also allows managing users` access rights, distinguishing between editing, watching and commenting.

CodeCollab.io [8] is a real-time collaborative coding platform. Similarly, to Google Docs, it allows users to edit code in the same file simultaneously and to manage access rights. Moreover, CodeCollab.io provides a chat functionality to simplify team members' communication.

Replit [9] is another teamwork platform. Its main advantage over similar projects is managing teams. In addition, Replit provides integration with Git [10] and GitHub [11].

IntelliJ IDEA [12] is a desktop IDE for such programming languages as Java [13], Kotlin [14], Python [15] and others. Besides typical IDE functionality, IntelliJ IDEA provides collaborative mode, allowing team members to work on the same project simultaneously. In this mode, they can see each other's changes in real time.

C. Design Goals

As described above, the host program must be application-independent. It is impossible to implement a fully application-independent program, but the amount of limitations should be minimal.

The analysis demonstrated that the most prominent feature for collaborative work is simultaneous real-time editing. Moreover, such feature would simplify teaching, which is an important aspect to popularize new programming languages such as the Reflex language.

For collaborative work, the ability to share a project is necessary. Currently, RIDE creates a workspace for each user and deletes the workspace after the session is closed. The user is responsible for downloading his work. However, in collaborative work, it would not be obvious who should download project files. Thus, it is a necessity to implement storage for users` project files on the server.

Consequently, each user should be able to participate in many projects. Moreover, since the server must store users` projects, it must provide project management functionality.

Last but not least, the server should demand users` authorization/authentication. It is necessary for implementing team members` rights management, which is also seen in almost any collaborative coding platform.

III. Server Architecture

After reviewing the analysis and recommendations outlined in [5], it was determined that implementing our own module would help us achieve our objectives. This solution has the added benefits of enhancing the project as user numbers increase, optimizing the application for all developers, and enabling its use in comparable projects due to the availability and clarity of the source code.

A. Architecture of the Administration Module

To implement the module, we will develop a web server application to allow new users to register and manage their projects. Fig. 1 illustrates the proposed architecture of the module. It consists of a network server (3) and a database (4) that stores all pertinent information about users and their containers. When users (1 and 2) log into the server, they will be able to create new application instances (5-7) and manage their individual projects. The local server (8) will store and execute the database, module, and application instances.

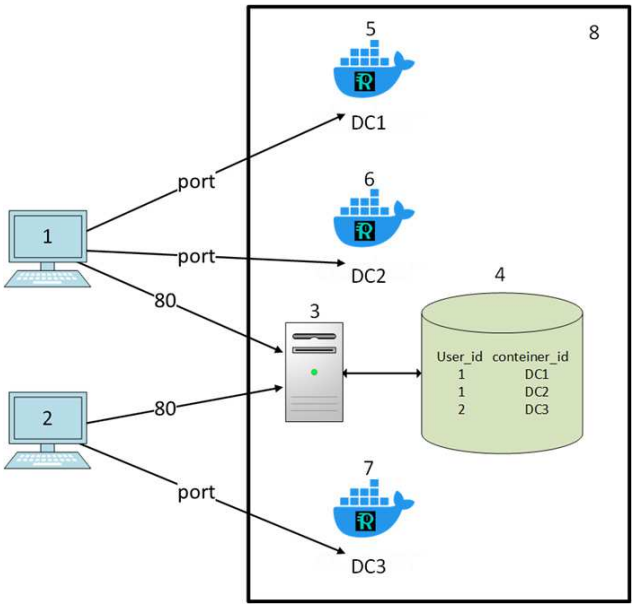

Fig. 1. Module architecture scheme: 1 and 2 – users; 3 – network server; 4 – database; 5-7 – application instances in Docker container; 8 – local server.

Fig. 2 shows the algorithm of the module. When users connect, they can register or log into their account, which involves working with the database. After passing all checks, users enter their profile and gain access to their existing projects, which they can interact with.

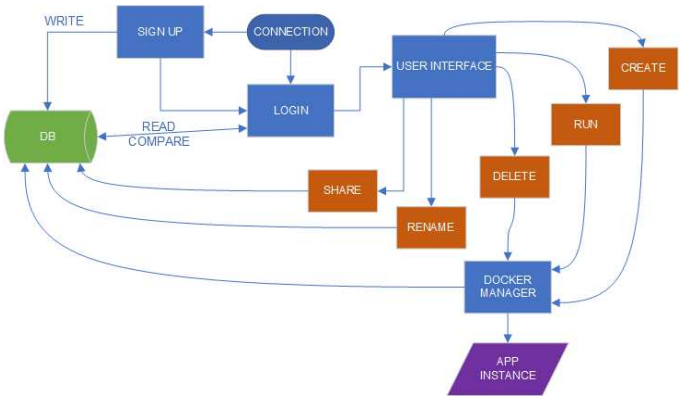

Fig. 2. Algorithm of the module.

B. Application Instance

When a user creates a new workspace, his application instance should be isolated from other instances. The server achieves this isolation through containerization, which allows

you to properly restrict and configure the file system of a particular project. Dangerous actions in one container will not affect other containers and the server.

Containerization is an easy alternative to virtualization [16]. It involves encapsulating an application in a container with its own operating environment. Thus, instead of installing an operating system for each virtual machine, containers use the host OS. The container shares the host OS kernel with other containers, and the shared part of the OS is read-only. Thus, containers have a small weight, so it is possible to deploy several containers on one server (or virtual machine). This allows us not to allocate the entire server for one application and use only one operating system for maintenance.

Scaling becomes fast and simple, without the need to increase server space. This gives good opportunities for growth and optimization of the system in the future, which is undoubtedly an advantage [17].

IV. Frameworks Survey

Having developed the architecture, we faced the question of choosing the necessary implementation tools.

A. Containerization

Docker [18] is the predominant and widely used technology for container creation, making it the go-to for packaging applications and managing the containerization process. To containerize with Docker, the initial step involves creating obligatory images. These pre-defined templates contain instructions for creating a container using a specific Dockerfile file outlining all necessary dependencies. This technology efficiently minimizes server resource allocation for standalone applications, increasing the possibility of accommodating more users on the server. The containers also lay out a vital foundation for enhancing the system through scaling.

B. Web-Server

The server must manage incoming connections and application instances, as well as handle user requests. To implement these tasks, it must meet the following requirements:

1) detect incoming connections,
2) process POST and GET requests,
3) redirect users to the pages they need,
4) get parameters from configuration files,
5) register new users,
6) authenticate users,
7) create containers,
8) start containers,
9) remove containers,
10) get container references,
11) ability to rename containers,
12) stop containers corresponding to closed connections.

The framework for implementing the host program should provide tools for these tasks.

C. Flask

Flask [19] allows you to store user-specific data between requests using "sessions". The session object works as a dictionary, but at the same time allows you to track changes. Sessions are stored in the browser and work on a cookie-like principle. However, unlike regular cookies, Flask cryptographically marks session cookies. This means that everyone can see the contents of cookies, but cannot change them without having a secret key for signing. Once session cookies are configured, each subsequent request to the server confirms the authenticity of the cookie using the same secret key. If Flask fails to do this, then its content is rejected and the browser receives new session cookies. Thus, Flask guarantees that the user without a secret key cannot change the contents of the session cookie.

D. Flask-Login

The task of user authentication is widespread in the design of web servers; therefore, the `Flask-Login` [20] module is provided to ensure user session management. It saves the ID of the active user in the session and makes it easy to log in and out of the account, and helps protect sessions from cookie theft.

E. Google Auth

We added a popular and convenient authentication via Google account, using the `google-auth-oauthlib` library [21] for this. After the users enters registration data, the server hashes it and stores in the database. Then, the server will compare log in parameters with the information in the database.

F. Database

The implementation of the tasks and ensuring the stable operation of the server requires a database. The server database stores two tables: `Users` and `Containers`. Fig. 3 demonstrates the relationship between the tables, as well as the attributes of the tables.

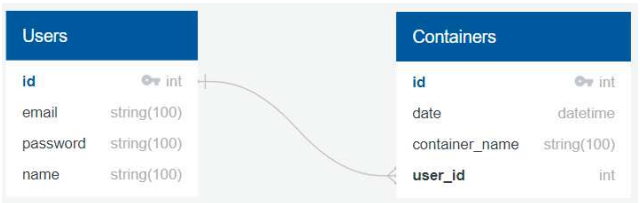

Fig. 3. The scheme of relations in the database.

Since the application is implemented with the Flask framework and does not require a large number of tables or complex operations on them, we decided to use the `SQLAlchemy` ORM [22], which is widely used in the development of web servers on Flask.

Since the project is a research one and currently does not imply a large number of users, we decided to use the SQLite [23] database. Nevertheless, this choice does not limit the further work and development of the project in any way, because the ORM separates the application from the database implementation details, making it easy to migrate to the desired database in the future.

V. Implementation

We implemented a host program using the proposed architecture. Its design allows combining it with many different applications besides RIDE. The main limitations are that the application must communicate with the user via a single port and log opening connections and closing connections.

To pack the original application in the Docker container, it is necessary to provide a Docker image file with configurations. In [5] we described a docker container configuration for RIDE. The same configuration is fully applicable for the current work.

Fig. 4 demonstrates the home page, the login page and the project management page. After the button Run on the project management page is pressed, the user is redirected to the RIDE interface.

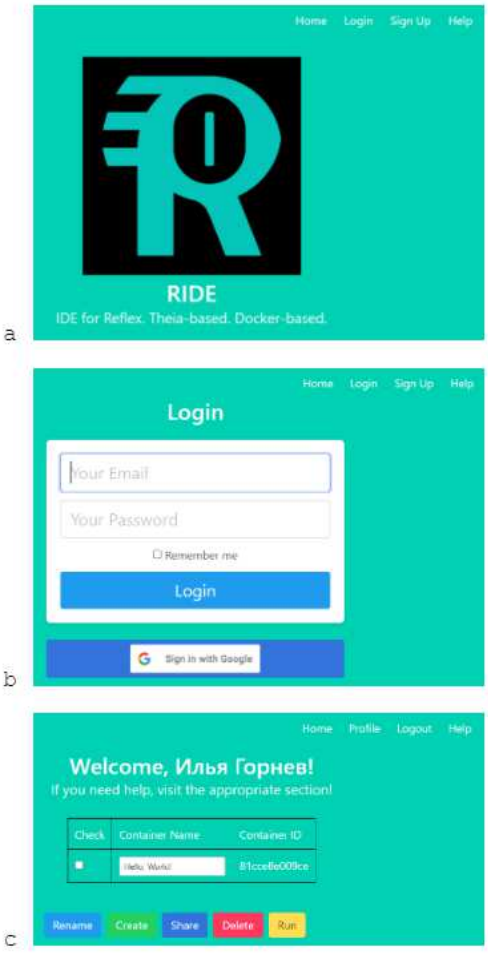

Fig. 4. (a) home page; (b) log in page; (c) project management page.

The design goals described in chapter II are fully met. The host program can use many other applications instead of RIDE. Users may share projects and edit their files in a real-time manner. This application stores projects on the server, so users do not need to download them anymore. Each user can be owner of many projects and manage them on the project management page. And each user must be logged in.

VI. Discussion

The proposed architecture is application-independent, so it can be used in combination with many different single-user applications. The requirements for the original applications are as follows:

- The original application must be a web application.
- It must listen to a single port for the input.
- It must log to the output stream the events of a new connection and breaking the existing connection. The host-program analyses this information to determine when the container may be closed.

The implemented host program provides such features as authorization/authentication, project management, and collaborative real-time editing. These are the most often met features in collaborative services.

By combining the proposed host-program with RIDE, we were able to enhance RIDE with a collaborative mode without needing to edit RIDE source code. The further development involves the following directions.

Sharing projects feature is to be polished. One of the ways to make it more convenient is to create a page with references to every project where the current user is invited.

Currently, there is no ability to manage users` access rights; each invited user can fully edit the project. We are planning to implement access rights management in the future.

Chat and commentary features are often seen in similar software. We are also planning to add them.

VII. Conclusion

We proposed a host program architecture for enabled collaborative mode for otherwise single-user applications. We formulated design goals based on the analysis of the similar software. The resulting host program meets the design goals and provides the most expected functionality.

We proved the approach by building a host program for RIDE web IDE. In the result, RIDE was enhanced with a collaborative mode without needing to edit its source code. The results may be used to add collaborative features to other single-user web applications.

References

[1] W. M. Taha, AE. M. Taha, and J. Thunberg, Cyber-physical systems: a model-based approach. Cham: Springer, 2021.

[2] 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.

[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] I. Gornev and T. Liakh, "Ride: Theia-based web ide for the Reflex language," 2021 IEEE 22nd International Conference of Young Professionals in Electron Devices and Materials (EDM), pp. 503–506, 2021.

[5] I. A. Gornev, V. V. Bondarchuk, and T. V. Liakh, "Towards multi-user mode in ride web-IDE," 2022 IEEE 23rd International Conference of Young Professionals in Electron Devices and Materials (EDM), 2022.

[6] "Theia – Cloud and Desktop IDE Platform," Eclipse Foundation, 2023. Accessed: Sep. 25, 2023. [Online]. Available: https://theia-ide.org/

[7] "Google Docs: Online Document Editor," Google, 2023. Accessed: Sep. 25, 2023. [Online]. Available: https://www.google.com.mt/intl/us/docs/about/

[8] Carl Ian Voller, "Goodbye," CodeCollab Pte Ltd, May 31, 2023. Accessed: Sep. 25, 2023. [Online]. Available: https://codecollab.io/

[9] "Replit: the collaborative browser based IDE – Replit," Replit, Inc, 2023. Accessed: Sep. 25, 2023. [Online]. Available: https://replit.com/

[10] "Git." Accessed: Sep. 25, 2023. [Online]. Available: https://git-scm.com/

[11] "GitHub: Let's build from here," GitHub, Inc., 2023. Accessed: Sep. 25, 2023. [Online]. Available: https://github.com/

[12] "IntelliJ IDEA – the Leading Java and Kotlin IDE," JetBrains s.r.o., 2023. Accessed: Sep. 25, 2023. [Online]. Available: https://www.jetbrains.com/idea/

[13] "Java," Oracle, 2023. Accessed: Sep. 25, 2023. [Online]. Available: https://www.java.com/en/

[14] "Kotlin programming language," JetBrains s.r.o.. Accessed: Sep. 25, 2023. [Online]. Available: https://kotlinlang.org/

[15] "Welcome to Python.org," Python Software Foundation, 2023. Accessed: Sep. 25, 2023. [Online]. Available: https://www.python.org/

[16] "Virtualization vs Containerization," Baeldung. Accessed: Sep. 25, 2023. [Online]. Available: https://www.baeldung.com/cs/virtualization-vs-containerization

[17] "What are Containers? – Benefits and Use Cases," NetApp, 2023. Accessed: Sep. 25, 2023. [Online]. Available: https://www.netapp.com/devops-solutions/what-are-containers/

[18] "Docker: Accelerated, Containerized Application Development," Docker Inc., 2023. Accessed: Sep. 25, 2023. [Online]. Available: https://www.docker.com/

[19] "Welcome to Flask — Flask Documentation (2.3.x)," Pallets, 2010. Accessed: Sep. 25, 2023. [Online]. Available: https://flask.palletsprojects.com/en/2.3.x/

[20] "Flask-Login — Flask-Login 0.7.0 documentation." Accessed: Sep. 25, 2023. [Online]. Available: https://flask-login.readthedocs.io/en/latest/

[21] "Google-auth-oauthlib — google-auth-oauthlib 0.4.1 documentation," Google, Inc., 2017. Accessed: Sep. 25, 2023. [Online]. Available: https://google-auth-oauthlib.readthedocs.io/en/latest/

[22] "Flask-SQLAlchemy — Flask-SQLAlchemy Documentation (3.0.x)," Pallets, 2010. Accessed: Sep. 25, 2023. [Online]. Available: https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/

[23] "SQLite Home Page,", SQLite Consortioum, Sep. 8, 2023. Accessed: Sep. 25, 2023. [Online]. Available: https://www.sqlite.org/index.html