

# **COLLECE 2.0: A DISTRIBUTED REAL-TIME COLLABORATIVE PROGRAMMING ENVIRONMENT FOR THE ECLIPSE PLATFORM**

Santiago Sánchez, Miguel A. Redondo, David Vallejo, Carlos González and Crescencio Bravo  
*Universidad de Castilla-La Mancha*  
*Paseo de la Universidad, 4, Ciudad Real, 13071, Spain*

## **ABSTRACT**

Collaboration with other users to solve programming problems has an effect on the speed of development and the final software quality. This is achieved through techniques like pair programming when users are co-located, or otherwise through real-time remote collaboration. Existing work for remote working on a programming project has failed to address the integration of new emerging technologies, like augmented reality. These new technologies would provide a way to improve the final user learning experience if integrated in a working programming environment. This paper describes the implementation of COLLECE 2.0, a plug-in for the Eclipse platform which provides a distributed real-time collaborative programming environment, and integrates support for augmented reality and mobile device capabilities. We discuss two specific use scenarios where the solution would work, and how the implemented awareness mechanisms improve the user experience. The paper closes with the conclusions, limitations and future work, and a first sign on what on-going works would look like.

## **KEYWORDS**

Collaboration, real-time edition, awareness, augmented reality, distributed development, eclipse

## **1. INTRODUCTION**

Distributed real-time collaborative programming has been one important discussion topic for a long time. The rise of tools allowing users to effectively achieve setting such an environment and the improvements made over highspeed networks demonstrates this. However, these new tools try to go a step further and build a complete system where users are presented with an integrated solution for a completely collaborative programming environment. But they do not achieve it, since they base their efforts on the implementation of sharing programming projects, or individual files to be more accurate, among collaborators (Bravo et al., 2013). For example, prior work like CoEclipse focuses mainly on topics like consistency maintenance, i.e. syntax and semantic consistency, but then again not growing on what's been built (Fan and Sun, 2012a, Fan and Sun, 2012b), thus being an incomplete but promising work for a whole collaborative programming experience.

A common scenario for real-time collaborative programming are Massive Online Open Courses (MOOCs), which state remote work is a fact, and so require students to collaborate with each other to successfully follow up the course. A complete environment that fulfills these requirements and also extends on other features would be appropriate in such context. In fact, existing studies (Shen and Sun, 2000) confirm that real-time collaborative programming is capable of accelerating the progress of problem-solving, creating a better design and shorter code length, and thus improving the quality of software projects.

The presented work builds over prior research named COLLECE (COLLaborative Edition, Compilation and Execution of programs), a project aimed to guide students in solving programming problems through distributed pair programming (Bravo et al., 2013). This new proposed version improves over the existing behavior and extends its features to achieve a complete real-time collaborative programming environment. These new features go beyond real-time editing problems and try new approaches to collaborative programming, like graphic visualization, context awareness, or problem solving. Also, it accomplishes, along with the common workspace awareness elements (Gutwin and Greenberg, 1995), that a real-time distributed

groupware system should accomplish, i.e. user participation (presence), user location, activity level, user actions, user intentions, user changes, objects used, etc.

In this paper we will discuss the new COLLECE 2.0 as a working tool and a programming learning environment, its features, basis, reworked stuff, common use scenarios, and future development. This conforms to a programming learning environment with emerging and interactive technologies, via augmented reality and touching capabilities, which fulfills new needs in countries to educate students in the programming sciences scope. In section 2 we discuss some related and past work the solution is based on. Then, in section 3 the project is introduced from a technical point of view, where an overview of the main modules and how they work together is presented. After that, in section 4, some common use scenarios are announced. Finally, in section 5, we address the findings drawn from the research.

## 2. RELATED WORK

Users interested in working through real-time collaborative programming may use some existing solutions that work at some degree; as the previously introduced CoEclipse, which integrates with the Eclipse IDE as a set of plugins so the user experience is the same as if he or she was using the environment without any collaboration features.

CoEclipse mainly invests its efforts in granting syntactic and semantic consistency among users. It follows a centralized client/server network model where the document editions are made locally and then applied remotely by operational transformations (Ellis and Gibbs, 1989). These operations were used in the Jupiter system (Nichols et al., 1995) as a mean to solve conflicting editions so that it combine both client and server operations to find one which allows the client and server to reach the same final state.

Figure 1 shows an example of operational transformation, where both clients start out with the same document state (i.e. the “abc” characters). If the clients edit that text concurrently, so client 1 inserts the ‘d’ character at position one, and client 2 deletes the character at position two, an inconsistency would arise since client 1 ends up with the “adc” characters and client 2 ends up with the “ab” characters, thus obtaining conflicting copies on both clients. In order to solve these concurrent editions, clients have to transform the received operations from the other client. In our example, client 1 should have to transform the deleted operation coming from client 2.

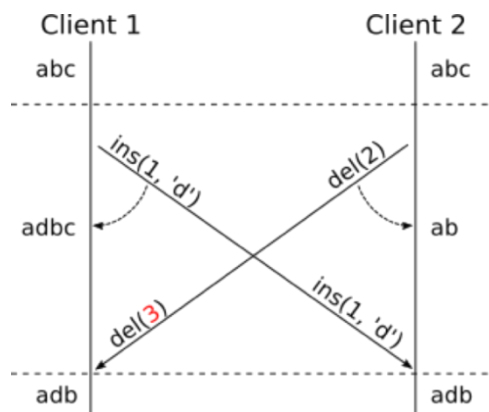


Figure 1. Operation transformation process –client 1 inserts character ‘d’ in position 1 of “abc” string, while client 2 deletes the “b” character at his or her local document; an operational transform is then needed for client 1 to have a synchronized copy with the other client

The algorithm implemented for operational transformation alongside other specific improvements, stands its basis on related research such as Apache Wave, originally developed by Google on its work centered on the Jupiter system, whose technology was later used on its Google Docs cloud applications (Wang et al., 2010).

In the same trail of Eclipse based solutions, Saros (Salinger et al., 2010) stands out for the ecosystem it offers. It has been developed actively since 2008 as a complete solution for real-time collaborative programming. It implements the same operational transforms algorithm to handle concurrent editing, but it also offers awareness mechanisms, such as presence in the way of chat and connected members, or workspace by showing the files users are editing. Visual Studio also includes the possibility to extend its environment with plugins, specifically with the VS Anywhere one, which endows the environment with collaboration capabilities. It offers a complete solution for project sharing and chat communication, and has reached a mature state ready to be used by final users.

On the other hand, the arise of web applications have emerged some cloud based tools, and so a couple of them exist. The most popular ones are Kobra (<https://kobra.io>), CodeShare (<https://codeshare.io>), or Cloud9 (<https://codeshare.io/>) (de Carvalho, 2015). All of them allow document sharing with some syntactic sugar like colours, and some communication methods like chat or video streaming. However, their IDE features are limited, since no workspace concept, autocompletion, or refactoring tools are provided; only a basic support for some interpreted languages like Python, or web development ones.

Other solutions like Floobits (<https://floobits.com>) provide a set of plugins for users to install in their favourite text editor, like Atom, Emacs, or VIM; but again, these editors lack IDE capabilities for languages like Java, or C++ (Thodi and Fujita, 2016).

A more traditional environment goes through the use of a setup of an online virtual private server with a terminal multiplexer (e.g. GNU Screen, or tmux) and a command line interface editor (e.g. vim, Emacs, or nano) (McDonnell, 2014). Also, a graphical user interface experience may also be achieved by just using Emacs and its make-frame-on-display command to connect to a remote X display. Again, these last setups do not provide common IDE features but a pair programming environment.

Another set of tools exist like CodeBunk (<https://codebunk.com/>) aimed for recruiters to ask programming questions to the candidate while they see how the interviewed works in the shared editor.

## 2.1 COLLECE

The work in this paper is born from a prior version named COLLECE (Bravo et al., 2013). It supports users joining to sessions where they can work in a shared document (i.e. a java source file for example). Sessions live in a remote server managed by a permission system based on users' whitelists, and handle documents which can be retrieved by client instances and then saved again to the server copy. The project was implemented as a multi-platform standalone Java Web Start (JavaWS) application, so users can access it through a web page.

Once users are logged, they can collaborate in the document in a variety of ways. The collaborative and awareness mechanisms the system supports are varied:

- Session panel with state information.
- Document locking for synchronous editing in a turns system way.
- User stats for further study.
- Multiple cursors identifying multiple users.
- History of past actions.
- Sounds.
- Chat window.
- Execution and compilation of code request functions.

Users may work through COLLECE by loading a specific problem and associated source code file, and then sharing it to other session logged members. Once a document is shared, a user may request his or her edition turn to make some changes and then grant the edition turn to other member. Once the code is changed according to the problem needs, a user may start a compilation and execution poll so other members are in agreement to run such actions. Program output is shared to the users through the console view.

The next section introduces to the reader what improvements and reworks are made in the new COLLECE 2.0, comparing to the previous version and other existing works.

### 3. ARCHITECTURE

COLLECE 2.0 builds its architecture over a modular set of views, editors, and wizards working as an Eclipse plugin deployed through an update site. This method of releasing plugins allows users to easily install them via the Eclipse package manager. Also, users without an Internet connection may install the plugin by simply copying to the Eclipse install directory through the file manager.

Using a modular approach allows the plugin to scale in future work fitting a cohesive and decoupled architecture. This way users may employ specific features of the environment without taking care of the other capabilities, or extended by the programmer who wants to grow up over what is done.

The main aim of COLLECE 2.0 is to provide a distributed real-time collaborative programming experience to the users. Thus, some features of the prior version have been improved or simply just removed. For example, we are referring to the turn request mechanisms to edit, compile, and run the shared program that has been reworked as an additional tool to be optionally used as a new feature.

The system distinguishes two entities, server and clients. The first one handles context information, i.e. sessions holding shared projects information, users' data, and information about the server itself. It is also in charge of keeping database persistence (based on Apache Derby), and synchronizing shared data among clients. On the other hand, clients are actual users connected to a common server that want to collaborate remotely in a programming project.

In order to allow clients to start collaborating, they first must to signup for the server to create their member data. These data are used as a way for the server and other members to identify them, using information like user name, email, or photos. The only mandatory field the user is requested to complete is its email, just for being unique within all the saved data; if it already exists, member creation is forbidden.

Clients may act as a server if they want to. COLLECE 2.0 allows creation of server instances through private networks or the Internet, so we achieve a decentralised network architecture with individual nodes composed of a server and different connected clients. Figure 2 shows a visual representation of this kind of network architecture.

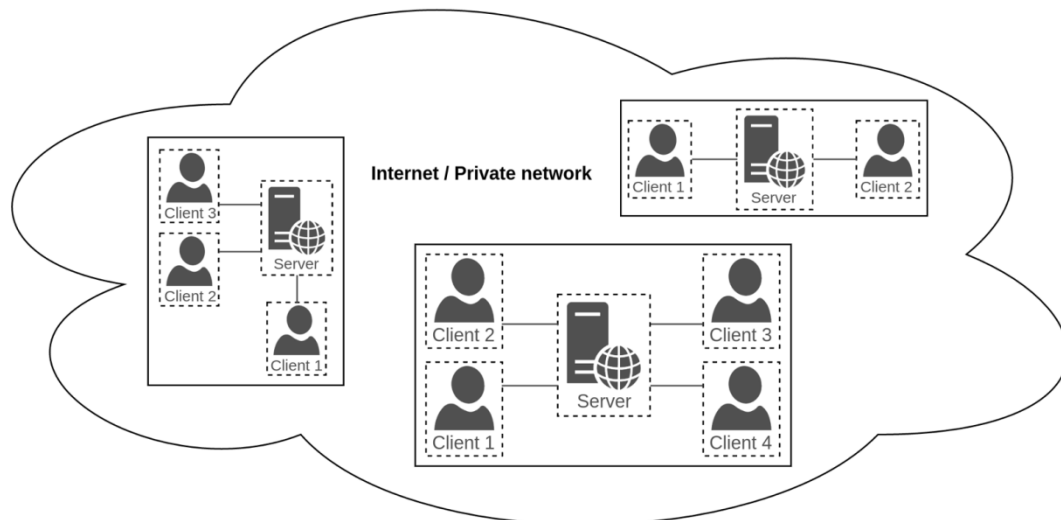


Figure 2. COLLECE 2.0 network architecture; servers live among existing others with their connected clients isolated

If the user is already registered in the server, he can log in and choose to join an existing session or create one. Sessions allow the server to differentiate collaborative workspaces. They store the list of allowed members to join, the project address repository, the duration of days the session can exist, and whether it is private, (so only allowed members can join), or publicly available. Once the client is connected to the server, he or she is provided with a copy of the repository to start working along with the other members. Earlier versions did not allow final users to behave as servers, therefore it depended on a previously setup server to work on some reachable host for allowing other users to use COLLECE.

The distributed real-time editing allows members to modify the shared project with no delays. It is achieved by applying the previously discussed operational transforms. The implementation is based in the one provided by the Sync module in the Eclipse Communication Framework (ECF). COLLECE 2.0 network communications are based all over ECF. Different channels are set up following a publish-subscription pattern where clients subscribe to the channels published by the server. This way, the system holds channels for remote methods invocations, document changes, textual/audio/video chat, shared whiteboards, augmented reality data streams, and awareness mechanisms.

Document changes are registered locally per user on every editor change, and then are sent to the server, where they are applied via operational transformations to its project copy. Once the change is valid, it is then relayed to the other clients through the document changes ECF channel (Figure 3.a). Also, to ensure document consistency, a background service keeps track of the project copy in the server periodically, and compares it to the existing ones in clients, restoring client copies in case of divergences exist.

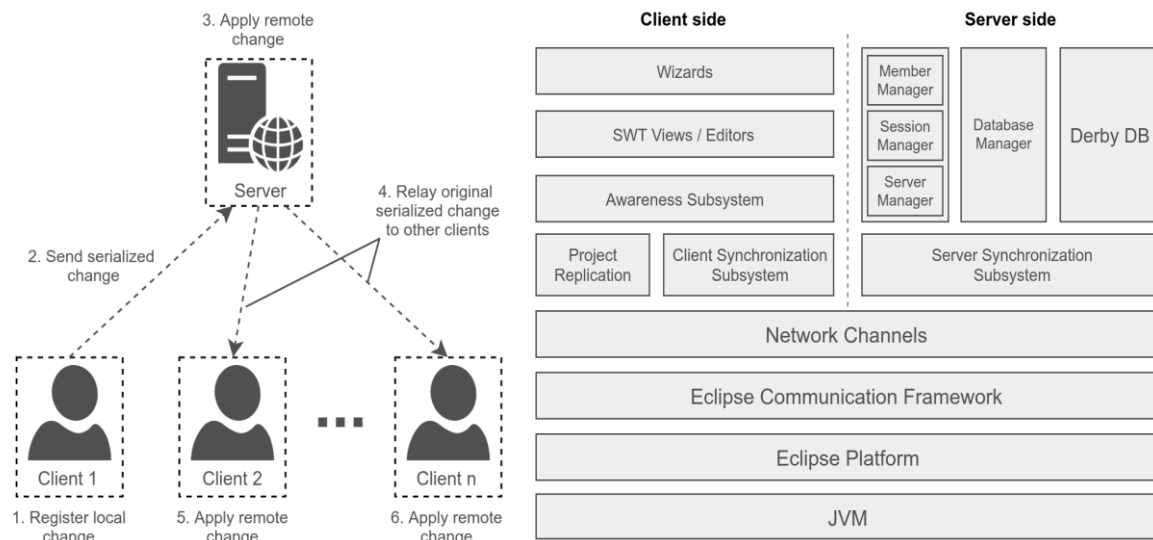


Figure 3. On left, (a) the distributed real-time editing workflow; on right, (b) a general overview of COLLECE 2.0 modules from a client/server perspective

Awareness mechanisms allow a working user to be informed about the other members actions. COLLECE 2.0 provides support for these mechanisms through:

- Multiple coloured cursors stating where the other members are editing.
- A panel displaying information from connected members.
- Displaying which regions of code are marked for read-only.
- A chat to talk with other team members.
- A console to show the output of shared command line interface programs executions.
- A shared whiteboard for members to sketch drawings.
- A following mode to see real-time specific user editions.
- A shared augmented reality view to improve the learning experience.
- A shared view to display information about mobile devices executions.

In brief, Figure 3.b shows a summary overview of the COLLECE 2.0 modules.

## 4. USE SCENARIOS

COLLECE 2.0 is intended to be used as a programming learning tool for students. In this section we introduce two common use scenarios where different group of students are provided with a COLLECE 2.0 environment to carry out a programming problemsolving task.

## 4.1 Pair Programming Classical View

Pair programming allow two users to work in a task. While one of them is working on the actual task, the other one is reviewing the other's work and thinking about solutions to the problem. COLLECE 2.0 fulfills this scenario by providing pair programming mechanism to the users.

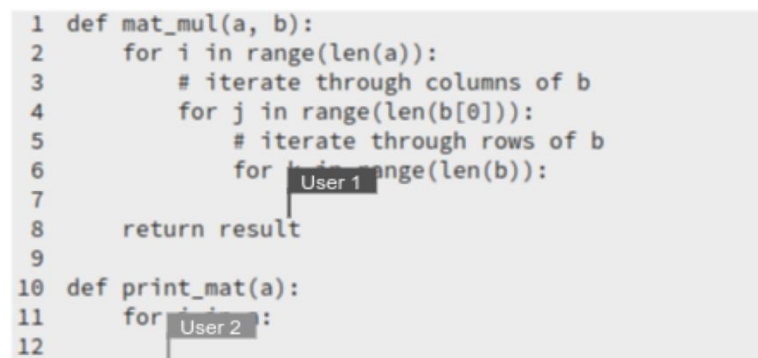
Once both users are joined to the same session, one of them would take the observer role, and the other the driver role. There are two ways to achieve this behavior using COLLECE 2.0. The driver may lock regions of code for the observer to review them. Thus, the observer may use any of the communication ways he is offered to communicate the code to the driver. Another way would employ the following mode. Once enabled, the observer watches all the driver operations made to the project, and thus successfully reviews it.

Roles may be arbitrarily swapped at any time. Therefore, the observer may request to be the driver through an intended mechanism in the user interface, and vice versa.

## 4.2 Real-time Concurrent Collaboration

The major capability of COLLECE 2.0 is the support to distributed real-time concurrent editing. Users may collaborate to solve a programming problem by editing the same source code at the same time. Thus, one of them may implement a specific method, while the other one is fixing an existing algorithm implementation.

For example, a simple programming implementation to multiply two matrices and output to the screen, could be carried out using a collaborative approach in which one member implements the actual multiplication method while the other implements the printing method. The Figure 4 shows an example of this workflow.



```

1 def mat_mul(a, b):
2     for i in range(len(a)):
3         # iterate through columns of b
4         for j in range(len(b[0])):
5             # iterate through rows of b
6             for k in range(len(b)):
7                 User 1
8     return result
9
10 def print_mat(a):
11     for User 2:
12

```

Figure 4. Two users editing the same source code at real-time

Both members could also use the shared whiteboard to sketch some ideas or to mock up the program interface.

A user may refactor the code too, distributing both methods in different files in the workspace. These files are also synchronized with the other collaborator since the whole workspace is shared.

Once the edition is complete and the program is finished, users may run it and see the result through the shared console.

## 5. CONCLUSION

Distributed real-time collaborative programming allows users to work on a programming project concurrently, improving the implementation quality of the solution. Also, it works as a learning environment where students may collaborate with each other to solve a programming problem.

In this work we have described the new COLLECE 2.0 solution for distributed real-time collaborative programming. The proposal builds on top of the Eclipse platforms, so users find it easy and familiar to work on, and thus the solution makes use of the underlying architecture and other well-known IDE features.

The usage of the Eclipse Communication Framework for implementing the network architecture and Apache Derby for database management, allows us to make an auto-contained solution where users do not

have to install any third-party software to use the environment in its entirety. In this line, other solutions require users to run external applications to handle chat communications, or a database server to handle data persistence, and therefore making it difficult for users to deploy their own instances for others to collaborate.

We have seen how COLLECE 2.0 provides users with different awareness mechanisms so they have a clear vision of what is happening at any given time. The capabilities for real-time shared workspaces allow users to collaborate over different programming projects. Also, the integration of emerging technologies like augmented reality and mobile capabilities, improves the learning experience by visualizing physical space aligned algorithms.

The solution presents some limitations, thus it is far from being perfect; high latency networks, different user editor settings, and specific Eclipse platform limitations, are some of them.

Ongoing work shall improve COLLECE 2.0 on what's been done, along with extending it with new features, and handling these limitations. These new features would allow users, for example, to share graphical user interface application executions so the same window is shared among users, or to propose shared polls with the aim of knowing other collaborators' answers to some questions.

## ACKNOWLEDGEMENT

This research has been partially funded by the Ministry of Economy, Industry and Competitiveness, and the European Regional Development Fund through the project TIN2015-66731-C2-2-R.

## REFERENCES

- Bravo, C., Duque, R. & Gallardo, J. 2013. A groupware system to support collaborative programming: Design and experiences. *Journal of Systems and Software*, 86, 1759-1771.
- De Carvalho, A. D. Cooperative live coding as an instructional model. *Computing Conference (CLEI)*, 2015 Latin American, 2015. IEEE, 1-7.
- Ellis, C. A. & Gibbs, S. J. Concurrency control in groupware systems. *Acm Sigmod Record*, 1989. ACM, 399-407.
- Fan, H. & Sun, C. Achieving integrated consistency maintenance and awareness in real-time collaborative programming environments: The CoEclipse approach. *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 23-25 May 2012 2012a. 94-101.
- Fan, H. & Sun, C. 2012b. Supporting semantic conflict prevention in real-time collaborative programming environments. *SIGAPP Appl. Comput. Rev.*, 12, 39-52.
- Gutwin, C. & Greenberg, S. 1995. Workspace awareness in real-time distributed groupware.
- McDonnell, M. 2014. Pair Programming. *Pro Vim*. Springer.
- Nichols, D. A., CURTIS, P., DIXON, M. & LAMPING, J. High-latency, low-bandwidth windowing in the Jupiter collaboration system. *Proceedings of the 8th annual ACM symposium on User interface and software technology*, 1995. ACM, 111-120.
- Salinger, S., Oezbek, C., Beecher, K. & Schenk, J. Saros: an eclipse plug-in for distributed party programming. *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, 2010. ACM, 48-55.
- Shen, H. & Sun, C. RECIPE: a prototype for Internet-based real-time collaborative programming. *Proceedings of the 2nd Annual International Workshop on Collaborative Editing Systems*. Philadelphia, Pennsylvania, USA, 2000. Citeseer.
- Thodi, M. & Fujita, S. Collaborative Development Environment in Peer-to-Peer Networks. *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, 2016. IEEE, 586-589.
- Wang, D., Mah, A. & Lassen, S. 2010. Google Wave Operational Transformation.