MALTE: Multi-Access Live Text Editor

Draft Report

Software Engineering Project, 1DL650

Adam Inersjö Jonas Norlinder Michael Rehn Joel Westerlund

December 10, 2019

Contents

1 Introduction **2**

2 Background **2**

2.1 Terminology 2

2.2 Consistency Model 2

2.3 Conflict-Free Replicated Data Type 2

3 Problem Formulation **3**

4 Related Work **3**

5 Requirements **3**

6 Technical Documentation **4**

6.1 API Specification 4

6.2 Concurrent Editing 4

6.3 Caret Synchronization 4

6.4 Shell Access 4

6.5 User Account Management 4

6.6 User Permissions 5

6.7 Development Instructions 5

7 Quality Control **5**

7.1 Tests 5

7.2 Continuous Integration and Deployment 5

7.3 Code Review 5

8 User Documentation **6**

9 Project Management **6**

10 Future Work **7**

1 Introduction

MALTE is a tool that runs in the browser and enables collaborative code editing. Moreover, each connected user have their own terminal on which they may compile and run the source code entered in the editor. Our tool runs in the cloud and all heavy-lifting (e.g. compilation) is done server side. This enables the users to utilize the product at any device capable of running a modern web browser. (More about MALTE itself here, features? target audience? more?)

The overall project goal was to allow users to collaboratively edit and compile code. To provide this the system was intended to have supporting functionality to make collaboration easy, this includes access to a terminal, commenting functions, chat functions, code high-lighting, etc.

2 Background

To enable users to edit code simultaneously some problems with synchronization has to be solved. When several users makes modifications to, e.g. a text file, the system has to ensure that all users has the same view of what the actual state of the text file should be, without inconsistencies. In order to provide a consistent state throughout the system, we make use of data structure called replicable growable array (RGA). RGA belongs to a class of data types called conflict-free data types (CRDT). In this section we firstly introduce some terminology commonly used, we then argue for our choice of consistency model. Finally we describe how we use CRDTs and RGA to enforce this consistency model.

2.1 Terminology

A *replica* in a distributed system represents a "copy" of some shared state. Each replica can be modified with *operations*, which is data that represents what modification is made, and how it should be performed.

The *operation* are then sent to other clients which apply them to their own replicas. This can be problematic in a distributed setting since some operations are not commutative, i.e., the order of operations influence the end state of the replica. For example, insertions into a string is not commutative since $a + b \neq b + a$. Algorithms behave differently and each algorithm guarantees their own properties of the final state. What properties can be guaranteed for an algorithm is said to be the algorithms *consistency model.*

2.2 Consistency Model

All distributed systems who deal with replicated data have to deal with the problem of consistency. There are many different consistency models, of which two are especially important. Eventual Consistency (EC) is a common consistency model which allow different replicas to diverge as long as they sometime in the future converge [1].

Strong Eventual Consistency (SEC) is a specific consistency model within the EC consistency model family. This model promises that all replicas that have observed the same operations, in any given order, will converge to the same state [1] without the need of any further synchronisation.

The promises of SEC is suitable for a system as MALTE as it ensures that all operations are considered for the final state. For the file tree, however, we are content to use the relaxed EC definition. The reasoning for this is that file system operations are less frequent and consistency can be solved with simpler means instead, e.g. checking if a file actually exists before trying to rename it. This is possible thanks to the use of a central server who has authority over the file system and that can resolve any potential race conditions.

2.3 Conflict-Free Replicated Data Type

A conflict-free replicated type (CRDT) is a data structure which guarantees SEC without the need for any synchronization [2]. The simplest examples of a CRDT would be a counter with the operations *add* and *subtract.* Both the *add* and *subtract* operations are commutative, thus, it does not require synchronization to converge, since e.g. $1 + 2 + 3 = 3 + 2 + 1$.

Operations in CRDT are often distinguished between upstream operations and downstream operations. The former being an operation generated by the local replica while the latter being an operation received from a remote replica. CRDTs for group editing usually have a very good downstream complexity whereas the upstream complexity suffers, resulting in less responsive group editing [3].

To improve the upstream time complexity of the data type RGA, one can use an auxiliary structure [3]. The auxiliary data structure has no need to be synchronized between replicas, yet improves the upstream operation complexity considerably. Implementing this auxiliary structure is out of scope for this project, but it is nonetheless worthwhile to know that improvements to the underlying structure are possible.

3 Problem Formulation

MALTE is with no doubt a distributed system with many points of replication. The text in each file is one point of replication. Another point of replication is the file system. Work has to be done to ensure either EC or SEC for both these points of replication.

As with all openly accessible system, it has to protected by some sort of authentication. To this end, authentication and user permissions is a problem which has to be solved. User data must also be stored in some form of persistent storage.

4 Related Work

There are already similar systems for collaborative, in-browser editing of code. These systems range from simple collaborative editors, such as Firepad, to full-fledged integrated development environments (IDEs) running on the cloud, such as Visual Studio Online [4, 5]. Visual Studio Online can be accessed through a browser and extended to allow for collaborative editing in a project. REPL.IT extends the concept of a project in Visual Studio Online by allowing for classrooms where students can learn and collaborate while teachers can assist the students and grade their solutions [6].

Other related software that experiences synchronisation issues are collaborative productivity tools such as Google Docs and Figma, all of which either uses Operational Transformations or Conflict-Free Replicated Data Structures to handle synchronisation issues [7–9]. As to our knowledge there are no open-source projects that provides the functionality we will implement.

5 Requirements

For the system to be considered complete, we considered the following requirements to be necessary

- Concurrent editing of files
- User and permission management
- Shell/terminal access
- File explorer with basic file system operations

Concurrent editing of files requires that two or more users should be able to edit the same file concurrently, with the guarantees of SEC, i.e. replicas are allowed to diverge as long as the converge when they have received all operations.

User and permission management requires the concept of a user. The users should be able to configure who has access to the project through the use of a UI.

Shell/terminal access requires that a user have access to a shell. This shell should be able to receive input from the user and display output from commands. The shell access should be personal and not shared between users.

At last, the file explorer requires a graphical view of the projects file structure. It should also be able to perform all basic file system operations, i.e. creation removal and renaming of a file or folder,

6 Technical Documentation

6.1 API Specification

// TODO: introducera vår API spec. Kanske bara kort info om socket.io's event system och möjligtvis länk till API spec på wiki

6.2 Concurrent Editing

Replicable Growable Array

To enable concurrent editing we have chosen to implement the Replicable Growable Array (RGA). The RGA is basically a linked list where each node contains either on or more characters, which together represents text. For example, Figure 1 show how the text "HELLO" could be stored in a linked list.

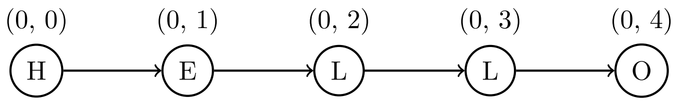


Figure 1: The basics of the RGA is a linked list with one character in each node.

Above every node in Figure 1 there is a tuple. This tuple is the node's identifier, or id. The first number in the tuple is the id of the user who inserted the node. The second number is the upstream vector clock value at the time the node was inserted. In our system we simplified the RGA structure to use Lamport timestamps instead of vector clock value. A node, or its id, is said to precede another node's id iff $clock_1 < clock_2$ or $clock_1 = clock_2 \wedge user\_id_1 < user\_id_2$ [3]. We use $id_1 \prec id_2$ to denote that id_1 precedes id_2 [3].

The RGA supports two operations: insertions and deletions. An insert operation contains two pieces of information, a reference node and the node that should be inserted. When applying the insertion, we will start by looking up the reference node. Then, we traverse the linked list until we find a node that precedes the new node; we will insert the new node to the left of the preceding node. By following this rule, concurrent operations will be ordered across all replicas in a deterministic matter, no matter the order of operations.

The insertion process is illustrated in Figure 2 where two concurrent insertions are made by user 1 and user 2. Both users want to concurrently insert a node to the right of the letter E. We have the following precedence order $(0, 2) \prec (1, 5) \prec (2, 5)$. By following the insertion rules specified above, we will end up with the final state shown in the last row of Figure 2.

Operation Mapping

// TODO: integration med monaco

6.3 Caret Synchronization

// TODO: Info om hur carets har synkroniserats

6.4 Shell Access

// TODO: Info om hur terminal/shell access implementerats (1 pty per user), powershell.exe för windows, bash för linux

6.5 User Account Management

// TODO: Info om GitHub integration, vart användar data kommer ifrån

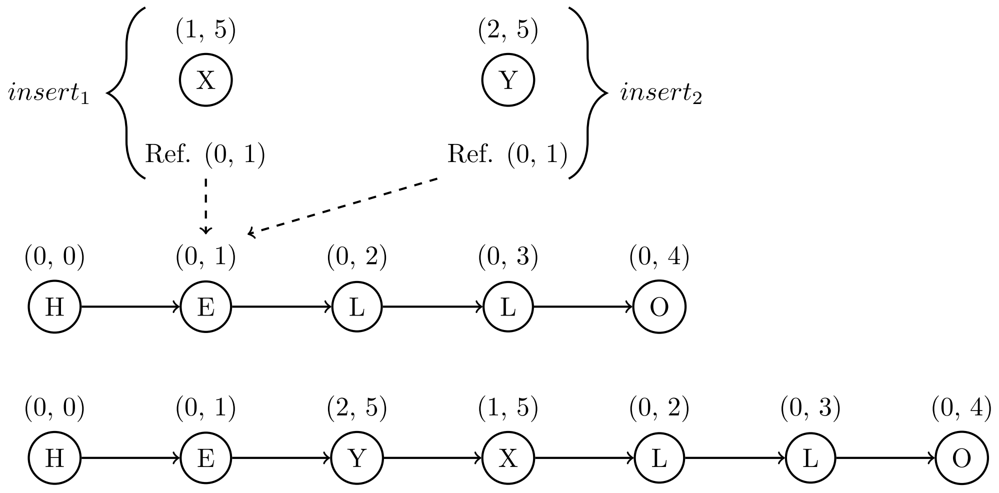


Figure 2: Two concurrent insertions, $insert_1$ and $insert_2$ modify the linked list shown in the first row in a deterministic manner to end up with the final state shown in the last row, no matter the interleaving of the insertions. The id specified next to "Ref." is the insertions reference id.

6.6 User Permissions

// TODO: Hur permissions systemet implementerats, dvs. första användare har alltid access, sen måste du bli "inbjuden" aka. få ditt GH användarnamn inskrivet i systemet

6.7 Development Instructions

// TODO: basically delar ur README.md, förklara projectstrukturen, hur köra tester, osv

7 Quality Control

We are utilizing modern techniques to achieve quality control: unit tests, regression testing with continuous integration and continuous deployment, linting tools and code review.

7.1 Tests

We have implemented tests for any part that were easily testable. Most importantly, the data structure which is the heart of this application, RGA, have a full test suite to verify proper implementation. Some parts that consisted of many or only side-effects we opted out from formalized testing. Those functions were instead verified by just using and trying to break it, this included mostly the frontend. See appendix X for a list of all functions that have tests.

7.2 Continuous Integration and Deployment

// TODO: CI/CD

7.3 Code Review

While developing we used the GitFlow workflow, a branching model for Git that is well-suited for collaboration in a team. This accommodated us well and was easily integrated into another requirement we had while checking in code: code reviews.

Each feature branch, bugfix branch or hotfix branch could never be checked into the main branch without first being reviewed by at least one other team member. There were two benefits for this. First, it established a baseline for code quality, since no one wanted to accept poorly written code into the main branch. Secondly, it increased the collective knowledge of the system within the group.

8 User Documentation

To use the system, there are two approaches. Either you are invited to a running instance of MALTE, or you can host an instance of MALTE yourself. If you are invited to a running instance of MALTE, all you have to do is visit the URL of the server, sign-in with GitHub, and start editing files.

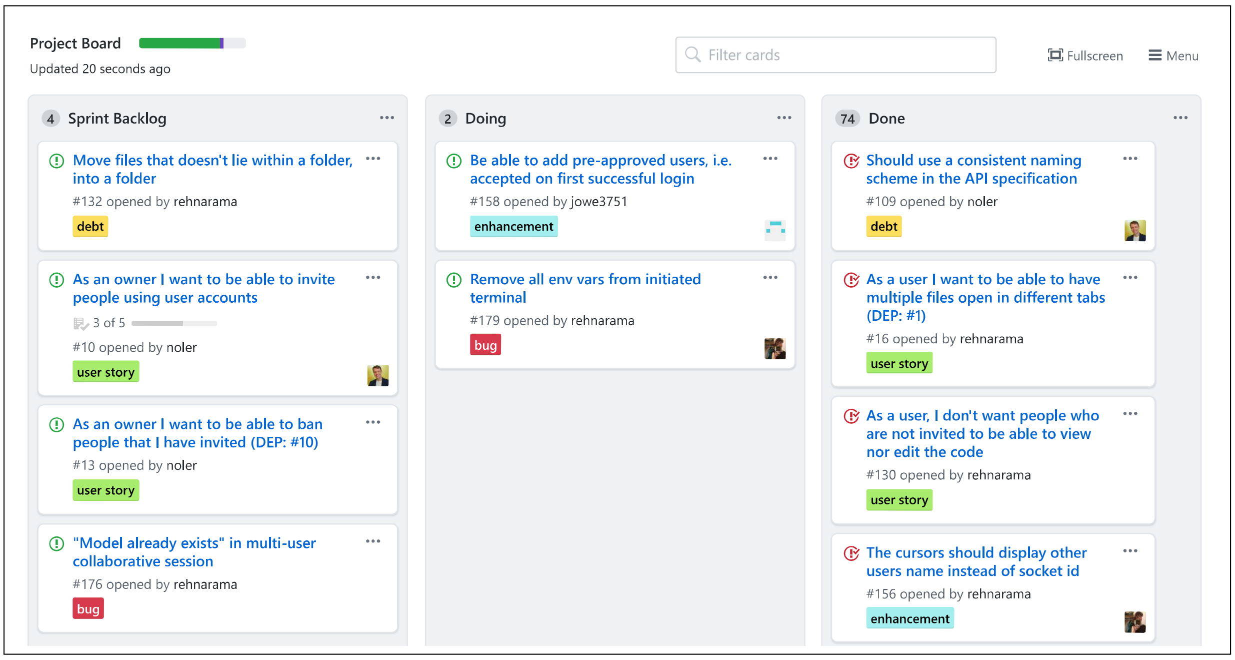
If you wish to host an instance of MALTE yourself, the preferred way is to use the provided docker image, malte-uu@hub.docker.com. Pay attention to the environment variables: these need to be configured for the system to work properly. An example command to run the app yourself would be

```
docker run −p 4000:4000 malte−uu:dev
```

and then visit the url *localhost:4000*. Alternatively, since the system is open source, you can access and build it from the source code directly, accessible at the GitHub repository rehnarama/MALTE.

9 Project Management

During the development of MALTE we used an agile project management methodology inspired by both SCRUM and Kanban. For example, we divided work into discrete sprints, just as in SCRUM, but visualised our work with a board similar to a Kanban board, seen in Figure 3. The tool we used for our work board was GitHub's projects functionality. This turned out to work well due to the integration with GitHub issues and pull requests, which made work easy to visualise and follow; if a change was made we could always track why the change was made.


Project Board
Updated 20 seconds ago
Filter cards
Fullscreen
Menu

4 Sprint Backlog
- Move files that doesn't lie within a folder, into a folder
  #132 opened by rehnarama
  debt
- As an owner I want to be able to invite people using user accounts
  3 of 5
  #10 opened by noler
  user story
- As an owner I want to be able to ban people that I have invited (DEP: #10)
  #13 opened by noler
  user story
- "Model already exists" in multi-user collaborative session
  #176 opened by rehnarama
  bug

2 Doing
- Be able to add pre-approved users, i.e. accepted on first successful login
  #158 opened by jowe3751
  enhancement
- Remove all env vars from initiated terminal
  #179 opened by rehnarama
  bug

74 Done
- Should use a consistent naming scheme in the API specification
  #109 opened by noler
  debt
- As a user I want to be able to have multiple files open in different tabs (DEP: #1)
  #16 opened by rehnarama
  user story
- As a user, I don't want people who are not invited to be able to view nor edit the code
  #130 opened by rehnarama
  user story
- The cursors should display other users name instead of socket id
  #156 opened by rehnarama
  enhancement

Figure 3: Example of the Kanban board used for the project for each sprint

One member of the team was working remote during all of the project which made communication key for the group work to function well. All group communication was handled via Slack, where we also had digital stand-up meetings (in true SCRUM spirit!) to share what we had worked on, what we will work on, and if we had any blockers.

After each sprint

// TODO:

10 Future Work

// TODO: software as service with meta server // TODO: CRDT improvements // TODO: Research scalability // TODO: More?

References

[1] J. Bauwens and E. Gonzalez Boix, "Memory efficient crdts in dynamic environments," in *Proceedings of the 11th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages*. ACM, 2019, pp. 48–57.

[2] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Symposium on Self-Stabilizing Systems*. Springer, 2011, pp. 386–400.

[3] L. Briot, P. Urso, and M. Shapiro, "High responsiveness for group editing crdts," in *Proceedings of the 19th International Conference on Supporting Group Work*. ACM, 2016, pp. 51–60.

[4] "An open source collaborative code and text editor." [Online]. Available: https://firepad.io/

[5] "Developing with Visual Studio Online," Apr 2016. [Online]. Available: https://code.visualstudio.com/docs/remote/vsonline

[6] Repl.it, "The World's Leading Online Coding Platform." [Online]. Available: https://repl.it/

[7] "Operational Transformation," Oct 2019. [Online]. Available: https://en.wikipedia.org/wiki/Operational_transformation

[8] "Conflict-Free Replicated Data Type," Nov 2019. [Online]. Available: https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type

[9] "How Figma's Multiplayer Technology Works." [Online]. Available: https://www.figma.com/blog/how-figmas-multiplayer-technology-works/