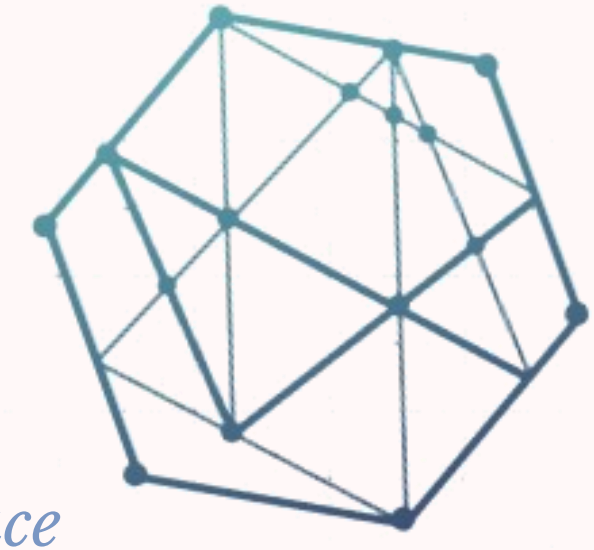
Programmazione di Sistema

Progetto del corso, a.a 2018/2019:
Sistema di editing testuale cooperativo

Introducing...

Symposium



The collaboration place

"Symposium is a free, real-time, collaborative text editor written in C++ language and using Qt framework.

Like in the ancient Greece, Symposium allows you to collaborate and exchange ideas with other people in an environment that suits you".

The team at a glance

Ksenia Del Conte Akimova

Software

File system

User identity

User Experience & User Interface

Riccardo Zaccone

Data analytics

Team Leader

Software Architecture

Server-side programming

Serialization&Persistance

Testing

Cristian Gianetto

Computer networks

Client-side programming

Network

Martina Bellissimo

Automatic

Directory Interface

TextEdit Implementation

TextEdit Interface

All started with a lot of talk...

"Much of the essence of building a program is, in fact, the debugging of the specification" ~ *Fred Brooks*

01

- Client-server desktop application in C++, platform independent;
- Users work on the same documents at the same time;
- Users are registered and see each other actions and positions;

02

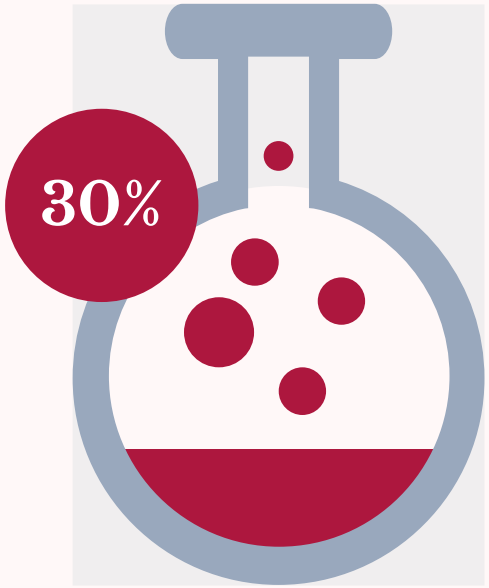
- Users own documents, they must be easy to access, once created;
- Users share documents, some form of access control is desiderable;

03

- Details of users, sharing policies, document properties and organization should be editable anytime;

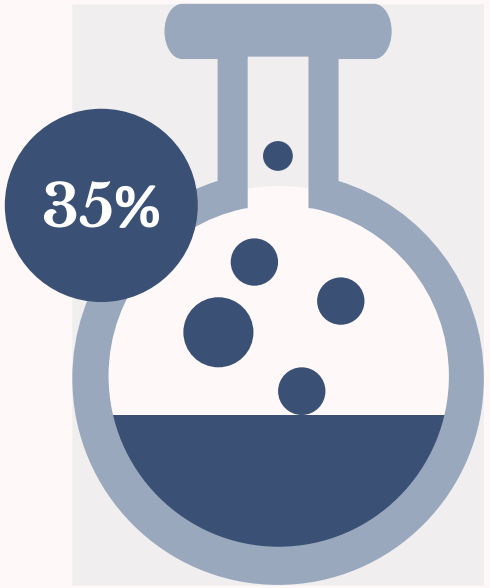
Main concern during development

Time and effort distribution across activities



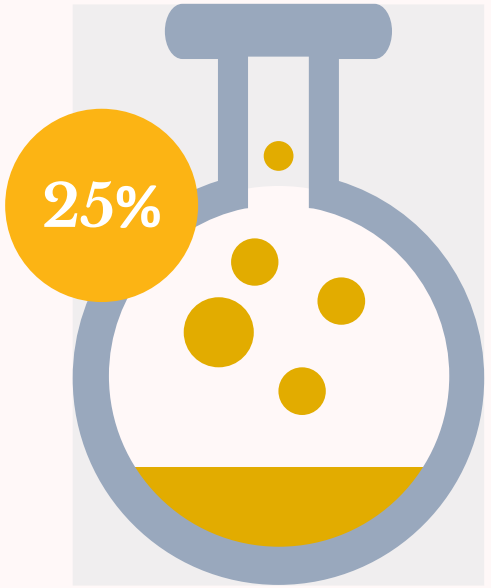
Structure

Build and mantain a good relationships between components



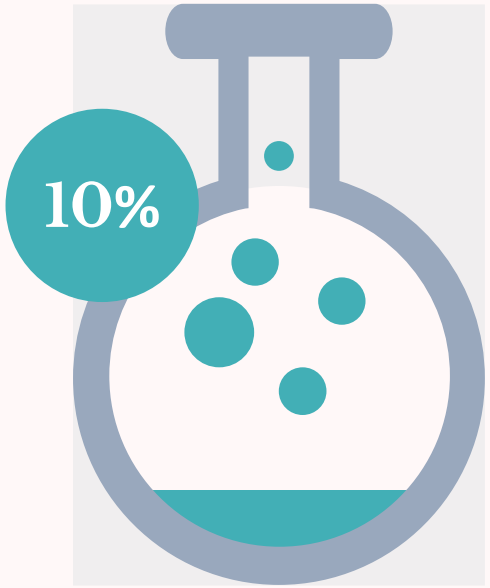
Testing

Building and refining components that really work on use cases



UX

Reasoning about ease of use from the user's perspective



Documentation

Make it all comprehensible to other developers (and yourself)

In short words:

Build to be reused and extended, like in a real case and not just for the exam.

Build a *programming product*[1] and not just a program

1. Taken from «The Mythical Man Month» (Fred Brooks, '75), p. 6

Standards and tools for documentation

"Documentation is a love letter that you write to your future self" ~ *Damian Conway*

Code documentation

Write documentation in the code (header files) and extract it in browsable document

Components&Relationships

UML diagrams of some of the key classes

Processes

BPMN diagrams describing in detail the execution flow of a client-server interaction

Symposium's structure

The goal of good software design is to minimize the human resources required to build and mantain the required system.

~ Robert C. Martin

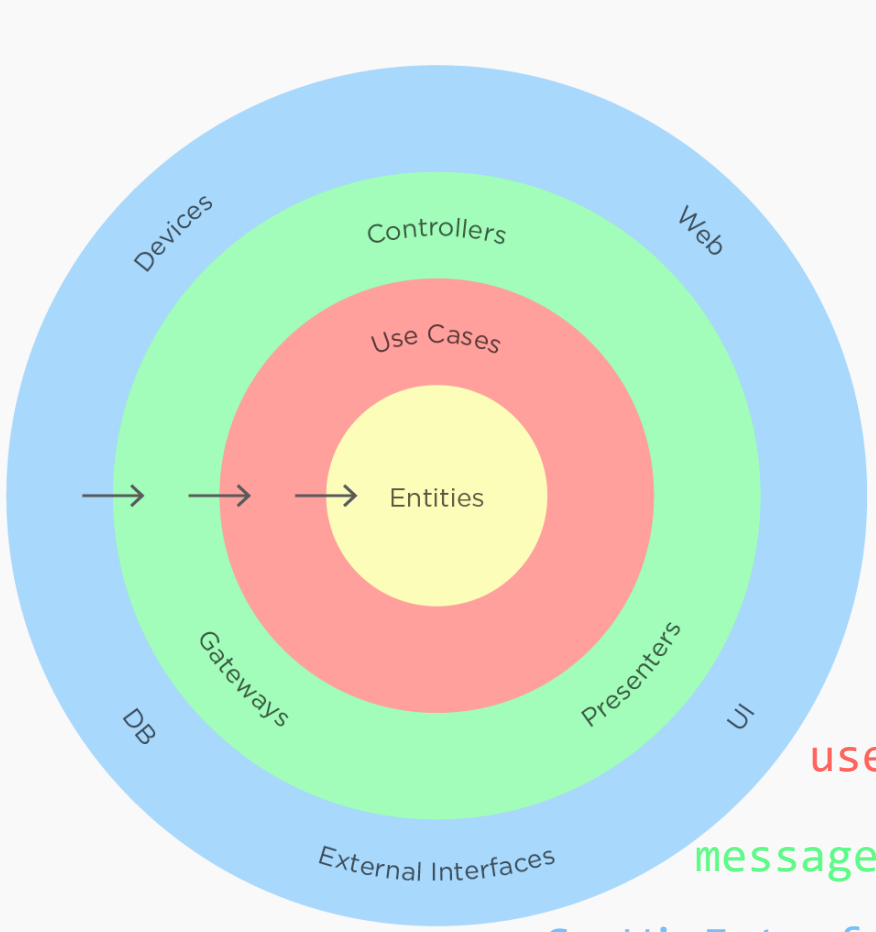
01 Software Architecture

02 Serialization& Persistance

03 Testing

Clean Architecture model – 1/2

"Software architecture is the art of drawing lines that I call boundaries" ~ *Robert C. Martin*



Principles followed:

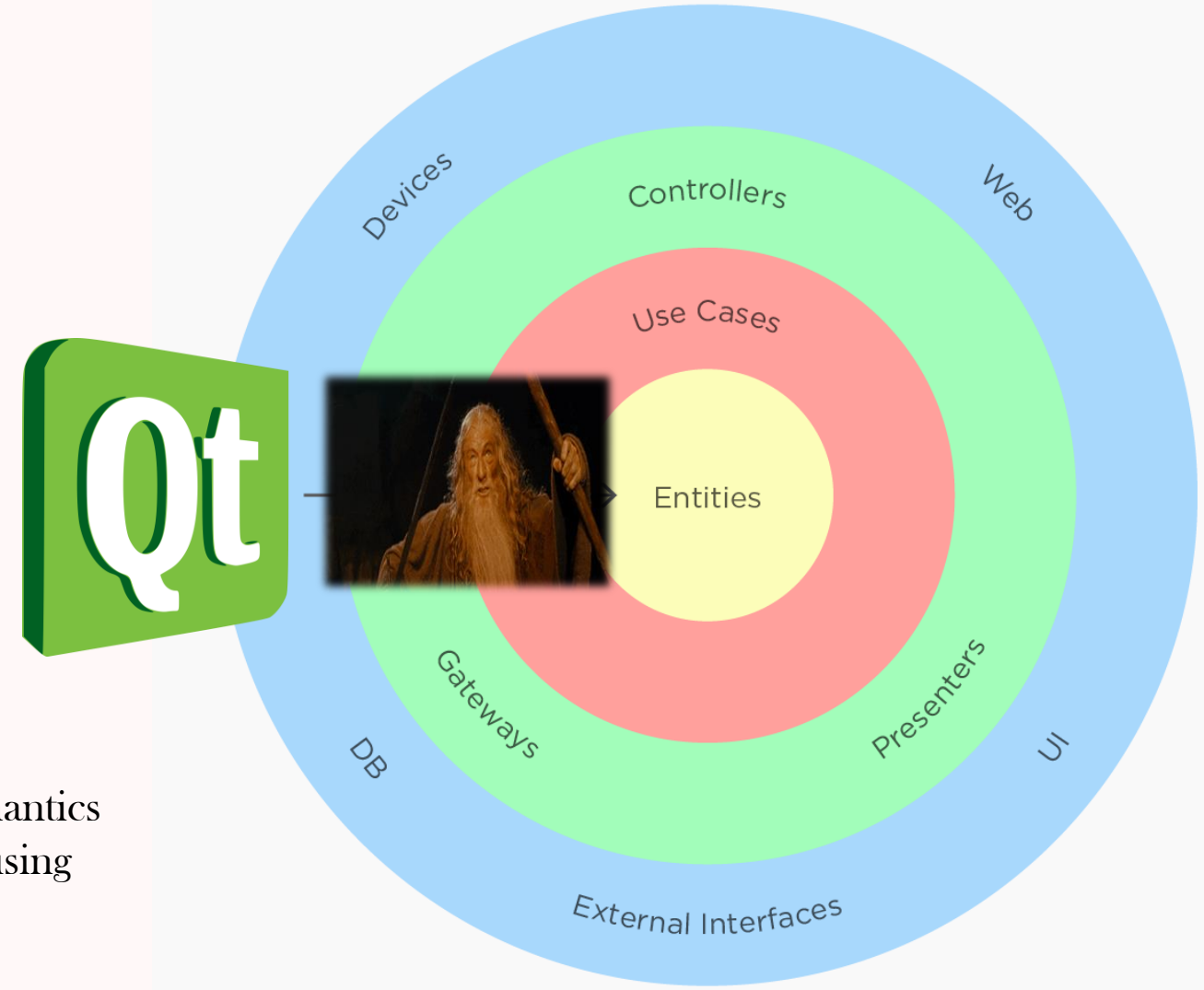
- *Common Closure Principle*: gather into components those classes that change for the same reasons and at same times;
- *Stable dependency Principle*: depend in the direction of stability;
- *Stable Abstraction Principle*: not applied at every boundary crossing, only when there is an evident advantage (one-dimensional boundaries);

```
privilege, uri, symbol, Color,
counter, AccessStrategy, resourceType

user, document, filesystem, SymClient, SymServer

message, clientdispatcher, serverdispatcher

SymWinInterface, SymWinManager, … other classes for windows
```

Clean Architecture model – 2/2

On one principle we stayed stiff:

- *Don't marry the framework:*
  - Keep the framework in the outer cycle so that does not introduce a dependency upon entities or use cases



Side benefits:

- Consistency with the Standard Library: often Qt optimization mechanisms do not respect C++ semantics (i.e implicit sharing), so special care is needed in using those components (i.e iterator problem);
- Tests can be run without linking to Qt;

Clean Architecture model – examples

How to extract the interaction flows between client and server? Messages!

A general mechanism

Actions by users on clients generate request to server and trigger actions on other clients: there are plenty of these;

Trasparent for clients and server

For remote actions, server and clients should not care about *which* specific action is required;

Easy to be extended

The difficulty in making a change in software should be proportional only to the scope of the change, and not the shape of change.

Clean Architecture model – examples

Solutions:

Decouple the action from the request

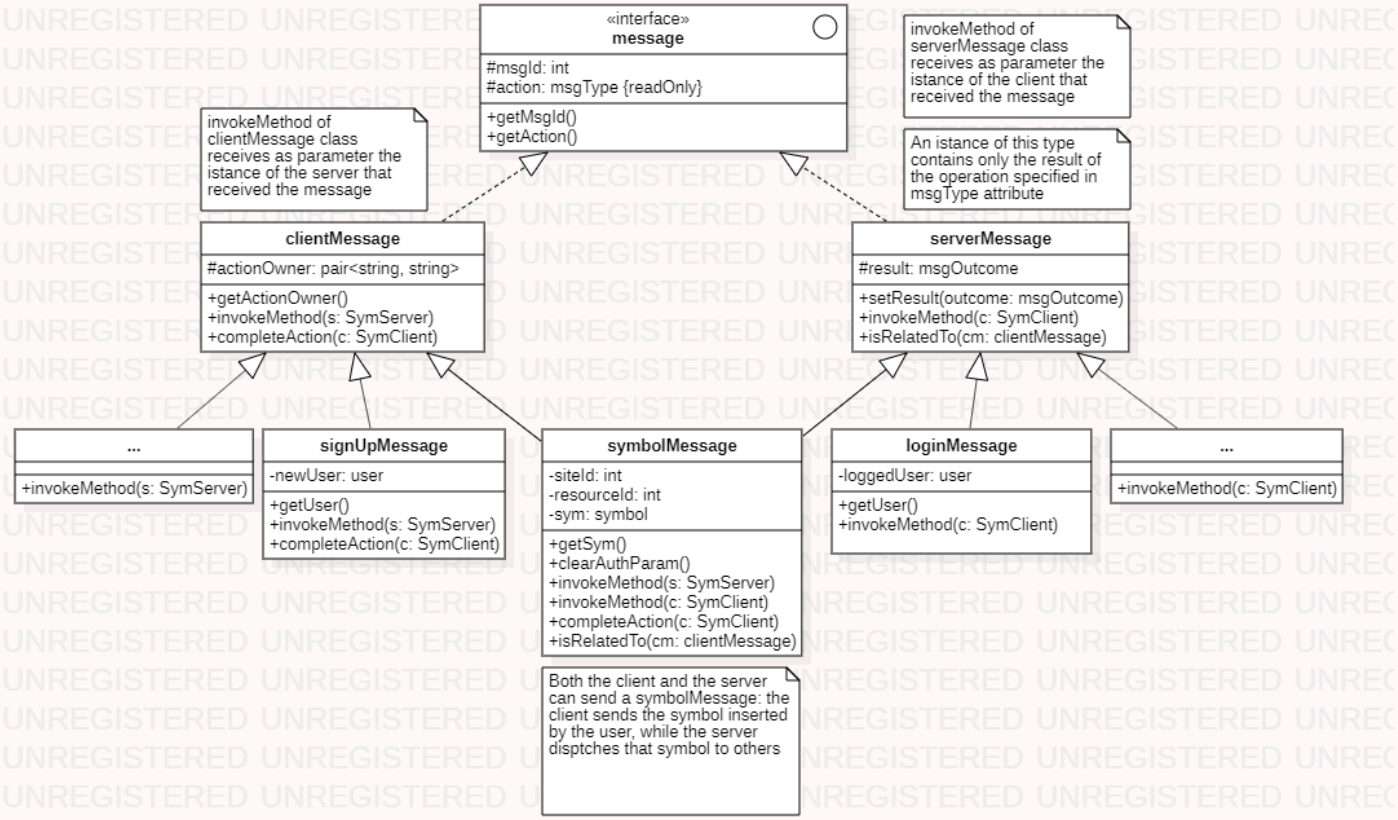
Introduce another class that deals with decoding which action has been requested;

Automatic Runtime resolution

No switch/if-then-else, error prone and a lot of clustered code. Polymorphism solves this automatically;

Use inheritance

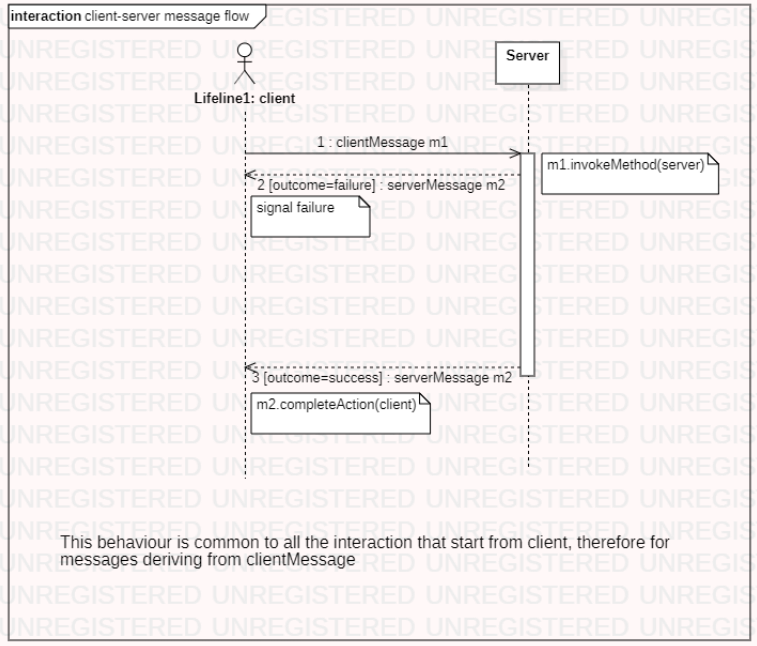
By the Open-Closed principle, a software artifact should be *open for extension* but *closed for modifications.*



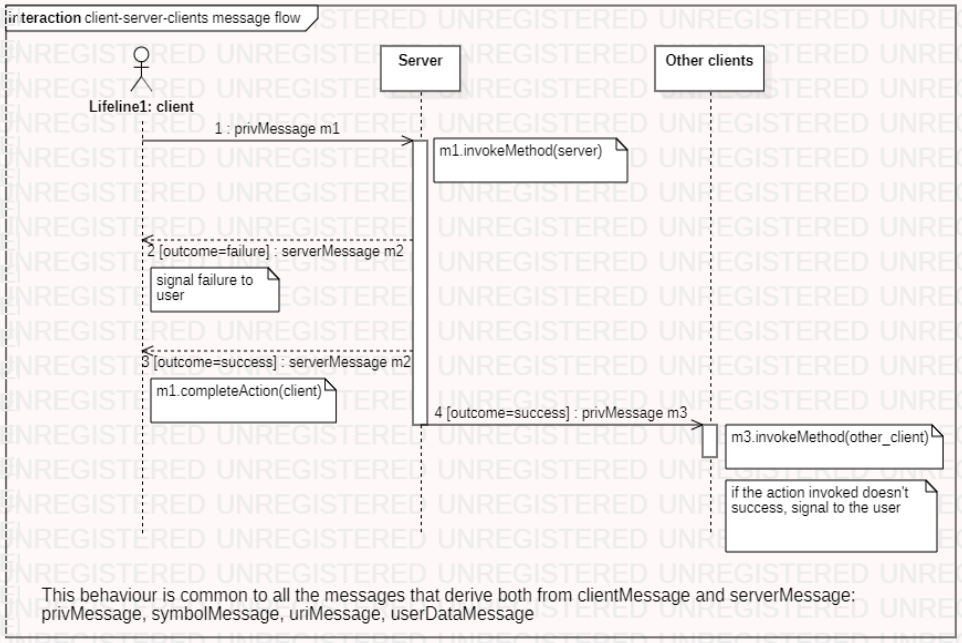
Clean Architecture model – examples

Models of interaction

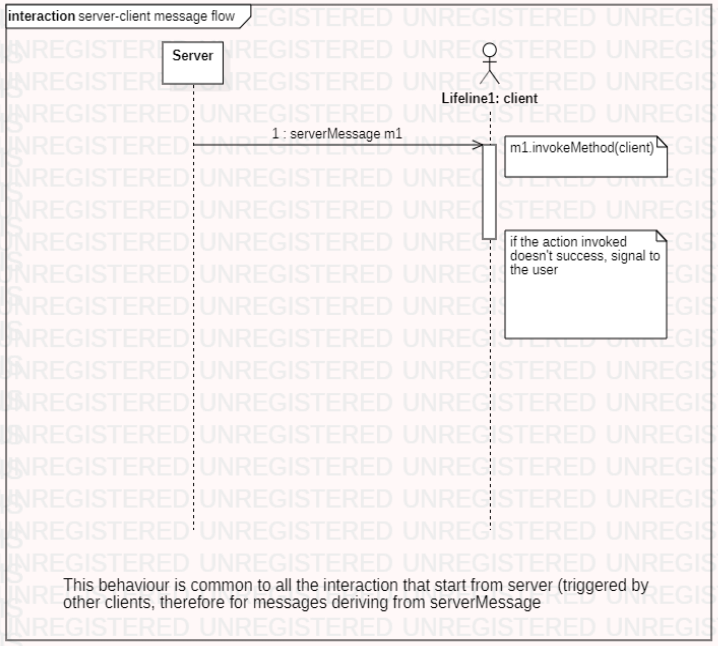
Client-server



Client-server-client

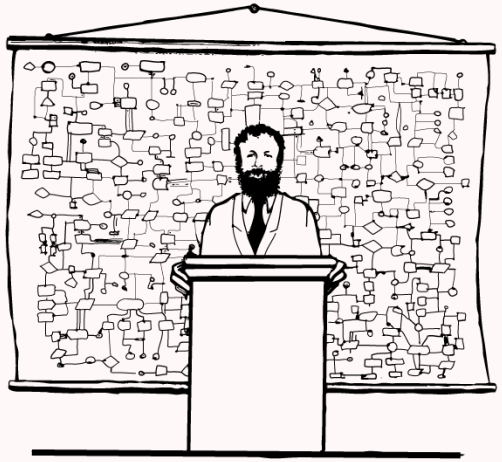


Server-client



From objects to bits and viceversa

A round trip through a message's life



"Now that you have an overview of the system, we're ready for a little more detail"

Image source: Wikipedia

01 Software Architecture

02 Serialization& Persistance

03 Testing

What's the matter in serializing an object?

In the first place: what are the constraints?

General mechanism

The action of serializing or deserializing an object should be transparent to the caller and uniform in its implemenation;

Support hierarchies

The main reason to serialize is to exchange messages, so should be possible to recostruct polymorphic types and pointers;

Little code to write

Avoid to write all the necessary logic from scratch, make it easy to serialize any new component;

Robustness & performance

The approach should be reliable and as fast as possible;

The winner is: `boost::serialization` 1/3

But… what were the alternatives?

Google Protocol Buffers

Bad points:

- Involves an external file where specify how to ser./deser. members;
- Protocol buffer classes are basically dumb data holders (like structs in C); they don't make good first class citizens in an object model.

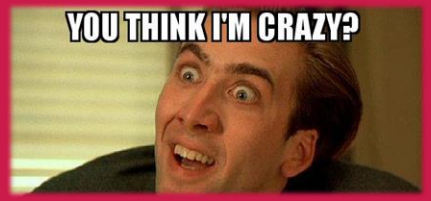
boost::serialization

Bad points:

- Seems easy to use, but just a small detail can cost a lot of debugging;

By hand

Seriously?



The winner is: `boost::serialization` 2/3

A deeper look at what convinced us to use it

From boost::serialization documentation:

- Code portability - depend only on ANSI C++ facilities.
- Code economy - exploit features of C++ such as RTTI, templates, and multiple inheritance, etc. where appropriate to make code shorter and simpler to use;
- Deep pointer save and restore. That is, save and restore of pointers saves and restores the data pointed to;
- Serialization of STL containers and other commonly used templates;
- Data Portability - Streams of bytes created on one platform should be readable on any other;

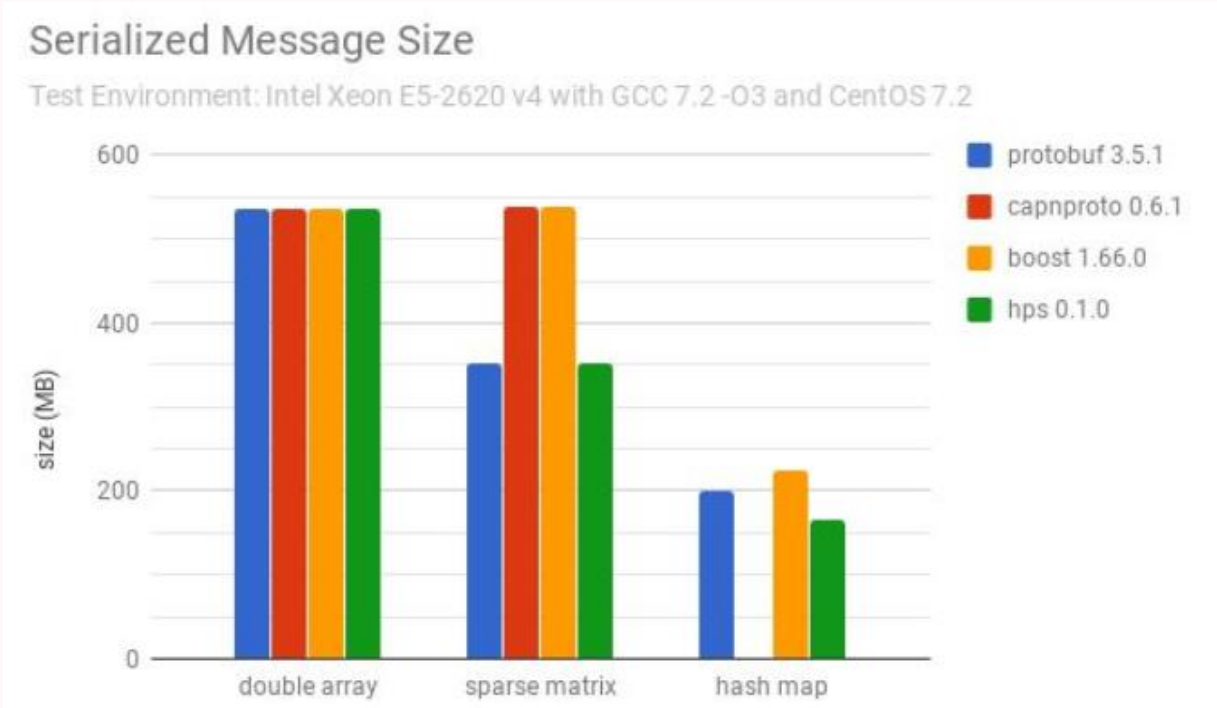
In addition:

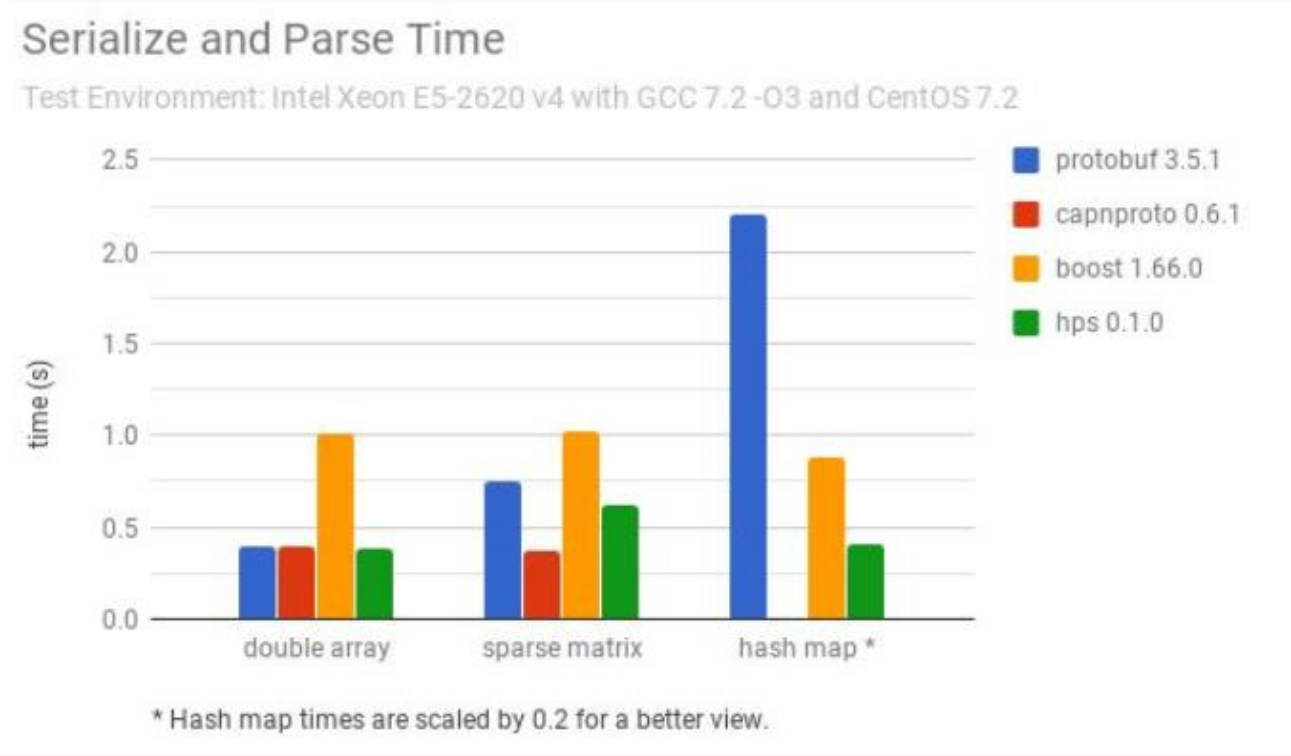
- Using boost components is nearly to use the Standard Library: in the past some boost components inspired the implementation of the same concepts in the STL (e.g. std::map, smart pointers like std::shared_ptr, mutexes, ecc.);

The winner is: `boost::serialization` 3/3

And... what about performace?

DISCLAIMER: we did not run performance tests, neither it was a main concern. That said, on the web is possible to find some statistics:





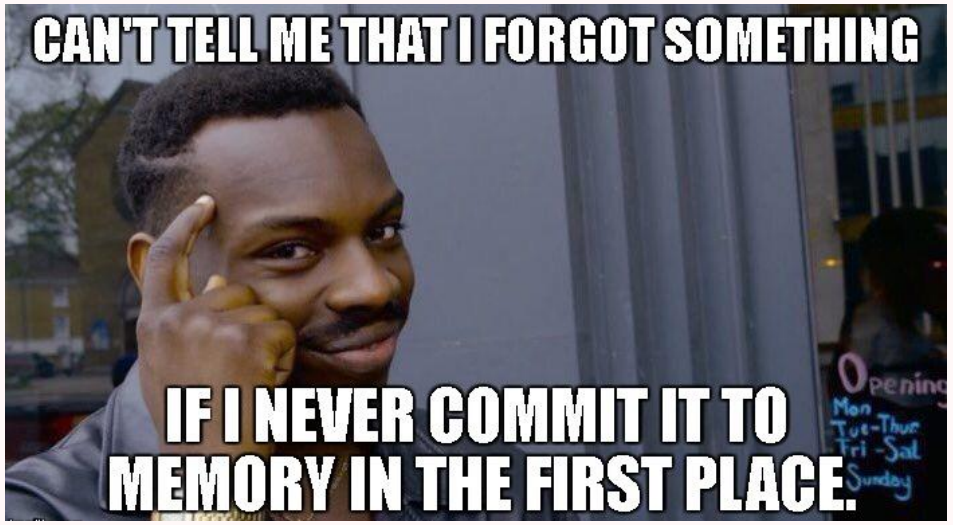
Source: https://arxiv.org/ftp/arxiv/papers/1811/1811.04556.pdf

Ok but… What about persistance? 1/2

Concerns and rationale

Why not a simple relational database?

- Lack of unique representation:
  - Our objects' nature doesn't fit in a standard relational database;
  - Representation may change over time;
- Convenient access to data:
  - Guarantees: we may like ACID properties but…
  - Performance: are we ready to accept the burden?
    - Everything is a transaction: another software layer, accessed by IPC;
    - Actions last longer: each transaction is a potential synchronous write on disk;

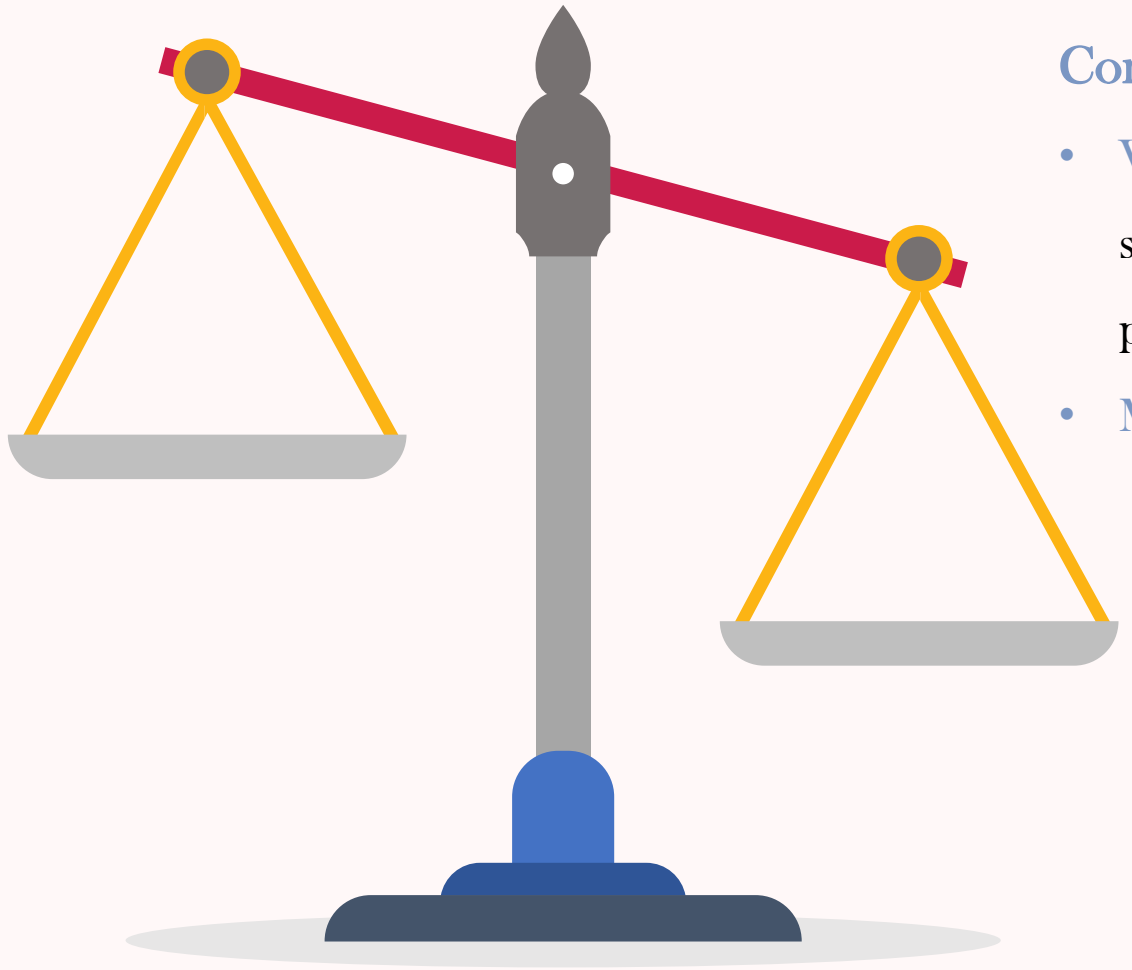


Ok but… What about persistance? 2/2

Solution adopted: re-use serialization

Pros:

- Performance: actions last the minimum require amount of time;
- Development: no ad-hoc code required, no additional dependency;



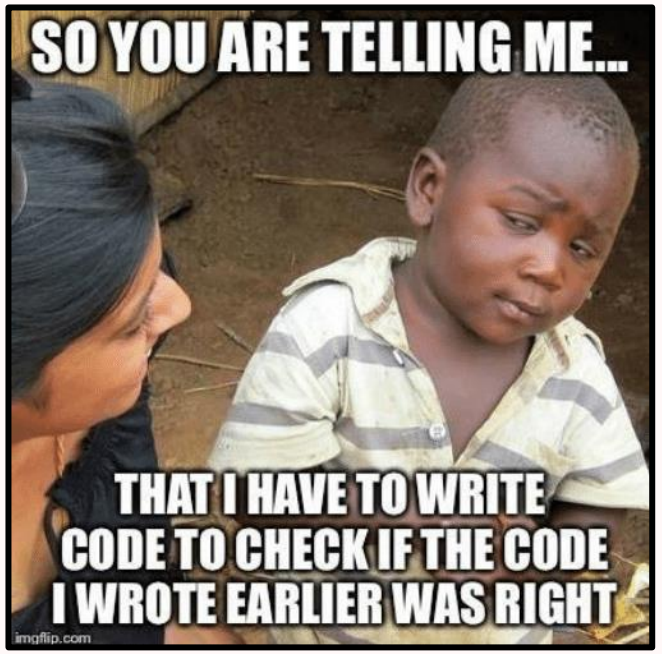
Cons:

- Weaker guarantees: not writing synchronously on disk comes at a price;
- More responsabilities:
  - Error handling: in memory data is a resource to be preserved even under error conditions;
  - Process death: must ensure process does not terminate abnormally;

Okay, Houston, we've had a problem here

Program testing can be used to show the presence of bugs, but never to show their absence!

~ Edsger W. Dijkstra



01 Software Architecture

02 Serialization& Persistance

03 Testing

Why testing?

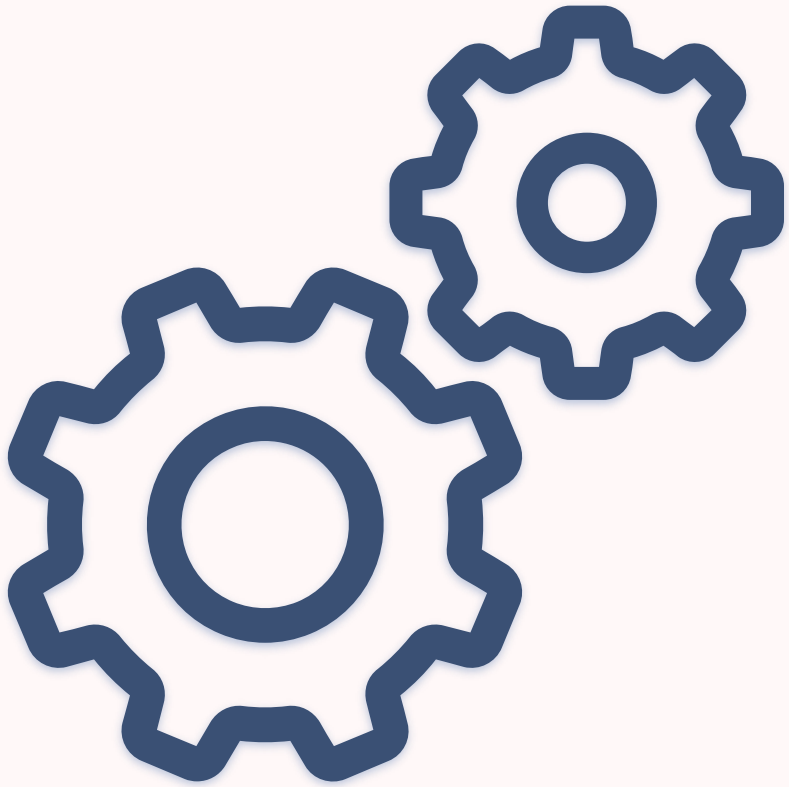
"The principle objective of software testing is to give confidence in the software" ~ *Unknown*

Isolate the bug

In a complex program is crucial to know what to look at when a bug occur; if nothing can be trusted, every instruction can be the problem;

Early feedback

You may want to see part of the feature running before the whole software is completed;



Challenge the design

Difficulties in testing can suggest that there must be a cleaner way to go;

Automate use cases

In a complex program can be difficult to try every action that can be asked to the system. Moreover is hard to keep track of these trials;

Testing and tools

Main concerns:

- Breaking up the dependencies: all classes but entities are required to be tested breaking up the relationship with lower level components;
  - Google Mocks uses inheritance to make it possible for a mock class to be used in place of the real class;
    - This requires function methods to be `virtual`;
  - Testing the internal (private) details: can be difficult if an internal member should be a mocked class and it's not received from outside:
    - Make members accessible from outside?
    - Modify the interface to allow intrusion by tests?
    - Declare test classes as `friend` of tested class?

googletest
Google C++ Testing Framewofk

Client-side

Client-side Programming

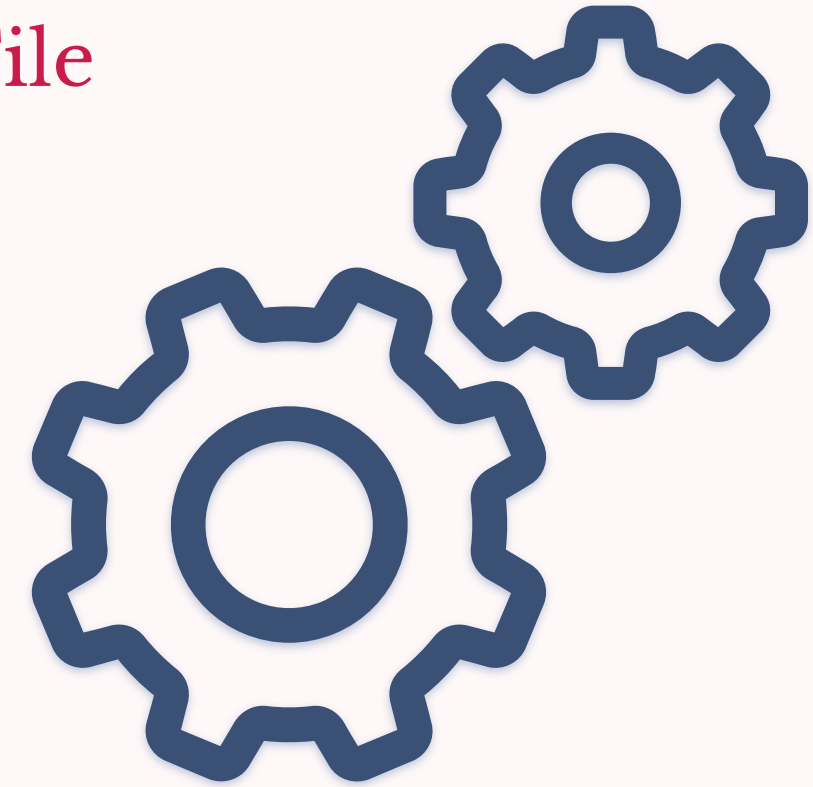
Network

Symposium Client 1/2

A SymClient class contains all the information used by the client at that time

activeDoc & activeFile

Two separated forward list: first one contains all the documents opened by the user, second one contains opened files



userColors

Map that contains mapping between UID, documentID and user, color

usersOnDocuments

Map that contains all users online on open documents

unanswered

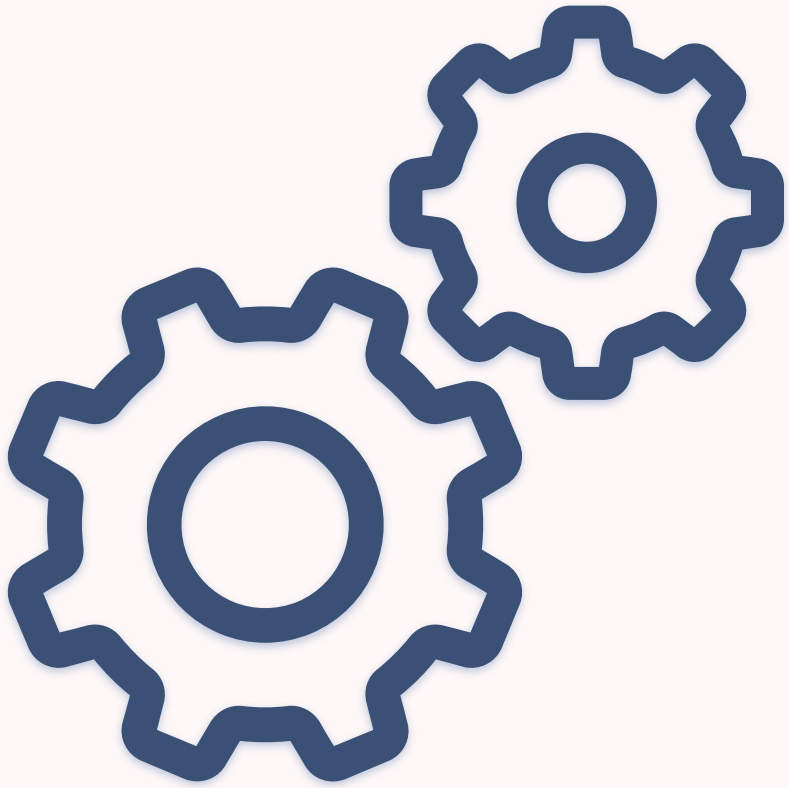
Forward list of messages sent to the server that have not yet been answered

Symposium Client 2/2

A SymClient class contains all the information used by the client at that time

loggedUser

Variable that contains all the informations about the user



dispatcher

Reference to clientdispatcher, class that implement interaction between network

Symposium Network

Client-side Programming

Network

The main goals

The main objectives that the Symposium network level must have:

Carry messages

To be able to reliably transport messages from clients to the server

Yes to details no complexity

To have complete control of the network while minimizing complexity

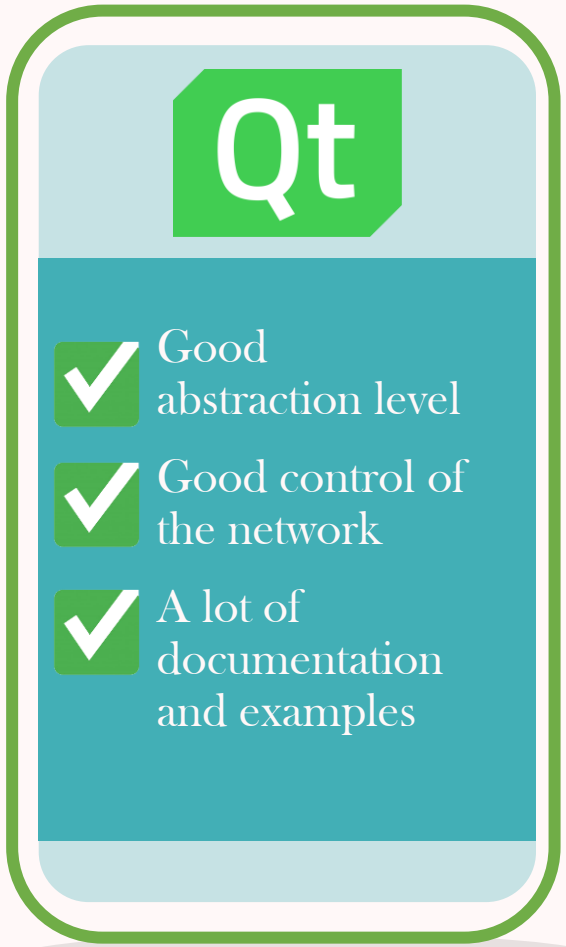
Completely transparent

The transport mechanism on the network must be completely transparent to the end-systems

The alternatives available

What alternatives have been considered:

- Good abstraction level
- Good control of the network
- Poor clear documentation

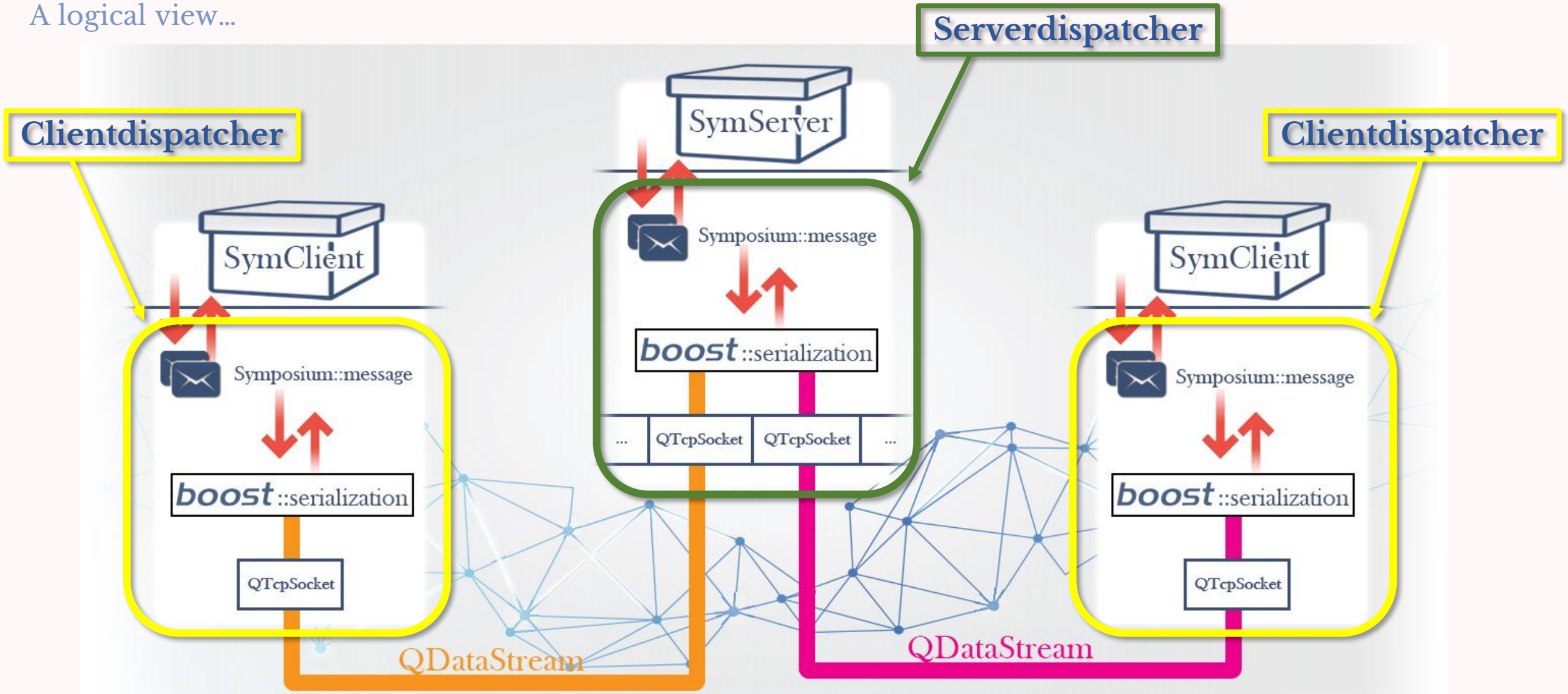


- Good abstraction level
- Good control of the network
- A lot of documentation and examples

Quite complex and little abstraction

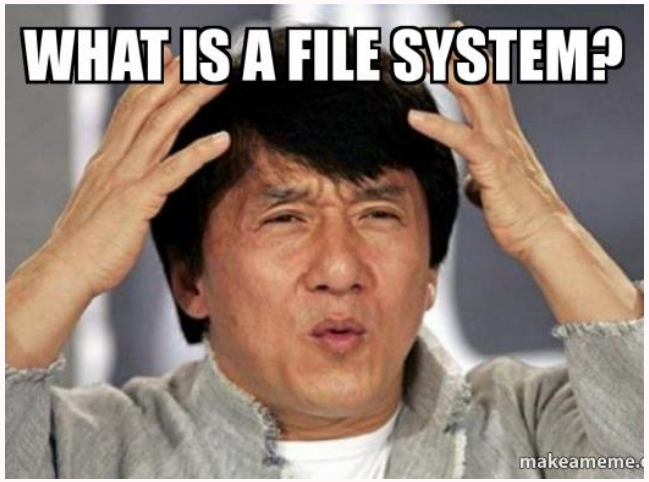
Structure of our network layer

A logical view...



Symposium's file system

How data are stored and retrieved



01 File system

02 Users online

03 UX & UI

File system

What we needed...

How we can organize users' data?

How we can keep it simple for users?

What is certain: the user does not have to think about it!

How we can keep it simple for us?

File system

...and what we did

01 The way to keep the user from thinking is to use a model he already knows.

02 Organize file system with directory and file. So users knows what these elements mean and visualize their data in friendly way.

03 Organize file system with directory and file. So we can separate in different levels the main operations of users and keep everything under control.

File system

What about sharing?

Object

The most familiar way for the user to visualize that something is pointed to something else – symlink

Features

What can we do to enrich the experience? We can allow users to share their files with different privileges like owner, writer and reader.

File system

More about privilege: at logical level we have four different privilege: owner, writer, reader and none.

Owner

User can do: sharing, change privilege of other users, modify document

Note: the object file can be remove only by this user and only if there is no other owners!

Writer

User can do: modify document

If user remove symlink with this privilege nothing hapens; if user have object file with this privilege he cannot remove this file, the operation can be done only by owner user!

Reader

User can do: read document

If user remove symlink with this privilege nothing hapens; if user have object file with this privilege he cannot remove this file, the operation can be done only by owner user!

None

User cannot do anything!

If user try to open symlink with this privilege it will be deleted automaticaly; if user have object file with this privilege he cannot remove this file, the operation can be done only by owner user!

File system

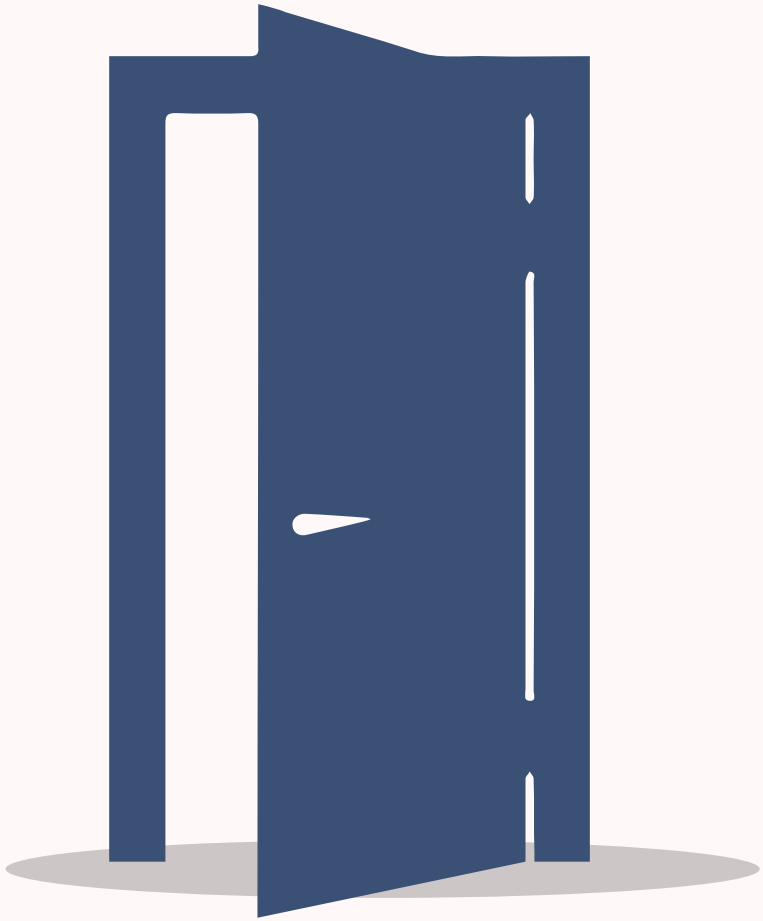
Possible improvements

Sharing

Why limit yourself only to documents? Structure that can be used with minimal effort even for folders!

Objects

Why limit yourself only to directory, file and symlink? Structure that can be used with minimal effort even for other objects like images and similar!



Identity of users

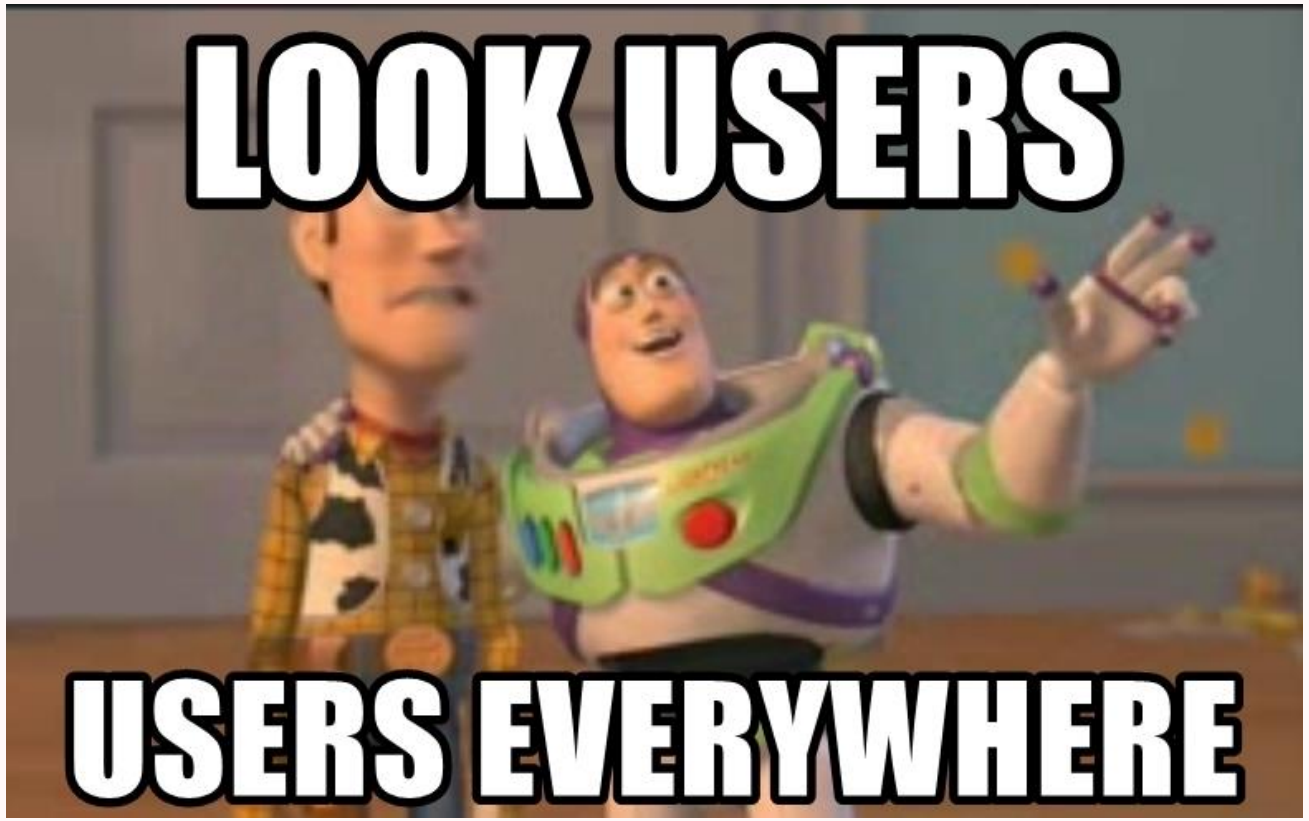
01 File system

02 Users online

03 UX & UI

Identity

- Diversify users through their username;
- Protect users' password server-side with sha256;
- Mark the position of users in the document with their name;
- Diversify multiple users visually in an open document? Use different colors for cursors also!



User Experience & User Interface

Deep down inside every software developer, there's a budding graphic designer waiting to get out. And if you let that happen, you're in trouble. Or at least your users will be, anyway...

~ Jeff Atwood

01 File system

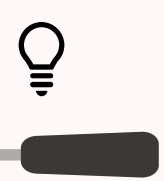
02 Users online

03 UX & UI

User Experience & User Interface

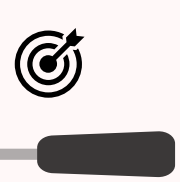
KISS technique – Keep It Simple, Stupid. Most system work best if they are kept simple

How to organize contents on screens? Where user expect them! So where other developers put similar contents



Position of contents

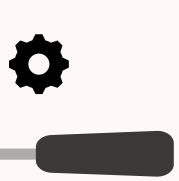
Giving user a sense of being in control, knowing what to do and how to do it



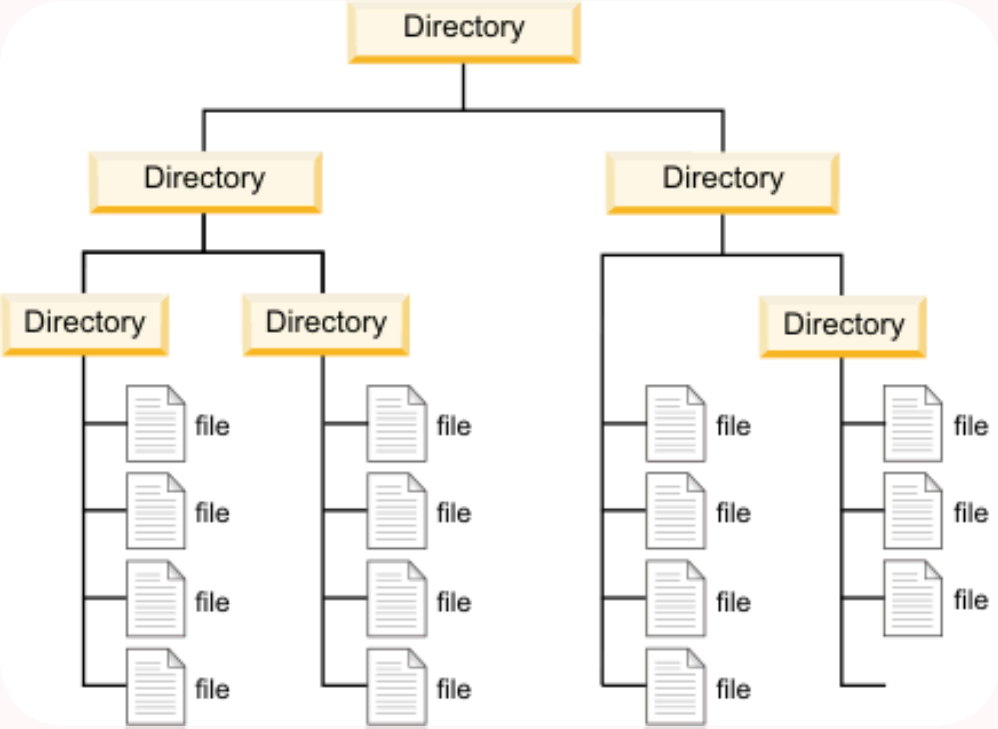
Keep user in control

Graphic, colors, font...

Use not more than 3 colors, same font and graphics line in all screens



Consistency



...What is a directory?..

an organizational unit used to organize folders and files

- ❑ Folders
- ❑ Files
- ❑ Symlink

Actions that can be performed:

- Go to Home
- Add Folder
- Add Link
- Add Document
- Open Source
- Delete Source
- Rename Source

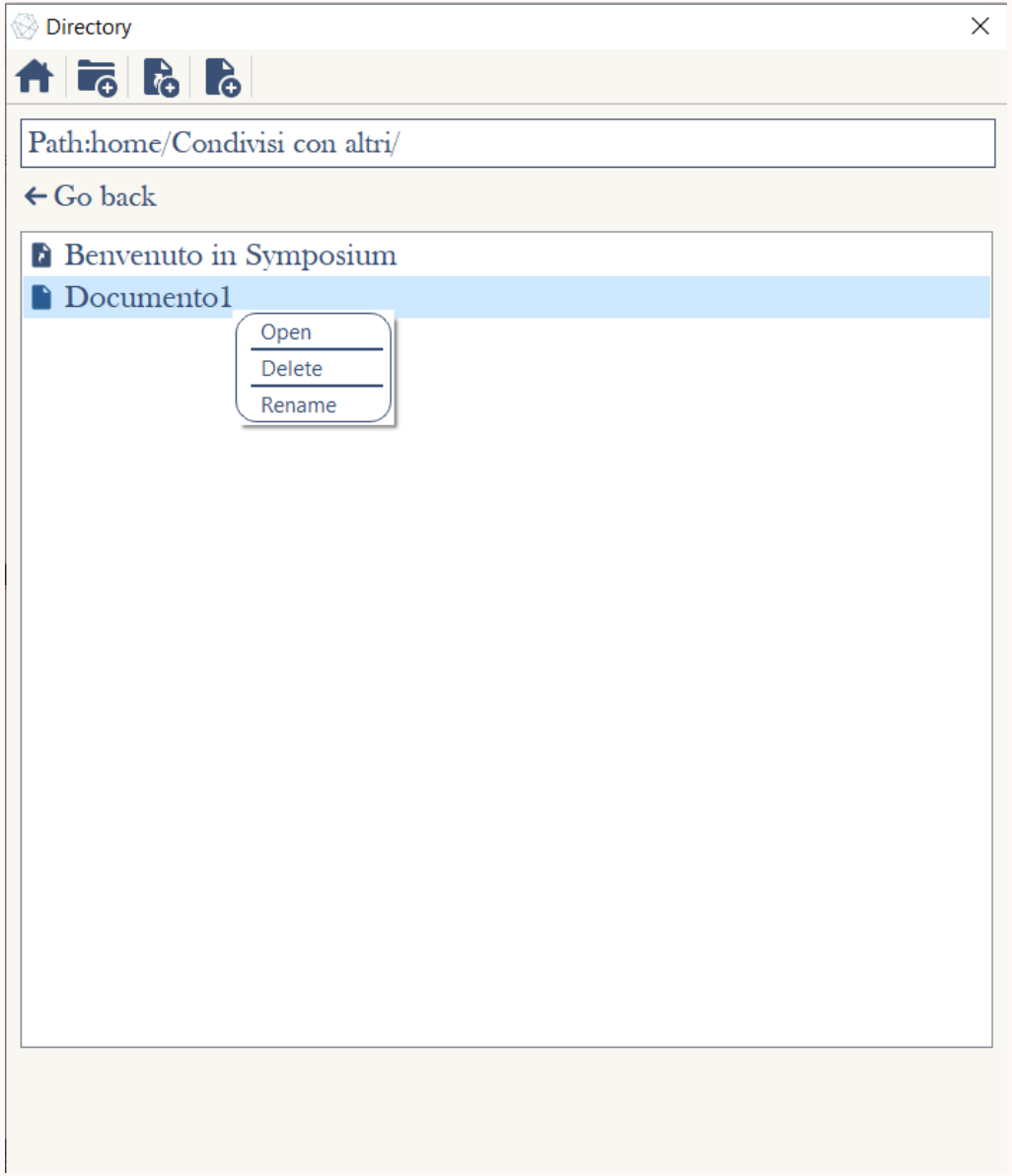


Image taken from Symposium

TEXT EDIT

Collaborative text editor implementation

The document class handles both the **local** and **remote** insertion and elimination of characters

The algorithm used to build these function was inspired by the CRDT strategy of Conclave

A crucial aspect is the positioning of the characters. It is important to preserve the order of characters with a text file.

..Operations..

The algorithm contains four basic operations:

Local Insert:

User inserts a character into his local text editor

Local Remove:

User deletes a character from his local text editor

Remote Insert

User inserts a character and it appears to all the users that have the same document

Remote Remove:

User deletes a character from his document and it will be removed from all the symlink linked to it

TEXT EDIT INTERFACE

Most relevant features

Character Validation

The insertion or elimination actions are executed when the user performs them, but they can be confirmed or not by the server.

- **Insertion Validation:**

  At first, when a user inserts a character, it will appear with a lighter color. This means that it has not been verified. As soon as the server confirms the correctness of the operation, the character will appear of the color chosen by the user.

If the insertion operation is not confirmed by the server, the characters will continue to appear with a lighter color.

- **Remove Validation:**

  The delete operations are also confirmed by the server.

  If this does not happen, the characters will not be deleted from the text block.

Validation of characters holds both for local operations and for remote ones.

Character Characteristics
- RICH TEXT-

The characteristics of a character are the ones that allow to reproduce it in a correct form, chosen by the user that inserts it, into the remote sources.

They describe la possibility to:

Add fonts with different point sizes and style, such as **Bold Text**, *Italic Text*, Underlined Text;

Add a variety of Foreground colors and all fonts available on the system.

It is possibile to add **paragraph options** including:

- List Disc
- List Circle
- ListSquare
- ListDecimal
- ListLowerAlpha
- ListUpperAlpha
- ListUpperRoman
- ListLowerRoman

It is possible to set:

Left Alignment
Center Alignment
Right Alignment
Justified Alignment

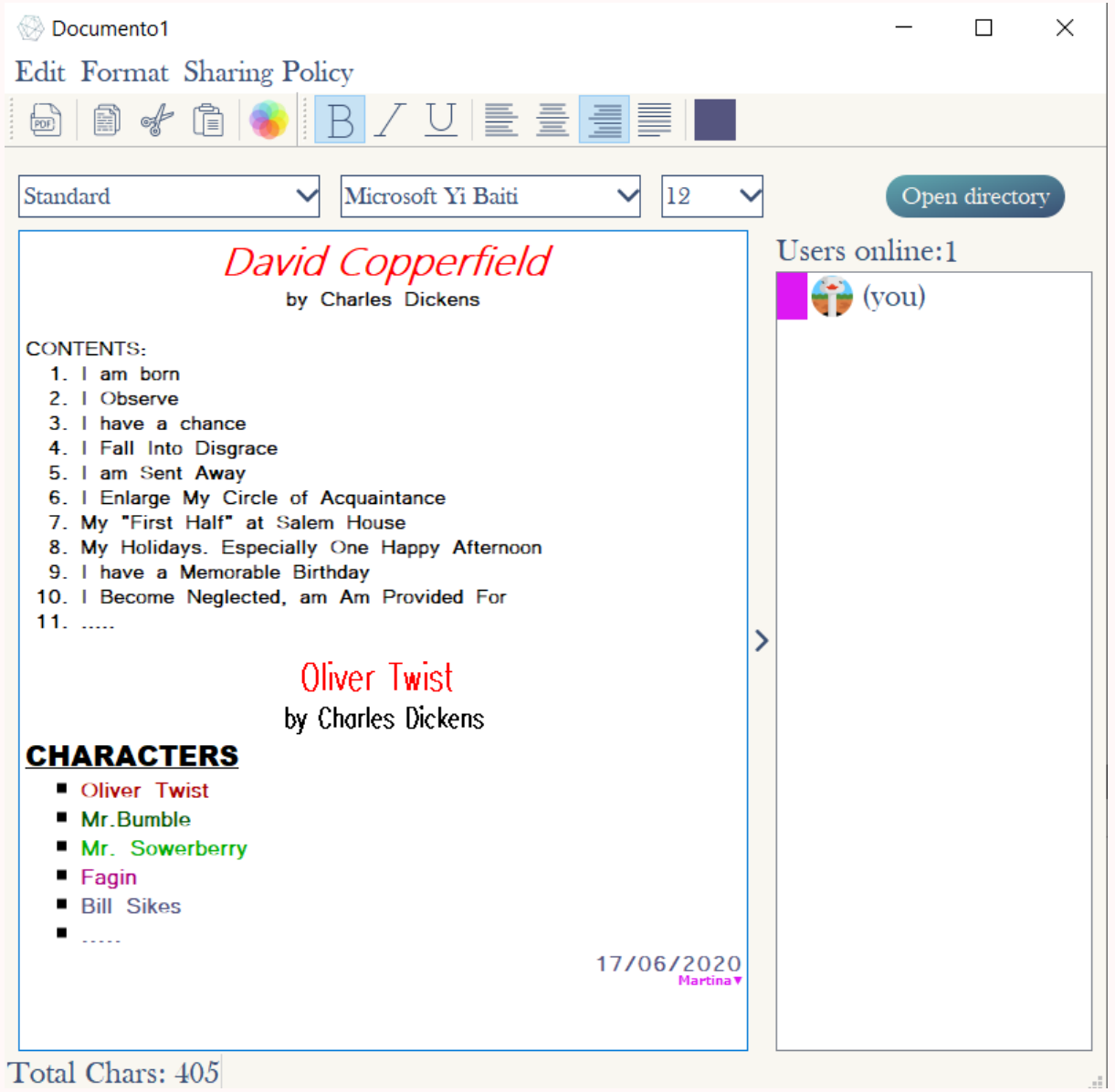


Image taken from Symposium

Characters Highlight

ACTIVATION OF THE FUNCTIONALITY

A user can visualize inside its text edit which users have inserted the characters. To do this, a highlight function has been implemented. A highlight color corresponds to a single user. When the action is requested, the characters will be highlighted with the color corresponding to the user that has inserted them.

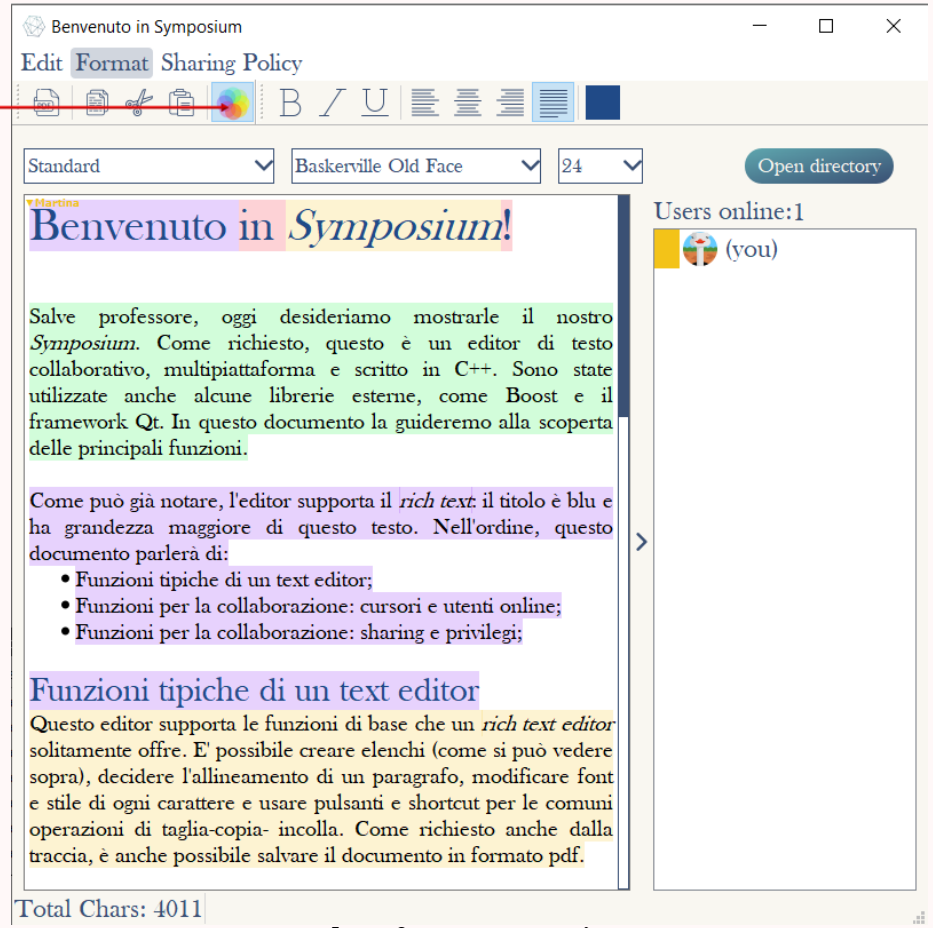
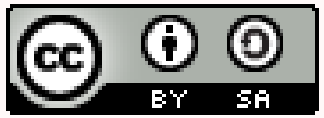


Image taken from Symposium

Thanks for you attention!

License



You are free to:

- **Share** — copy and redistribute the material in any medium or format;
- **Adapt** — remix, transform, and build upon the material for any purpose, even commercially;

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.