Impl Display For Group

Materia: Taller de Programación I
Curso: Deymonnaz

2c 2022

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Impl Display For Group

Integrantes

- Agustin Ariel Andrade 104046 aandrade@fi.uba.ar GitHub
- Tomás Apaldetti 105157 tapaldetti@fi.uba.ar GitHub
- Carolina Di Matteo 103963 cdimatteo@fi.uba.ar GitHub
- Valentina Laura Correa 104415 vcorrea@fi.uba.ar GitHub

Repositorio de Trabajo

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Índice

Introducción

Objetivo

Proyecto IRC (Internet Rust Chat)

➔ Internet Relay Chat Protocol

Alcance

- Parseo de mensajes
- Comunicación Cliente - Servidor
- Soporte para múltiples servidores
- Ejecución de mensajes
- Interfaz Gráfica
- Persistencia de datos
- Testing

¿Cómo levantar el proyecto?

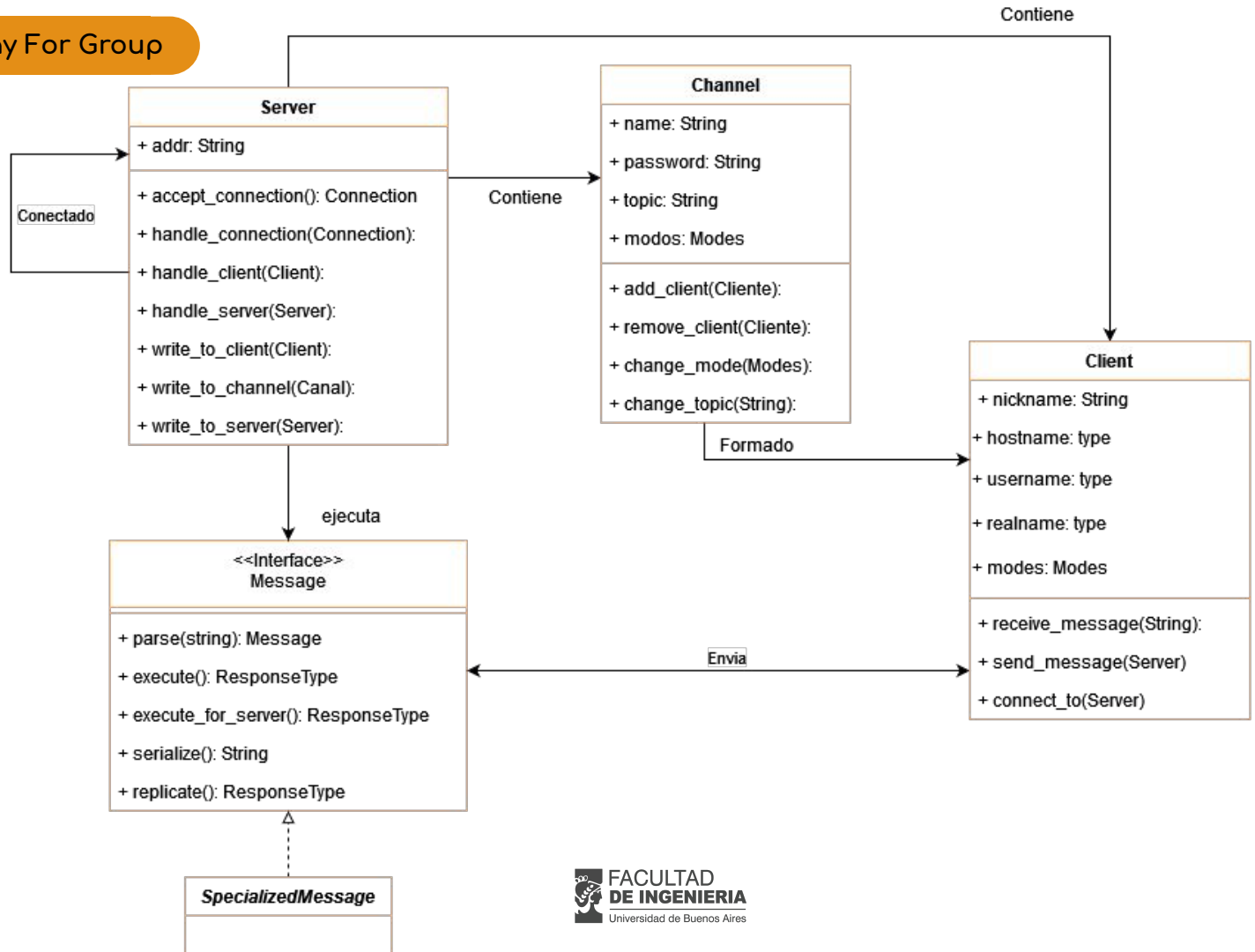
Levantar el proyecto

- cargo run **server** *<puerto>*
- cargo run **client** *<ip cliente> <puerto server>*
- cargo run **server-connect** *<puerto nuevo server> <ip server a conectar> <puerto server a conectar> <contraseña>*

Diagrama de clases

Impl Display For Group

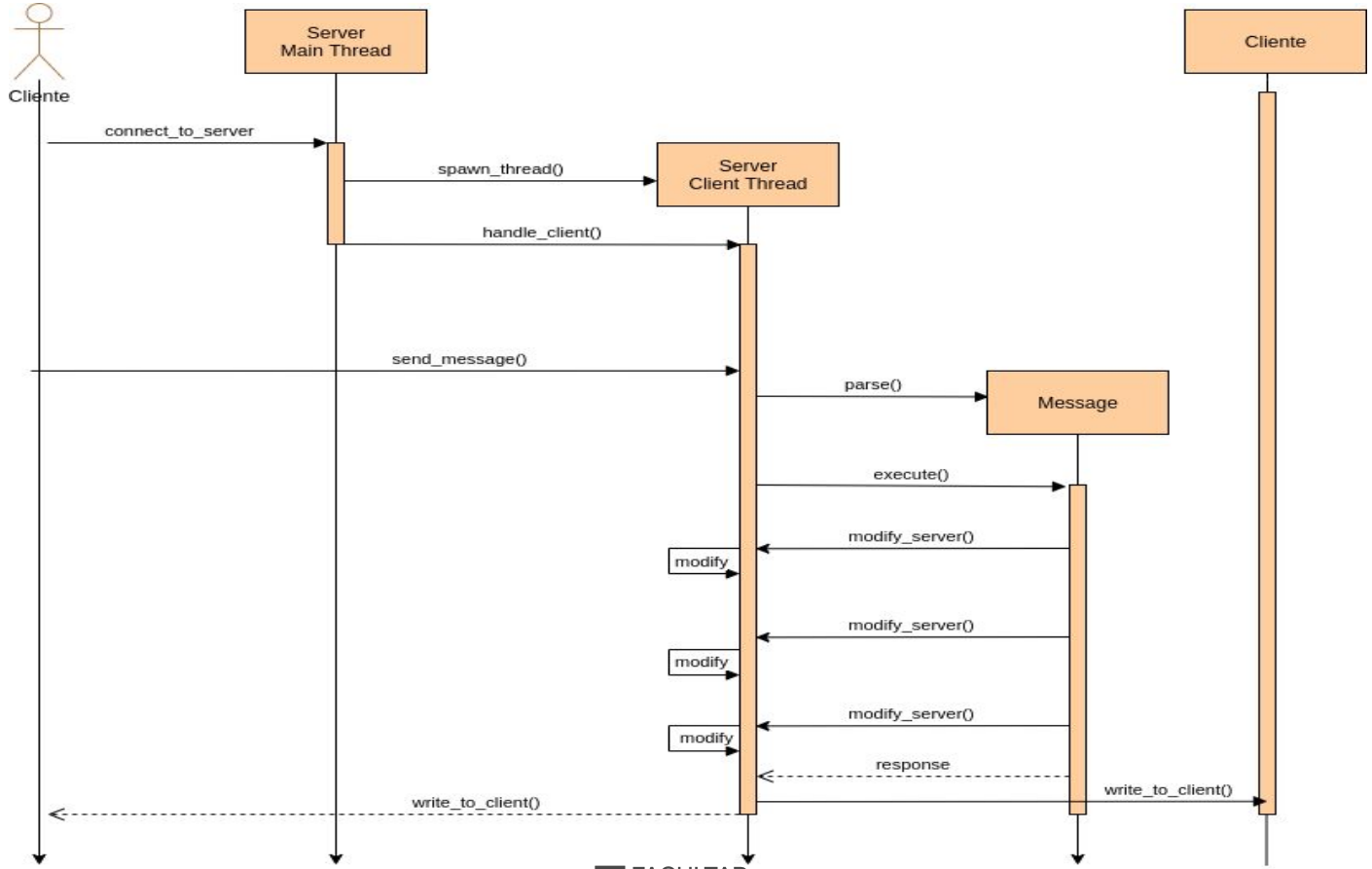
Impl Display For Group



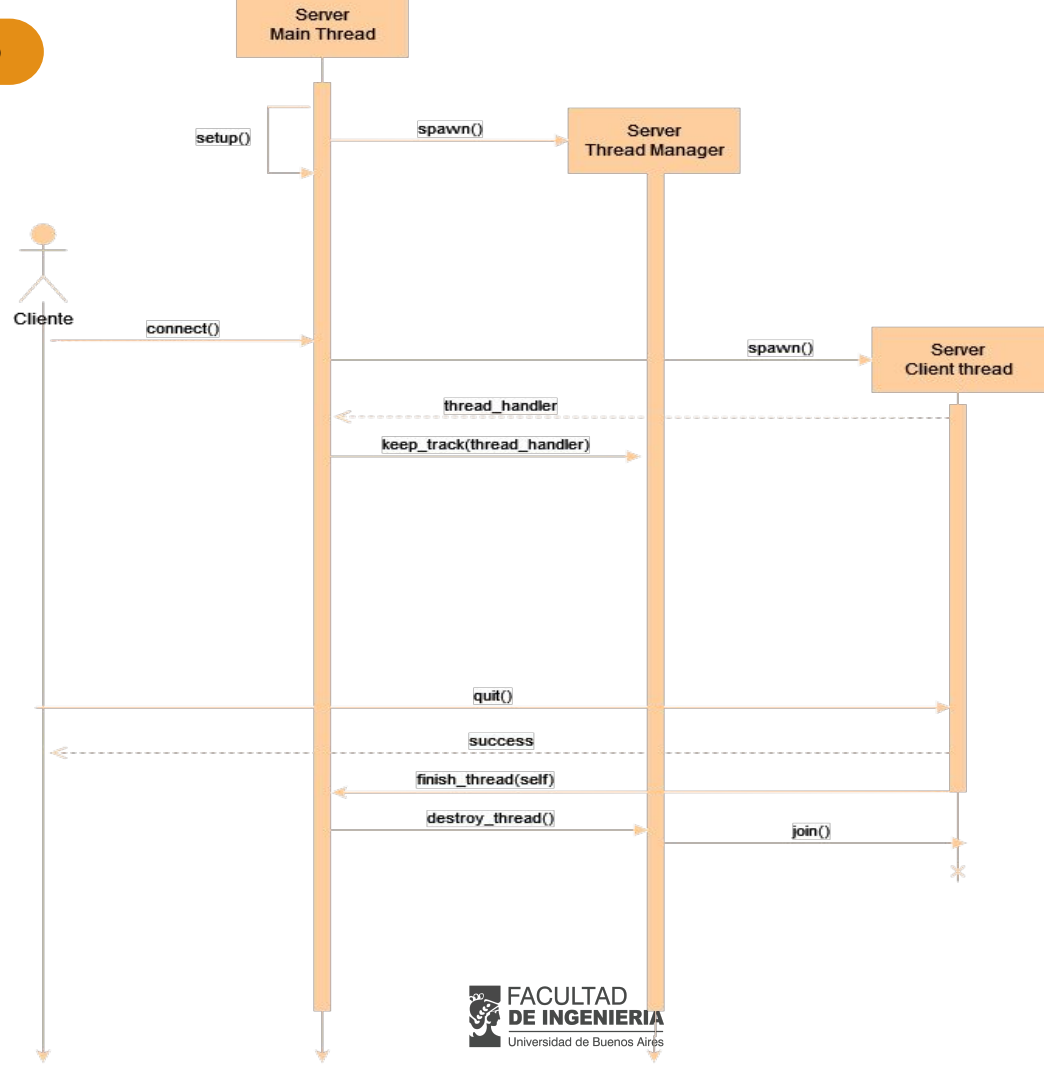
FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Diagramas de Secuencia

Impl Display For Group

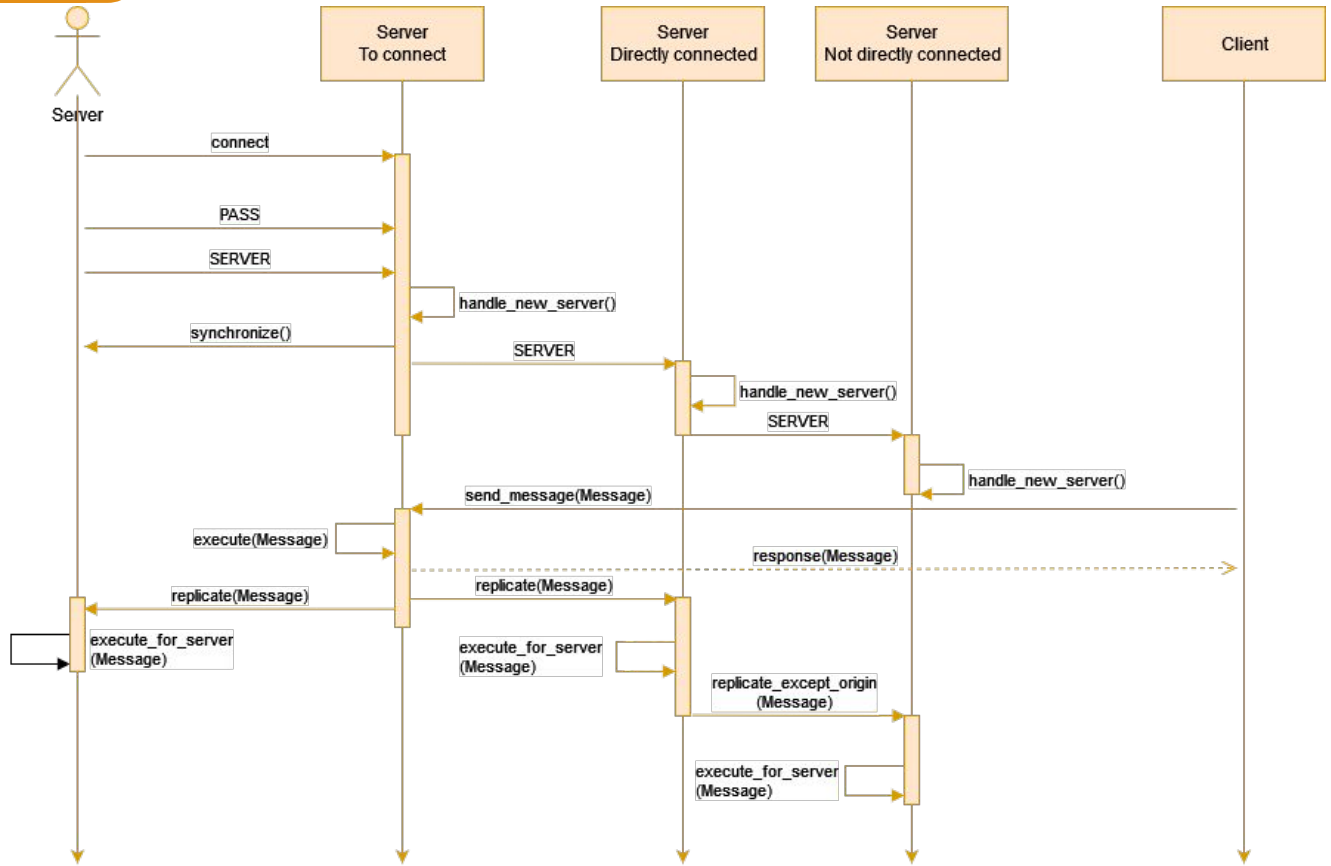


Impl Display For Group



FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Impl Display For Group



Back - Un poco del cómo...

Impl Display For Group

```rust
impl<'a> FromGeneric<'a> for Oper<'a> {
    fn from_generic(mut generic: GenericMessage<'a>) -> Result<Self, MessageError> {
        (...)
        Ok(Self { user, password })
    }
}
```

```rust
pub struct GenericMessage<'a> {
    pub command: Command,
    pub prefix: Option<&'a [u8]>,
    pub parameters: VecDeque<&'a [u8]>,
}
```

```rust
pub struct Oper<'a> {
    pub user: &'a [u8],
    pub password: &'a [u8],
}
```

```rust
pub enum MessageError {
    EmptyMessage,
    MessageTooLong,
    InvalidCommand,
    InvalidFormat,
    InvalidHostmask,
    TooManyParams,
    IRCDefined(usize),
}
```


Impl Display For Group

```rust
impl Executable for List<'_> {
    fn _execute(&self, server: &Server, client: Arc<Mutex<Client>>) -> Vec<ResponseType> {
        let mut response = ResponseBuilder::new();
        response = response.add_content_for_response(RPL_LISTSTART, "Channel :Users  Name".to_owned());

        let desc_list = match &self.channels {
        (...)
        for desc in desc_list {
            response = response.add_content_for_response(RPL_LIST, desc);
        }

        response = response.add_content_for_response(RPL_LISTEND, "End of /LIST".to_owned());
        response.build()

}
```

```rust
pub struct ResponseBuilder {
    numeric_response: Vec<usize>,
    content: HashMap<usize, Vec<String>>,
    internal_response: Vec<InternalType>,
}
```

```rust
pub enum ResponseType {
    NoResponse,
    Code(usize),
    Content(Response),
    InternalResponse(InternalType),
}
```


Impl Display For Group

```rust
pub struct Replicability{
  client_response: Vec<ResponseType>,
  should_replicate: bool
}

pub trait Replicable: Serializable {
    fn inner_execute(&self, server: &Server, client: MTClient) -> Replicability;

    fn execute(&self, server: &Server, client: MTClient) -> Vec<ResponseType> {
        let response = self.inner_execute(server, client);
        let forward = append_origin_client(self.serialize(), client);
        if response.should_replicate{
            server.replicate_to_all_servers(&forward);
        }
        response.client_response
    }
}

impl Server {
  pub fn replicate_to_all(&self, message: &str) {
        for server_connection in self.sv_connections.values() {
            if server_connection.hopcount == 1 {
                server_connection.write_line(message);
            }
        }
    }
}
```


Impl Display For Group

```
pub trait ServerExecutable: Serializable{
    fn inner_execute_for_server(&self, server: &Server) -> Vec<ResponseType>;

    fn forward(&self, _: &Server, _: &MTServerConnection) -> String {
        self.serialize()
    }

    fn replicate(&self, server: &Server, origin: MTServerConnection) {
        let forward = self.forward(server, &origin);
        server.replicate_to_all_servers_except_origin(&forward, &origin.servername);
    }

    fn execute_for_server(&self, server: &Server, origin: MTServerConnection) -> Vec<ResponseType> {
        let res = self.inner_execute_for_server(server);
        self.replicate(server, origin);
        res
    }
}

impl Server {
  pub fn replicate_to_all_servers_sans_origin(&self, message: &str, origin: &str) {
        for server_connection in self.sv_connections.values() {
            if server_connection.hopcount == 1 && server_connection.servername != origin {
                server_connection.write_line(message);
            }
        }
    }
}
```


Impl Display For Group

```
impl Server {
  fn launch_persistency_thread(server: Arc<Server>, tx: Sender<ServerCommand>) ->
  (JoinHandle<()>,Arc<(Mutex<bool>, Condvar)>)
  {
        let exited = Arc::new((Mutex::new(false), Condvar::new()));
        let pair = exited.clone();
        (thread::spawn(move || {
            let (lock, cvar) = &*exited;
            let mut exited = try_lock!(lock);
            loop {
                match cvar.wait_timeout(exited, Duration::from_secs(PERSIST_TIMER)) {
                    Ok((v, _)) => {
                        exited = v;
                        if *exited {
                            break;
                        }
                        persist_notice(server.clone(), tx.clone());
                    }
                    Err(e) => {
                        exited = e.into_inner().0;
                    }
                }
            }
        }), pair)
    }
}
```


Front - Un poco del cómo...

Impl Display For Group

```
let (tx2, rx2) = channel();
```

```
run_app(tx1, rx2);
```


```
let (tx1, rx1) = channel();
```


Impl Display For Group

```rust
fn send_message_from_entry(tx: Arc<Sender<String>>, e: &Entry) -> Result<String, SendError<String>> {
    let mssg = String::from(e.text());
    e.set_text("");
    match tx.send(mssg.to_owned()) {
        Ok(_) => Ok(mssg),
        Err(e) => Err(e),
    }
}
```


Impl Display For Group

```rust
fn receive_from_server(rx: Arc<Receiver<String>>) -> Vec<String> {
    let mut responses: Vec<String> = Vec::new();
    while let Ok(response) = rx.recv_timeout(time::Duration::from_millis(100)){
        responses.push(response);
    }
    responses
}
```


Impl Display For Group

```
    // Responses del Server
    glib::timeout_add_local(time::Duration::from_millis(200), move || {
        let text = check_responses(r.clone());
        let title = w.title().unwrap_or_else(|| GString::from(""));
        if !title.contains(" - ") {
            let nick = check_nickname(&text);
            if !nick.is_empty() { w.set_title(&format!("{} - {}", title, nick)) }
        }
        if !text.is_empty() { append_on_buffer(b.clone(), &text); }
        Continue(true)
    });
```


Impl Display For Group

```
// Build Channels
glib::timeout_add_local(time::Duration::from_secs(3), move || {
    check_channels(t.clone(), r.clone(), &c);
    Continue(true)
});
```


Impl Display For Group

```
    clistbox.connect_add(move |_clistbox: &ListBox, widget: &Widget| {
        let button_widget = w.downcast::<gtk::Button>().expect("Couldn't get button for channel");
        button_widget.connect_clicked(clone!(@weak button_widget => move |_|
            let cname = button_widget.label().expect("Couldn't get name for channel").to_string();

            // Limpio Pantalla
            set_text_on_buffer(bb.clone(), "");

            // Pongo el titulo del channel
            cc.set_text(&cname);

            // Escucho por usuarios del channel
            glib::timeout_add_local(time::Duration::from_secs(3), move || {
                check_users_for_channel(ttt.clone(), rrr.clone(), uuu.clone(), cname.to_owned());
                Continue(true)
            });
        ));
    });
```


Demo time!

Bibliografía

Bibliografía

- RFC1459
- docs.rs
- rust-by-example
- GTK introduction
- Material del curso

¿Preguntas?