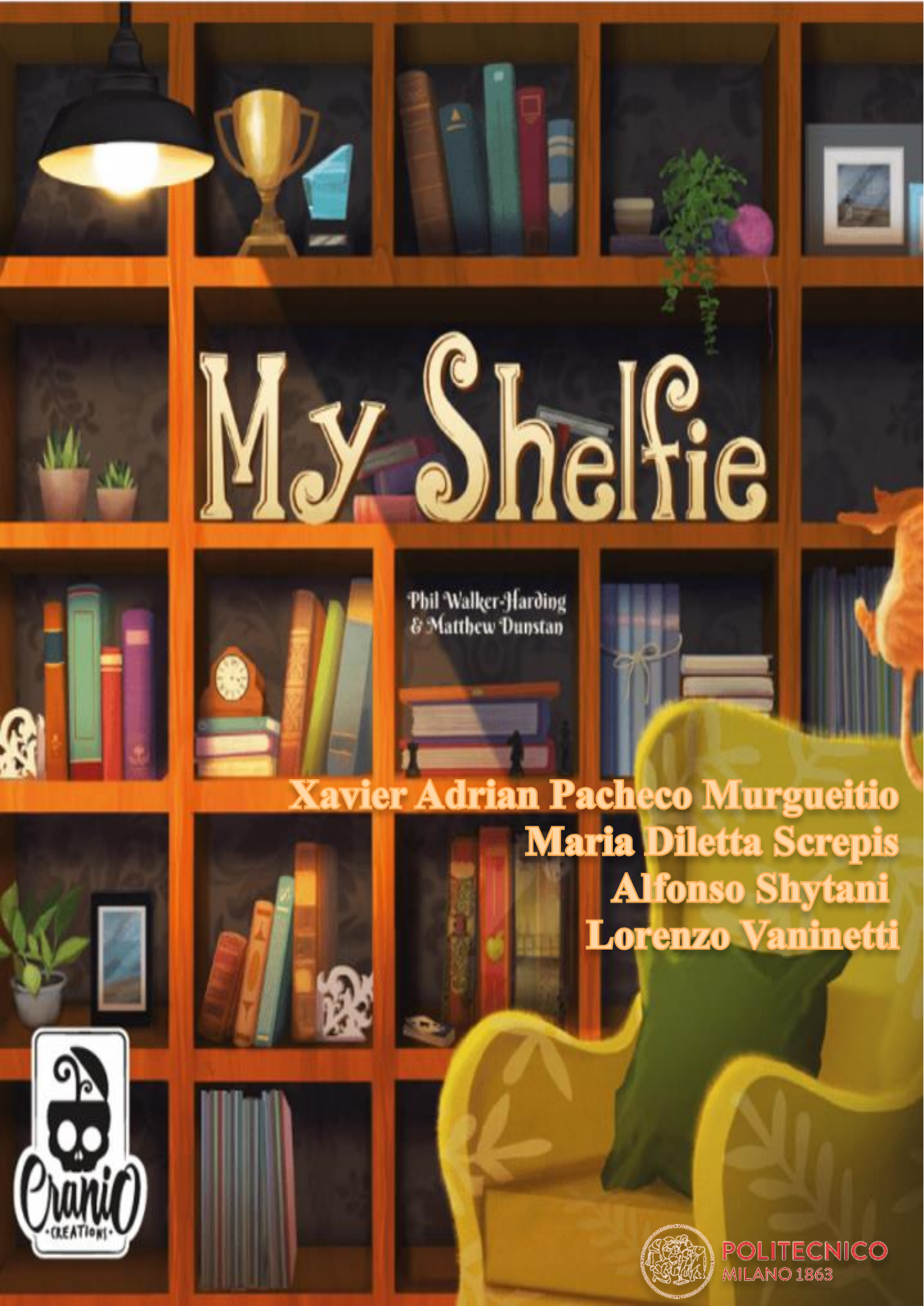
My Shelfie
Phil Walker-Harding
& Matthew Dunstan
Xavier Adrian Pacheco Murgueitio
Maria Diletta Screpis
Alfonso Shytani
Lorenzo Vaninetti
CranIO
CREATIONS
POLITECNICO
MILANO 1863

1 Presentazione UML - Network

Il testo, per quanto breve, focalizza l'attenzione sugli aspetti fondamentali per la comprensione dell'implementazione del protocollo di rete. L'UML del Network, diviso nelle sue componenti ServerSide e ClientSide, presenta sei classi pubbliche principali e due ulteriori classi necessarie a gestire il funzionamento della componente view lato client. Riteniamo doveroso specificare che al momento abbiamo deciso di mettere a disposizione degli utenti la sola interfaccia testuale e che il collegamento server-client avviene solo tramite protocollo RMI.

Leclassi proposte per lo sviluppo funzionale del protocollo di rete sono le seguenti:

1. Client;
2. RemoteClientInterface;
3. RemoteClientImplementation;
4. RMIServer;
5. RemoteServerInterface;
6. RemoteServerImplementation;
7. View;
8. TUI;

ClientSide

Client è la classe astratta utilizzata per definire gli scheletri dei metodi che ogni client è in grado di mettere a disposizione dell'utente e del server. Estende la classe UnicastRemoteObject per permettere la generazione dello skeleton necessario al corretto funzionamento del protocollo RMI. La classe Client garantisce flessibilità e permette inoltre di implementare in modo veloce ed efficace sia una connessione tramite RMI che una tramite Socket.

RemoteClientInterface è un'interfaccia che corrisponde all'oggetto remoto che il client mette a disposizione del server affinchè quest'ultimo possa inviargli dei messaggi. **RemoteClientImplementation** rappresenta invece il client RMI vero e proprio. In quanto Observer della componente view esso viene notificato di tutte le scelte che l'utente opera attraverso le UI e le trasmette attraverso la rete al server RMI a cui è opportunamente collegato in fase di login del client. **RemoteClientImplementation** lavora anche in verso opposto, trasmettendo le modifiche apportate sul GameModel (che riceve dal server sempre tramite protocollo RMI) alla view perché esse possano venir mostrate all'utente.

ServerSide

Come già esplicitato, abbiamo al momento deciso di occuparci solo della costruzione della connessione tramite RMI. Per questo motivo, **RMIServer** rappresenta la classe preposta alla gestione del server e alla creazione dell'RMI Register necessario. **RemoteServerInterface** ha un comportamento speculare a quanto visto per **RemoteClientInterface**: è l'oggetto remoto, fornito ai client, contenente la lista dei metodi e delle funzionalità che il server mette a disposizione. **RemoteServerImplementation** è un Observer della classe GameModel e trasmette le modifiche su di esso attraverso la rete RMI direttamente ai client. Come RemoteClientImplementation, anche RemoteServerImplementation lavora in entrambi i sensi: si occupa di trasmettere le indicazioni, fornite dall'utente e ricevute dai client, direttamente a GameController che, seguendo il paradigma MVC, si occuperà di apportare le modifiche su GameModel.

View è una interfaccia creata per avere flessibilità e maggior facilità di implementazione nell'evenienza di una possibile scelta tra TUI e GUI in fase di inizializzazione del client. **TUI** al momento è l'unica UI disponibile: in quanto Observable essa notifica le scelte del player a **RemoteClientImplementation** e, dall'altro lato, si occupa di fornire all'utente la rappresentazione fedele dello stato della partita grazie alle informazioni che riceve in tempo reale dal server sulle modifiche apportate al GameModel.