Programmazione di sistema

Anno accademico 2018-2019

Esercitazione 3

Facendo riferimento all'algoritmo CRDT descritto nel progetto del corso, si implementino le seguenti classi.

La classe **NetworkServer** simula il comportamento di un sistema di rete. Internamente essa mantiene un vettore di puntatori ad oggetti SharedEditor, una coda di oggetti di tipo Message, che rappresentano informazioni da trasmettere verso i diversi client.

Essa offre i seguenti metodi pubblici:

- `int connect(SharedEditor* sharedEditor);`
  Registra, nel proprio vettore di puntatori a SharedEditor, il puntatore ricevuto e restituisce un identificatore univoco mediante il quale l'editor potrà distinguersi dalle altre istanze della stessa classe
- `void disconnect(SharedEditor* sharedEditor);`
  Elimina il puntatore dal vettore degli editor collegati.
- `void send(const Message& m);`
  Aggiunge il messaggio m alla coda dei messaggi da distribuire, senza ancora mandarlo.
- `void dispatchMessages();`
  Distribuisce tutti i messaggi accodati a tutti gli editor attualmente collegati, fatta eccezione, però, per l'originatore del messaggio stesso.

La classe **Symbol** modella un singolo carattere all'interno dell'editor; oltre al carattere vero e proprio, al suo interno contiene un identificativo univoco (costituito dall'id associato all'istanza del client che lo ha generato e un numero progressivo definito da quest'ultimo) e una posizione frazionaria, rappresentata mediante un `std::vector<int>`. Tale posizione è tale per cui due elementi consecutivi all'interno dell'editor devono avere posizione frazionaria strettamente crescente.

La classe **SharedEditor** modella una istanza di un editor condiviso. Essa contiene, al proprio interno diverse informazioni private:

- `NetworkServer& _server;`
  Rappresenta l'istanza condivisa del server di rete. Viene passato al costruttore che provvede ad invocarne il metodo `connect(…)` allo scopo di ottenerne un `siteId`
- `int _siteId;`
  Identificatore univoco, assegnato all'atto della connessione con il server
- `std::vector<Symbol> _symbols;`
  Contiene la sequenza di tutti i simboli presenti nel documento.
- `int _counter;`
  Contiene un numero che viene via via incrementato e utilizzato per creare, insieme a `_siteId` un identificativo univoco a tutti i caratteri che vengono inseriti, attraverso questa istanza di editor, nel documento.

Tale classe offre i seguenti metodi pubblici:

- `void localInsert(int index, char value);`
  Questo metodo costruisce un symbol che incapsula il carattere value, gli associa un identificativo univoco ed una posizione frazionaria tale da risultare compresa tra le posizioni frazionarie degli elementi di `_symbols` all'indice `index-1` e `index` (se esistenti). A seguito dell'inserimento, nell'indice `index` verrà a trovarsi il nuovo simbolo, e tutti quelli successivi scaleranno di una unità. Le loro posizioni frazionarie, tuttavia, non verranno modificate e si manterrà sempre l'invariante tale per cui tutti gli elementi adiacenti nel vettore `_symbols` hanno posizione frazionaria strettamente minore l'una dell'altra. A seguito dell'inserimento, prepara un oggetto di tipo `Message`, all'interno del quale descrive l'azione compiuta: inserimento da parte di `_siteId` del carattere `value`, con identificativo univoco e posizione frazionaria corrispondente. Tale messaggio verrà inviato all'oggetto `_server` che provvederà ad accodarlo, in attesa di inviarlo a tutte le altre istanze della classe `SharedEditor`.

- `void localErase(int index);`
  Questo metodo elimina dal vettore `_symbols` l'elemento all'indice indicato, prepara un oggetto di tipo Message in cui descrive l'azione compiuta e lo affida all'oggetto `_server` affinché lo consegni agli altri `SharedEditor`.

- `void process(const Message& m);`
  Questo metodo esamina il contenuto del messaggio m e provvede a eseguirne le relative azioni: se si tratta di un messaggio di inserimento provvede ad identificare, a partire dalla posizione frazionaria contenuta nel messaggio, l'indice nel vettore `_symbols` in cui inserire il nuovo simbolo; se, invece, si tratta di una cancellazione, cerca nel vettore `_symbols` se è presente alla posizione frazionaria contenuta nel messaggio un simbolo con l'identificatore univoco corrispondente e, nel caso, lo elimina.

- `std::string to_string();`
  ricostruisce la sequenza dei caratteri contenuta nell'editor

Esempio di uso

```
int main() {
    NetworkServer network;
    SharedEditor ed1(network);
    SharedEditor ed2(network);

    ed1.localInsert(0, 'c');
    ed1.localInsert(1, 'a');
    ed1.localInsert(2, 't');

    network.dispatchMessages();
    std::cout<<"ed1: "<<ed1.to_string()<<std::endl;
    std::cout<<"ed2: "<<ed2.to_string()<<std::endl;

    ed1.localInsert(1,'h');
    ed2.localErase(1);

    network.dispatchMessages();
    std::cout<<"ed1: "<<ed1.to_string()<<std::endl;
    std::cout<<"ed2: "<<ed2.to_string()<<std::endl;
```

```
    return 0;
}
```

Output

```
ed1: cat
ed2: cat
ed1: cht
ed2: cht
```

Competenze da acquisire

- Algoritmo CRDT
- Uso dei contenitori della Standard Template Library
- Analisi e soluzione dei problemi