Progetto di Sistemi Distribuiti

Dott. Diego Borsoi
Dott. Filippo Callegari
DMIF, University of Udine, Italy

Version 0.1, 10 aprile 2021

Sommario

The aim of this document is to describe how to write a report for the project-exam for the course Distributed Systems, at the University of Udine. It is also a guideline for the development of the project itself.

This document is very brief and succint, and it is by no means comprehensive of all the informations that should be given about a software project. You are invited to supplement these guidelines as needed to best describe your work.

Capitolo 1

Introduzione

Il progetto in questione riguarda la creazione di un sistema distribuito per la comunicazione fra dispositivi all'interno di una rete sparsa, tramite l'utilizzo di eventi.

1.1 Descrizione del problema

Ogni dispositivo corrisponde ad un nodo della rete ed è caratterizzato da:

- **Id** : numero univoco del nodo.
- **Stato** : tupla di variabili che rappresentano delle specifiche caratteristiche del nodo (es. temperatura di un sensore, stato di accensione di una termocoppia, ecc).
- **Regole** : insieme di regole del tipo ECA (Event Condition Action) che possono attivarsi a seguito di un evento inviato al nodo. Queste regole possono essere di due tipi: locali, l'azione modifica solamente lo stato del nodo in cui si attiva, oppure globale, l'azione viene inviata a tutti i nodi della rete perché venga letta ed, in caso la valutazione della guardia associata sia positiva, eseguita.

$$\{event;\ condition;\ action \mid \text{if } guard \text{ then } action\}$$

Lo stato della rete si evolverà ogni qual volta un evento verrà attivato, andando a sua volta ad innescare eventuali nuovi eventi e creando quindi una sequenza di azioni a cascata.

1.2 Struttura dell'implementazione

La rete in questione ha una struttura a mesh sparsa (cioè ogni nodo sarà al più connesso a un numero di nodi molto basso, rispetto alla totalità). I nodi sono idempotenti in modo tale da avere un sistema fortemente decentralizzato. Le varie comunicazioni fra i nodi sono eseguite al di sopra di connessioni

TCP, in tal modo possiamo garantire la consegna di ogni messaggio nell'ordine prestabilito. Per quanto concerne invece le comunicazioni riguardanti il sistema di *heartbeat* (TODO: link al capitolo), queste vengono eseguite utilizzando connessioni UDP.

1.3 Trasparenze

Di seguito sono descritte le trasparenze che concergono e sono implementate dal sistema:

Trasparenza ai fallimenti: nel momento in cui un nodo fallisce/si disconnette, il resto della rete continua a funzionare normalmente.

Trasparenza alla scalabilità: la rete può espandersi in dimensione senza che il funzionamento dei nodi vari.

Trasparenza alla mobilità: un nodo può spostarsi all'interno della rete senza che il funzionameto suo e degli altri nodi vada a modificarsi.

1.4 Algoritmi

Il sistema implementa solamente due algoritmi:

- **Flooding Algorithm:** viene usato per la comunicazione di un'azione a tutta la rete nel momento in cui in un nodo una regola globale viene attivata.
- **Lamport clock (modificato):** viene utilizzata una versione modificata del Lamport clock per identificare i vari $flood$ eseguiti; questo clock viene incrementato solamente dall'invio (o ricezione) di un messasggio, e non dalle azioni interne ad un nodo.

Il sistema non implementa particolari algoritmi essendo che si vuole realizzare una rete distribuita dove ogni nodo conosce esclusivamente i vicini ed evolve il suo stato solamente a causa di eventi ricevuti tramite dei messaggi.

1.5 Testing

Per testare il sistema verrà utilizzata un'entità *Ambiente*, la quale simulerà:

- la creazione iniziale della rete, caricando da dei file appositi la struttura degli stati, la lista di regole e la conformazione della rete
- la scoperta di nuove connessioni
- variazioni di variabili legate all'ambiente (es. temperatura registrata da un sensore)

- fallimenti di nodi
- ritardi nell'invio di messaggi fra nodi

Ogni modulo verrà testato singolarmente ed infine verranno eseguiti dei test completi del sistema.

1.6 Piano di sviluppo

Le future fasi di sviluppo seguiranno il seguente ordine:

1. Riunione con il committente per convalidare la risoluzione del problema
2. Implementazione ambiente virtuale per la gestione dei nodi
3. Implementazione della struttura del nodo
4. Implementazione sistema *heartbeat*
5. Implementazione del sistema algoritmico
6. Test totale
7. Validazione

Capitolo 2

Analisi

In questo capitolo vengono descritti nel dettaglio requisiti funzionali e non funzionali della soluzione proposta.

2.1 Requisiti Funzionali

I requisiti funzionali individuati sono:

- **Categorizzazione di un nodo:** ogni nodo ha un tipo il quale ne identifica lo stato e le sue regole;
- **Modifica delle regole di un nodo:** ogni tipo di nodo può avere le sue regole, codificabili attraverso la programmazione dello stesso;
- **Modifica dello stato di un nodo:** ogni evento permette di avere o degli *effetti locali* o degli *effetti globali*:
  - ***effetti locali***: la regola va a modificare lo stato interno;
  - ***effetti globali***: la regola può modificare lo stato delle variabili interne, e può generare un evento sugli altri nodi;
- **Aggiunta dinamica di un nodo:** un nodo può essere aggiunto alla rete in qualsiasi momento senza perturbarne la dinamicità, limitando l'aggiornamento ai vicini a cui si collega.
- **Esecuzione di un'azione ricevuta dai vicini:** nel momento in cui un nodo riceve un messaggio dai propri vicini esso va a verificare la soddisfacibilità della guardia (se presente) e nel caso di una valutazione positiva viene eseguita l'azione associata, andando quindi a modificare il proprio stato.
- **Attivazione di una regola:** ogni qual volta avviene un cambiamento nello stato di un nodo, viene eseguito un controllo delle regole, per vedere se gli eventi generati possano attivare una o più regole del nodo; nel caso in cui una regola venga attivata, in base al tipo (locale o globale) viene portata a termine l'azione corrispondente.

2.2 Requisiti Non Funzionali

I requisiti non funzionali individuati sono:

- **Decentralizzazione:** nessun nodo ha il controllo dell'ordine degli eventi, grazie al fatto che ogni nodo è idempotente;
- **Tolleranza ai guasti:** poichè tutti i nodi sono idempotenti, nel momento in cui un nodo si scollega dalla rete, la rete rimanente continua ad operare normalmente;
- **Etereogenità:** fintanto che i nodi aggiunti utilizzano il protocollo descritto, qualunque nodo di qualunque tipo (hardware o categoria) potrà essere aggiunto alla rete;
- **Scalabilità:** l'aggiunta dinamica dei nodi alla rete permette di scalare orizzontalmente con estrema facilità;
- **Trasparenze:** le trasparenze implementate sono quelle descritte al capitolo precedente (paragrafo 1.3).

Capitolo 3

Progetto

In questo capitolo vengono descritti in modo più approfondito l'achitettura del progetto, i moduli, i protoclli e gli algoritmi utilizzati.

3.1 Architettura logica

3.2 Protocolli ed algoritmi

3.3 Architettura fisica e deployment

3.4 Piano di sviluppo

Capitolo 4

Implementation

Details about the implementation: every choice about platforms, languages, software/hardware, middlewares, which has not been decided in the requirements.

Important choices about implementation should be described here; e.g., peculiar data structures.

Capitolo 5

Validation

Check if requirements from Chapter **??** have been fulfilled. Quantitative tests (simulations) and screenshots of the interfaces are put here.

Capitolo 6

Conclusions

What has been done with respect to what has been promised in Chapters 1 and **??**, and what is left out.

Appendice A

Appendix

In the Appendix you can put code snippets, snapshots, installation instructions, etc.

Evaluation

Your system will be judged mainly on how it operates as a distributed system. The primary evaluation will be according to whether your system has the following attributes:

- It should be an interesting distributed system, making use of some of the algorithms we have covered in class for distributed synchronization, replication, fault tolerance and recovery, security, etc.
- The software should be well designed and well implemented, in terms of the overall architecture and the detailed realization.
- You should devise and apply systematic testing procedures, at both the unit and systems levels.
- The system should operate reliably and with good performance, even in the face of failures.

Important, but secondary considerations include:

- Time taken to do the project (the sooner the better, but do not miss details in order to end sooner)
- How nice is the application's appearance: does it have a nice interface or a compelling visual display?