

“Software Engineering”

Course

a.a. 2017-2018

Template version 1.0

Deliverable #3

Lecturer: Prof. Henry Muccini (henry.muccini@univaq.it)

Industrial Robot dashboard

| | |
|-------------|------------------|
| Date | 12/02/2018 |
| Deliverable | Deliverable 3 |
| Team (Name) | Academic Fellows |

| Team Members | | |
|------------------------|----------------------|--|
| Name & Surname | Matriculation Number | E-mail address |
| Leonardo Formichetti | 242150 | leonardo.formichetti@student.univaq.it |
| Samuele Petrucci | 242385 | samuele.petrucci@student.univaq.it |
| Andrea D'Ascenzo | 243358 | andrea.dascenzo@student.univaq.it |
| Sara Rebecca Di Naccio | 244306 | sararebecca.dinaccio@student.univaq.it |
| Miriana Del Cotto | 243741 | miriana.delcotto@student.univaq.it |

Project Guidelines

[do not remove this page]

This page provides the Guidelines to be followed when preparing the report for the Software Engineering course. You have to submit the following information:

- *This Report*
- *Diagrams (Analysis Model, Component Diagrams, Sequence Diagrams, Entity Relationships Diagrams)*
- *Effort Recording (Excel file)*

Important:

- **document risky/difficult/complex/highly discussed** requirements
- *document decisions taken by the team*
- **iterate**: do not spend more than 1-2 full days for each iteration
- **prioritize** requirements, scenarios, users, etc. etc.

Project Rules and Evaluation Criteria

General information:

- *This homework will cover the 80% of your final grade (20% will come from the oral examination).*
- *The complete and final version of this document shall be not longer than 40 pages (excluding this page and the Appendix).*
- *Groups composed of five students (preferably).*

I expect the groups to submit their work through GitHub

Use the same file to document the various deliverable.

Document in this file how Deliverable “i+1” improves over Deliverable “i”.

Project evaluation:

Evaluation is not based on “quantity” but on “quality” where quality means:

- *Completeness of delivered Diagrams*
- *(Semantic and syntactic) Correctness of the delivered Diagrams*
- *Quality of the design decisions taken*
- *Quality of the produced code*

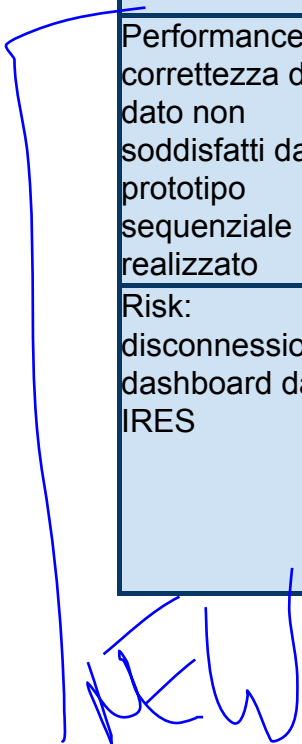
Tabella Contenuti Deliverable 3

- Requisiti (generici) - v3
- Requisiti (specifici): della GUI, della Business Logic, e del DB - v2
- Analysis Model - v3
- Architettura Software di tutto il sistema - v3
- ER Design - v2
- Class Diagram - v2
- Design Decisions - v3
- Mapping - v2
- Effort Estimation - v3
- Prototipo - v2

List of Challenging/Risky Requirements or Tasks

| Challenging Task | Date the task is identified | Date the challenge is resolved | Explanation on how the challenge has been managed |
|---|-----------------------------|--------------------------------|--|
| Estrarre l'inefficiency rate (cioè il downtime) dalla moltitudine di segnali che arrivano | 10/11/2017 | 13/11/2017 | Ideazione del sistema di creazione e gestione delle Finestre Temporali di downtime |

| | | | |
|--|------------|------------|---|
| Visualizzazione comoda del dato sull'interfaccia grafica degli ingegneri | 17/11/2017 | 17/11/2017 | Divisione dell'interfaccia su più layer, pagine, che consentono una visualizzazione del dato sempre più specifica |
| Presenza del dato su 100 terminali | 10/11/2017 | 17/11/2017 | Divisione del sistema in parte "IRES" e parte "Dashboard", connesse tramite tecnologia Ethernet |
| Ricezione ed invio dei dati | 12/11/2017 | 18/11/2017 | Ideazione dei moduli di invio e ricezione dati, originariamente pensati come parte integrante di IRES |
| Problemi implementativi di codifica relativi all'utilizzo di strutture dati dizionari e conversioni fra tipi aritmetici. | 14/12/2017 | 15/12/2017 | Uso delle Stringhe al posto degli array di caratteri come chiavi dei dizionari, pagando per ora un prezzo di complessità spaziale ma velocizzando la stesura del prototipo; learning su internet sulle conversioni di tipo coinvolte nelle operazioni aritmetiche, al fine di calcolare correttamente l'IR e approssimarlo per eccesso alla prima cifra decimale come da specifica. |
| Calcolo effettivo dell'IR di robot e cluster | 12/12/2017 | 13/12/2017 | Progettazione di un algoritmo adeguato che crea finestre temporali di down corrispondenti: per i robot, ai sensori che vanno down; per i cluster, ai robot che vanno down. Fondamentale per l'algoritmo è il fatto che le finestre vengano "chiuso" da un timestamp destro solo quando: per i robot, tutti i sensori ritornano up; per i cluster, tutti i robot ritornano up. |
| Performance e correttezza del dato non soddisfatti dal prototipo sequenziale realizzato | 06/02/2018 | 11/02/2018 | Strutturazione del sistema in modo parallelo, rendendo ogni azione fondamentale indipendente su flusso di codice separato. Utilizzo di Gson che rende più efficiente la serializzazione in fase di invio del dato. |
| Risk: disconnessione dashboard da IRES | 05/02/2018 | 05/02/2018 | Non essendoci molto da fare per IRES in una tale circostanza, è stato semplicemente implementato un componente che apre un canale di comunicazione con le dashboard quando si connettono e lo chiude quando si disconnettono, sia per chiusura del programma o crash. |



| | | | |
|-------------------------------|------------|------------|--|
| Risk: disconnessione DB | 05/02/2018 | 05/02/2018 | Risk affrontato già a valle della realizzazione del prototipo con struttura asincrona. Quando il componente di IRES che interagisce con il DB scopre che quest'ultimo è down avvisa gli altri e le query vengono tenute in RAM finché è possibile flusharle di nuovo e continuare il calcolo e l'invio del dato. |
|-------------------------------|------------|------------|--|

A. Requirements Collection

A.1 Functional Requirements

Estratti dai requisiti formali:

1. Il sistema dovrà calcolare l'inefficiency rate (IR) dei Robot sulla base dei segnali di up e down che essi invieranno secondo questa formula:

$$IR_{Robot} = DN_{Sensori\ totale} / finestra\ temporale$$

dove per " $DN_{Sensori\ totale}$ " si intende il tempo di Down calcolato in modo cumulativo per ogni segnale del Robot all'interno della "*finestra temporale*", che è l'intervallo di tempo su cui calcolare il dato;

2. Il sistema dovrà calcolare l'inefficiency rate dei Cluster sulla base dell'inefficiency rate dei Robot al loro interno, seguendo questa formula:

$$IR_{Cluster} = DN_{Robot\ totale} / finestra\ temporale$$

dove per " $DN_{Robot\ totale}$ " si intende il tempo di Down calcolato in modo cumulativo per ogni Robot all'interno del Cluster, nell'intervallo di tempo "*finestra temporale*".

Requisiti della GUI:

3. Il sistema dovrà permettere di visionare la lista di Robot all'interno di uno specifico Cluster.
4. Il sistema dovrà permettere di visionare una lista dei Cluster attivi.
5. Il sistema metterà in evidenza Cluster e Robot con IR superiore ad una soglia impostabile;

Requisiti della GUI individuati dal team di sviluppo:

6. Il sistema dovrà permettere la ricerca di uno specifico Robot tramite il suo codice identificativo alfanumerico;
7. Il sistema dovrà permettere la ricerca di uno specifico Cluster tramite il suo codice identificativo alfanumerico;
8. Il sistema dovrà permettere di visionare la percentuale di Robot aventi ciascun sensore down all'interno di un Cluster.
9. Il sistema dovrà permettere di visionare i sensori Down di uno specifico Robot.

Requisiti di Logica di business:

10. Il sistema deve essere progettato affinché le azioni fondamentali relative alla logica di business siano compiute in quest'ordine:

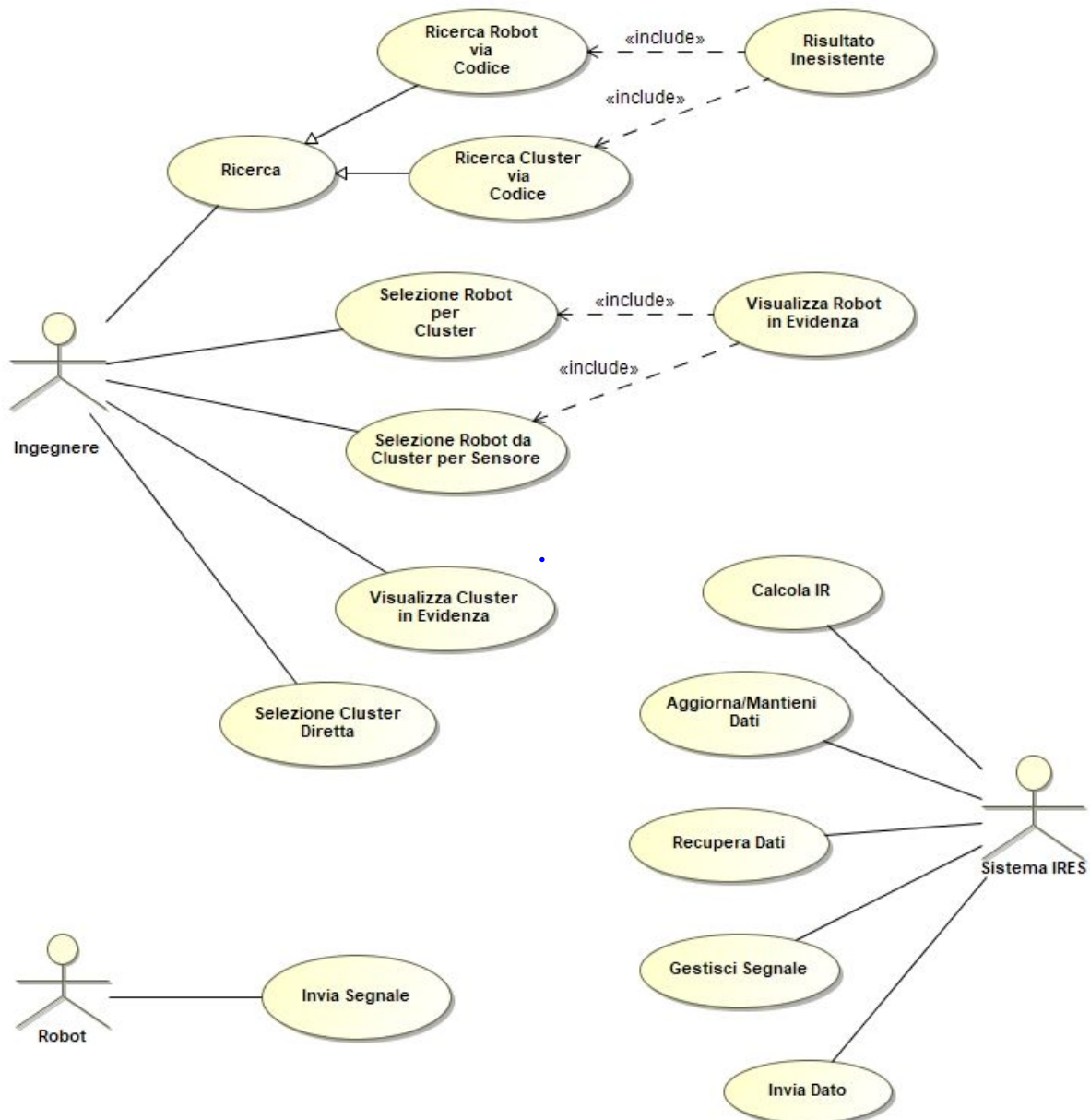
- 1.1 Ricezione e decodifica dei segnali;
- 1.2 Analisi segnali;
- 1.3 Calcolo IR a partire dai segnali analizzati;
- 1.4 Invio del dato alle Dashboard;

Le operazioni 1.2, 1.3. e 1.4. hanno infatti come condizione preliminare lo svolgimento corretto delle rispettive precedenti.

Requisiti di Mantenimento Dati:

11. Il sistema deve garantire il mantenimento dei segnali inviati dai Robot: ipotizzando che la finestra temporale di calcolo dell'IR sia uguale ad un'ora, il sistema dovrà essere in grado di gestire segnali per un periodo di tempo maggiore, al fine di mantenere l'ultimo cambio di stato avvenuto.

Use Case Diagrams



A.1.1 Use Case Diagram del Sistema.

1. Lo use-case **Ricerca** rappresenta il caso in cui un ingegnere, a conoscenza di un codice identificativo, voglia visualizzare direttamente l'oggetto d'interesse. I due casi d'uso che ne derivano sono **Ricerca Robot/Cluster via Codice** che possono portare entrambi alla pagina con i dettagli desiderati oppure ad una schermata di errore, indice del fatto che il Robot/Cluster cercato non è registrato nel sistema. Questo use-case non deriva direttamente da un

requirement di sistema, ma da uno della GUI di cui il team ha ritenuto utile l'inserimento.

2. Lo use-case **Selezione Robot per Cluster** rappresenta il caso in cui un ingegnere voglia visualizzare l'elenco dei Robot all'interno di uno specifico Cluster, messo magari in evidenza dal sistema stesso poiché avente un IR elevato. Questo caso conduce allo use-case **Visualizza Robot in Evidenza**, che rappresenta la possibilità di controllare i dettagli dei Robot, all'interno del Cluster, possibilmente con tassi di inefficienza elevati.
3. Lo use-case **Selezione Robot da Cluster per Sensore** descrive il caso in cui un ingegnere debba visualizzare l'elenco di Robot all'interno di un Cluster che hanno un determinato Sensore Down. Questo caso d'uso del sistema rappresenta una notevole facilitazione nella gestione dei periodi di manutenzione dei Robot perché permette di confrontarli in base ai loro Sensori in stato Down.
4. Lo use-case **Visualizza Cluster in Evidenza** costituisce la situazione di partenza del sistema. Dalla schermata iniziale dovrà essere infatti possibile visualizzare l'elenco dei Cluster registrati nel sistema e facilmente distinguere quelli che presentano IR da tenere sotto controllo. Il caso d'uso **Selezione Cluster Diretta** è immediatamente collegato al suddetto e rappresenta il caso in cui un ingegnere, allertato dal tasso di inefficienza, voglia controllare i dettagli di un Cluster elencato.
5. Lo use-case **Invia Segnale** rappresenta l'unica funzione che il Robot svolge nei confronti del sistema (tramite ESB), e consiste in un invio broadcast di un segnale contenente informazioni relative ad un suo Sensore (tornato Up o andato Down).
6. Lo use-case **Calcola IR** costituisce il calcolo del dato IR da parte del sistema che dovrà poi essere visualizzato tramite le dashboard.
7. Gli use-cases **Aggiorna/Mantieni Dati** e **Recupera Dati** rappresentano tutti i casi in cui il sistema dovrà comunicare con il sistema di persistenza dati scelto, dall'inserimento di dati all'update fino al recupero dei dati per svolgere calcoli.
8. Lo use-case **Gestisci Segnale** rappresenta le attività preliminari svolte dal sistema all'arrivo di un segnale da un Robot, che vanno dalla decodifica a una prima memorizzazione.

9. Lo use-case **Invia Dato** consiste nell'operazione di invio alle dashboard da parte del sistema IRES del dato contenente il tasso di inefficienza IR.

Use Case – Requisiti Prioritari

Di seguito si riportano i requisiti funzionali (trasformati in use case) che sono stati ritenuti più importanti, sia per motivi di frequenza di utilizzo sia perché da soli permettono una gestione basilare ma efficace dell'ambiente di produzione industriale in considerazione.

Tabella 1: Selezione Cluster Diretta

| Nome Use Case | Selezione Cluster Diretta |
|------------------|---|
| Attori Coinvolti | Ingegnere |
| Pre-condizioni | L'ingegnere, controllando tutti i Cluster dalla pagina principale, vuole entrare nelle specifiche di uno di essi. |
| Flow | <ol style="list-style-type: none">1. L'ingegnere vuole controllare i dettagli di un Cluster che ha attirato la sua attenzione tra quelli listati sulla dashboard.1. L'ingegnere clicca sulla casella del Cluster.1. L'ingegnere è portato sulla pagina dei dettagli relativi a quel Cluster.1. Finita l'esaminazione l'ingegnere preme il pulsante "Indietro" ed è riportato alla pagina principale. |
| Post-condizioni | Successo - L'ingegnere entra nella pagina del Cluster e visualizza i dettagli. Fallimento - / Garanzie Minime - I dati sono corretti e con aggiornamento risalente a non più di 5 minuti dall'ora corrente |

A.1.T1 Forma Tabellare dello Use Case "Selezione Cluster Diretta"

Tabella 2: Selezione Robot da Cluster per Sensore

| Nome Use Case | Selezione Robot da Cluster per Sensore |
|------------------|--|
| Attori Coinvolti | Ingegnere |

| | |
|------------------------|---|
| Pre-condizioni | L'ingegnere vuole controllare i dettagli di un gruppo di Robot (appartenenti ad un Cluster già selezionato) in base alla percentuale di Down totale su un determinato sensore. |
| Flow | <ol style="list-style-type: none"> 1. L'ingegnere si trova nella pagina dei dettagli del Cluster. 1. Da qui l'ingegnere clicca su uno dei 7 bottoni presenti (i quali mostrano la percentuale di Robot di quel Cluster che hanno il relativo sensore Down). 1. L'ingegnere è portato ad una schermata che lista tutti i Robot suddetti con correlato IR e segnali down. 1. Finita l'esaminazione l'ingegnere preme il pulsante "Indietro" ed è riportato alla pagina del Cluster. |
| Post-condizioni | <p>Successo</p> <ul style="list-style-type: none"> - L'ingegnere visualizza correttamente i dettagli di suo interesse riguardo i Robot di un Cluster. <p>Fallimento</p> <ul style="list-style-type: none"> - / <p>Garanzie Minime</p> <ul style="list-style-type: none"> - I dati sono corretti e con aggiornamento risalente a non più di 5 minuti dall'ora corrente. |

A.1.T2 Forma Tabellare dello Use Case "Selezione Robot da Cluster per Sensore"

A.2 Non Functional Requirements

Performance

1. Il sistema deve essere in grado di gestire almeno un messaggio di cambio stato al minuto per ciascun Robot. Data la presenza di circa 90000 Robot iniziali, si tratta di una media di 1500 possibili segnali al secondo. Si parla di media poiché possono esserci stati che cambiano di rado - i.e. segnale del laser - e segnali che si modificano più frequentemente - i.e. la temperatura di pressione - ;
2. Il sistema deve sostenere la visualizzazione di circa cento dashboard contemporaneamente: dovrà quindi gestire l'invio di dati di output sulla scala delle centinaia;

3. Le operazioni di memorizzazione e aggiornamento dei dati ricevuti dovranno essere abbastanza efficienti da garantire che, sommando il tempo necessario allo svolgimento di queste a quello che il sistema utilizza per calcolare gli IR e aggiornare le Dashboard, non si ecceda un tempo limite di 5 minuti.

Scalabilità

4. Il sistema deve essere espandibile nel caso in cui siano aggiunti Robot nel processo produttivo, e quindi in grado di distinguere autonomamente i nuovi Robot da memorizzare da quelli già presenti all'interno del sistema stesso.
5. Per lo stesso motivo sopracitato il sistema deve garantire scalabilità all'aumentare del numero dei Robot (e quindi dei segnali) presenti, cercando di mantenere i tempi di aggiornamento sempre al disotto dei 5 minuti (compresi tempi di calcolo, gestione del sistema di persistenza scelto e visualizzazione sulle dashboard). Tempi di aggiornamento accettabili sono necessari affinché la gestione dei Robot sia ottimale, ovvero si riesca a mantenere un Up-Time totale superiore al 98%.

Accuratezza

6. Il sistema non deve calcolare e inviare il dato alle dashboard se non è stata ultimata l'analisi dei segnali arrivati fino a quel momento.

Usabilità

7. Ai fini di individuare comodamente Cluster/Robot, ognuno di essi sarà identificato da un codice alfanumerico;

A.3 Content

Le tecnologie utilizzate saranno le seguenti:

1. Database relazionale MySQL;
2. Linguaggio Java;
3. API JDBC per la comunicazione fra IRES e DB;
4. Libreria java.net per la comunicazione fra IRES e Dashboard;
5. Libreria Google gson per l'interfacciamento con i Robot e per l'invio dei dati alle dashboard;

#1) Il team ha eseguito uno stress test consistente nell'esecuzione di operazioni di insert dell'ordine delle centinaia di migliaia in un database relazionale MySQL e le tempistiche si sono sempre mantenute al disotto di 10 secondi. Inoltre questa tecnologia ha permesso al team di non investire ulteriore tempo in una fase di learning addizionale.

#2) Durante le prime fasi di progettazione il team si è reso conto che il sistema sarebbe stato probabilmente impostato sull'esecuzione parallela. Il linguaggio Java pertanto, correlato alle giuste librerie ed API che si trovano a disposizione, offre tutte ciò di cui il team ritiene di aver bisogno per implementare il sistema, con particolare riferimento al sistema dei Thread.

A.4 Assumptions

1. La manutenzione avviene in modo indipendente dal nostro sistema: il nuovo Robot che sostituisce quello da mantenere avrà lo stesso ID di quello sostituito;
2. Quando un Robot viene sostituito i suoi sensori sono tutti up e il sistema lo saprà tramite invio simultaneo di N segnali dove N è il numero di sensori down che il Robot sostituito aveva;
3. I segnali arrivano integri e standardizzati secondo un formato fornito dal team di sviluppo;
4. Il budget per realizzare il sistema è illimitato;
5. All'avvio di ogni Robot, i propri segnali sono tutti up;
6. Quando il calcolo di un inefficiency rate è decimale va approssimato all'intero superiore;
7. Un Robot è considerato down in un certo range temporale se durante quel periodo anche solo uno dei suoi sensori era down;
8. Un Cluster è considerato down in un certo range temporale se durante quel periodo anche solo uno dei suoi Robot era down;
9. La finestra temporale considerata è di un'ora, eventualmente modificabile;

A.5 Prioritization

1. Il sistema dovrà calcolare l'inefficiency rate (IR) dei Robot sulla base dei segnali di up e down che essi invieranno secondo questa formula:

$$IR_{Robot} = DN_{Sensori\ totale} / finestra\ temporale$$

dove per " $DN_{Sensori\ totale}$ " si intende il tempo di Down calcolato in modo cumulativo per ogni segnale del Robot all'interno della "*finestra temporale*", che è l'intervallo di tempo su cui calcolare questo dato;

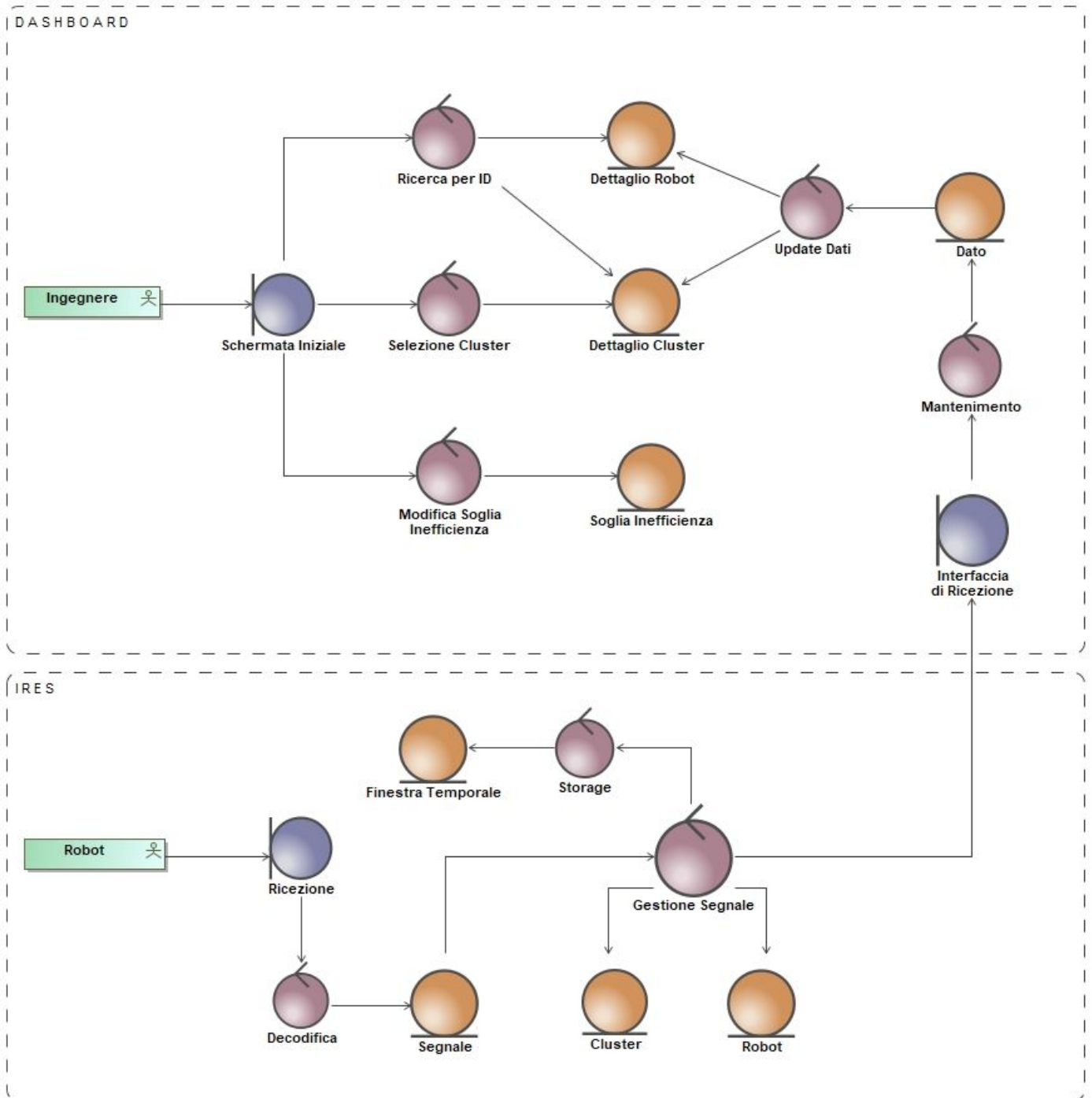
2. Il sistema dovrà calcolare l'inefficiency rate dei Cluster sulla base dell'inefficiency rate dei Robot al loro interno, seguendo questa formula:

$$IR_{Cluster} = DN_{Robot\ totale} / finestra\ temporale$$

dove per “ $DN_{Robot\ totale}$ ” si intende il tempo di Down calcolato in modo cumulativo per ogni Robot all'interno del Cluster, nell'intervallo di tempo “*finestra temporale*”.

3. Il sistema deve garantire il mantenimento dei segnali inviati dai Robot: ipotizzando che la finestra temporale di calcolo dell'IR sia uguale ad un'ora, il sistema dovrà essere in grado di gestire segnali per un periodo di tempo maggiore, al fine di mantenere l'ultimo cambio di stato avvenuto;
4. Le operazioni di memorizzazione e aggiornamento dei dati ricevuti dovranno essere abbastanza efficienti da garantire che, sommando il tempo necessario allo svolgimento di queste a quello che il sistema utilizza per calcolare gli IR e aggiornare la Dashboard, non si ecceda un tempo limite di 5 minuti.
5. Ai fini di individuare univocamente cluster/robot, ognuno di essi sarà identificato da un valore alfanumerico;
6. Il sistema dovrà permettere la ricerca di uno specifico Robot tramite il suo codice identificativo alfanumerico;
7. Il sistema dovrà permettere la ricerca di uno specifico Cluster tramite il suo codice identificativo alfanumerico;

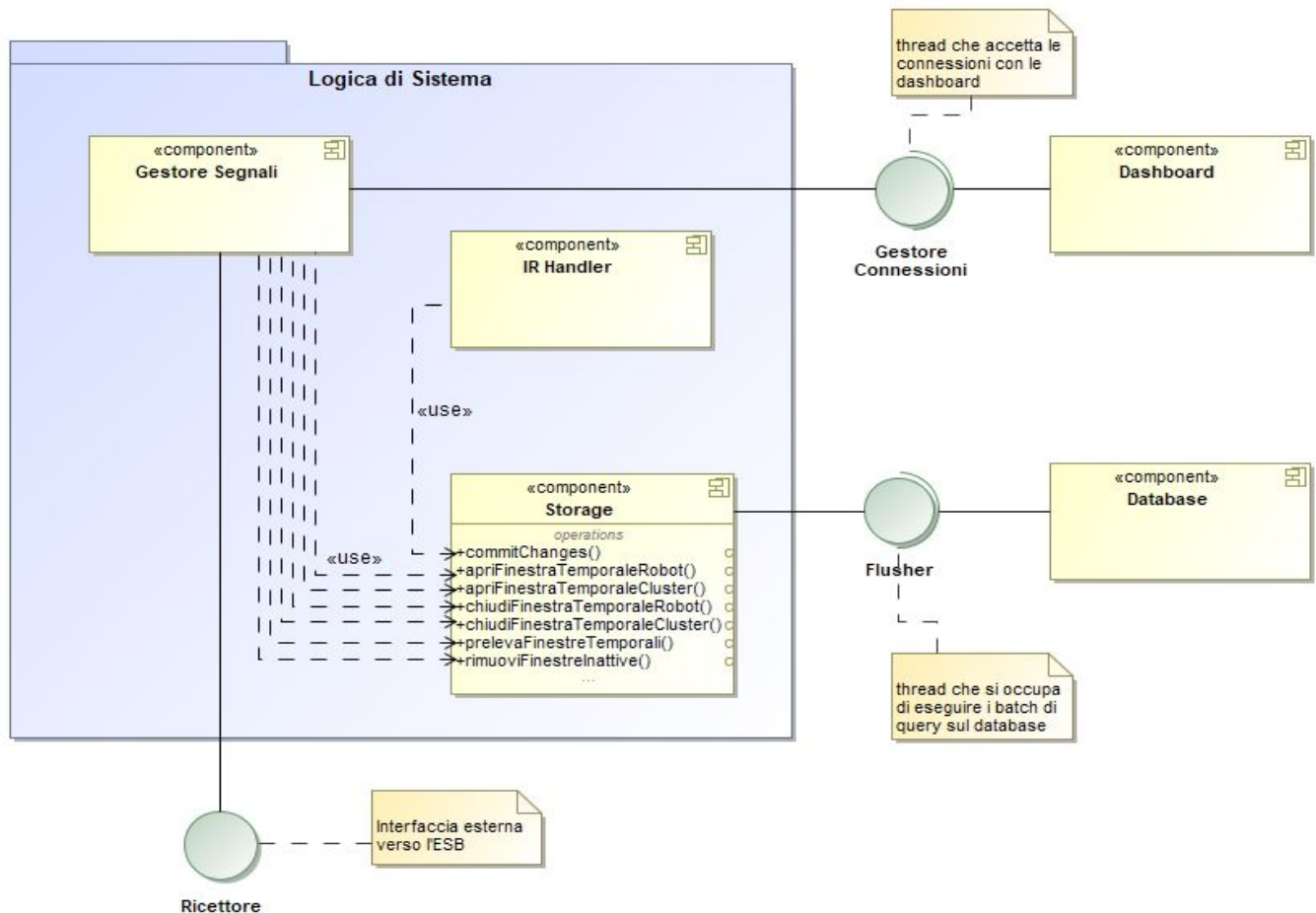
B. Analysis Model



B.1 Robustness Diagram del Sistema

C. Software Architecture

C.1 The static view of the system: Class Diagram and Component Diagram



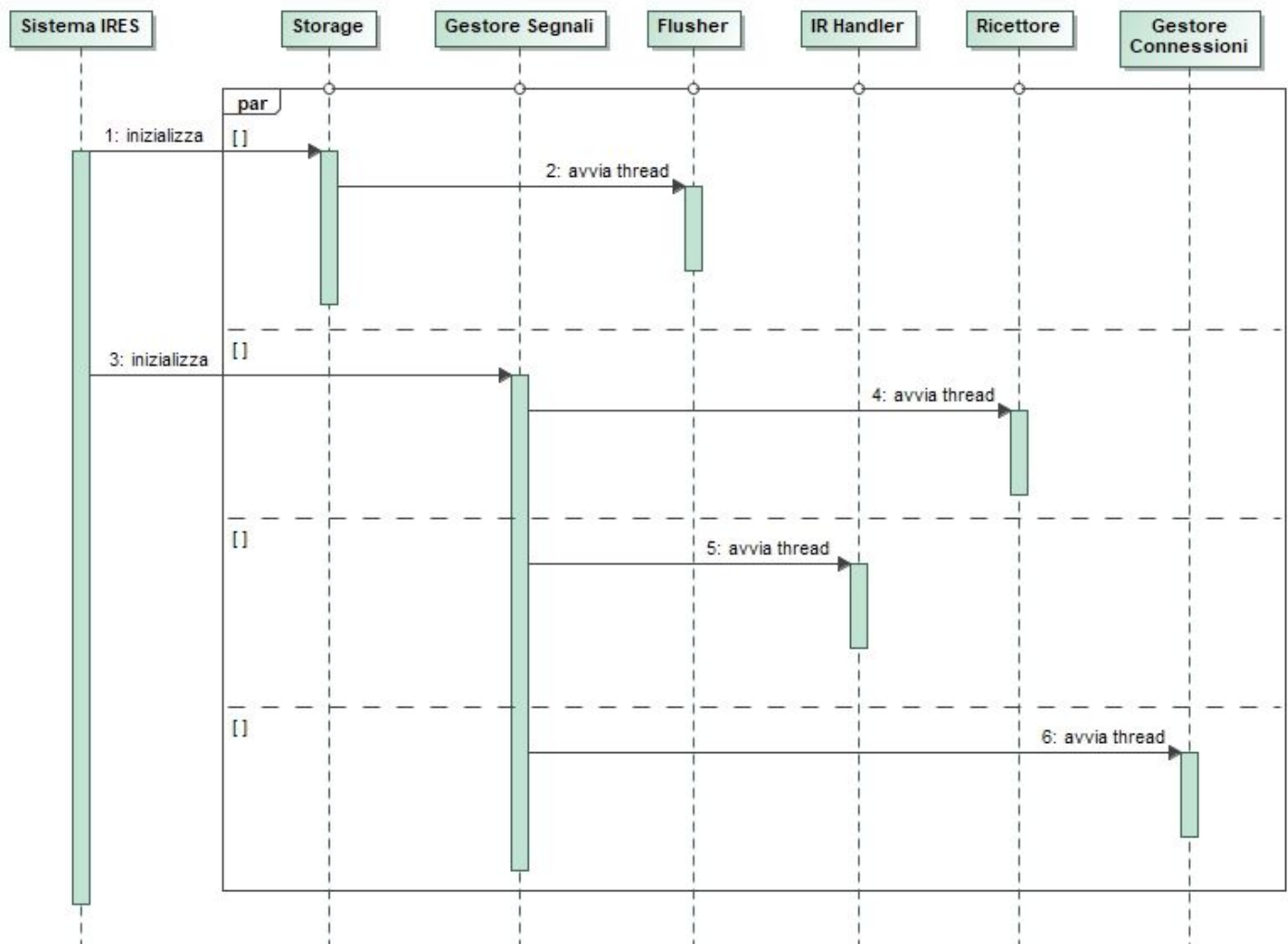
C.1.2 Component Diagram del Sistema

Il component diagram è stato costruito dopo la realizzazione del prototipo:

- Il **Gestore Segnali** è incaricato di analizzare i Segnali ricevuti e, tramite una serie di metodi forniti dalla classe Storage, popolare i batch delle dovute query, che il Flusher eseguirà in blocco autonomamente.
- La componente **Ricettore** (thread) è a tutti gli effetti l'interfaccia che il Gestore Segnali mette a disposizione ai Robot per la ricezione dei segnali. In particolare esso si occupa di ricevere dati dall'ESB.
- Il **Gestore Connessioni** (thread) stabilisce le connessioni tra il sistema IRES e le varie dashboard.

- L' **IR Handler** (thread) si occupa invece di effettuare il calcolo dell'IR e comunicare al Flusher, tramite metodo fornito dalla classe Storage, che ci sono batch pronti per l'esecuzione.
- Il **Flusher** (thread) ha il compito di eseguire i batch di query sul DB ed è pertanto l'unica componente a comunicare con esso. Perciò è corretto dire che costituisce l'interfaccia del sistema verso il Database.

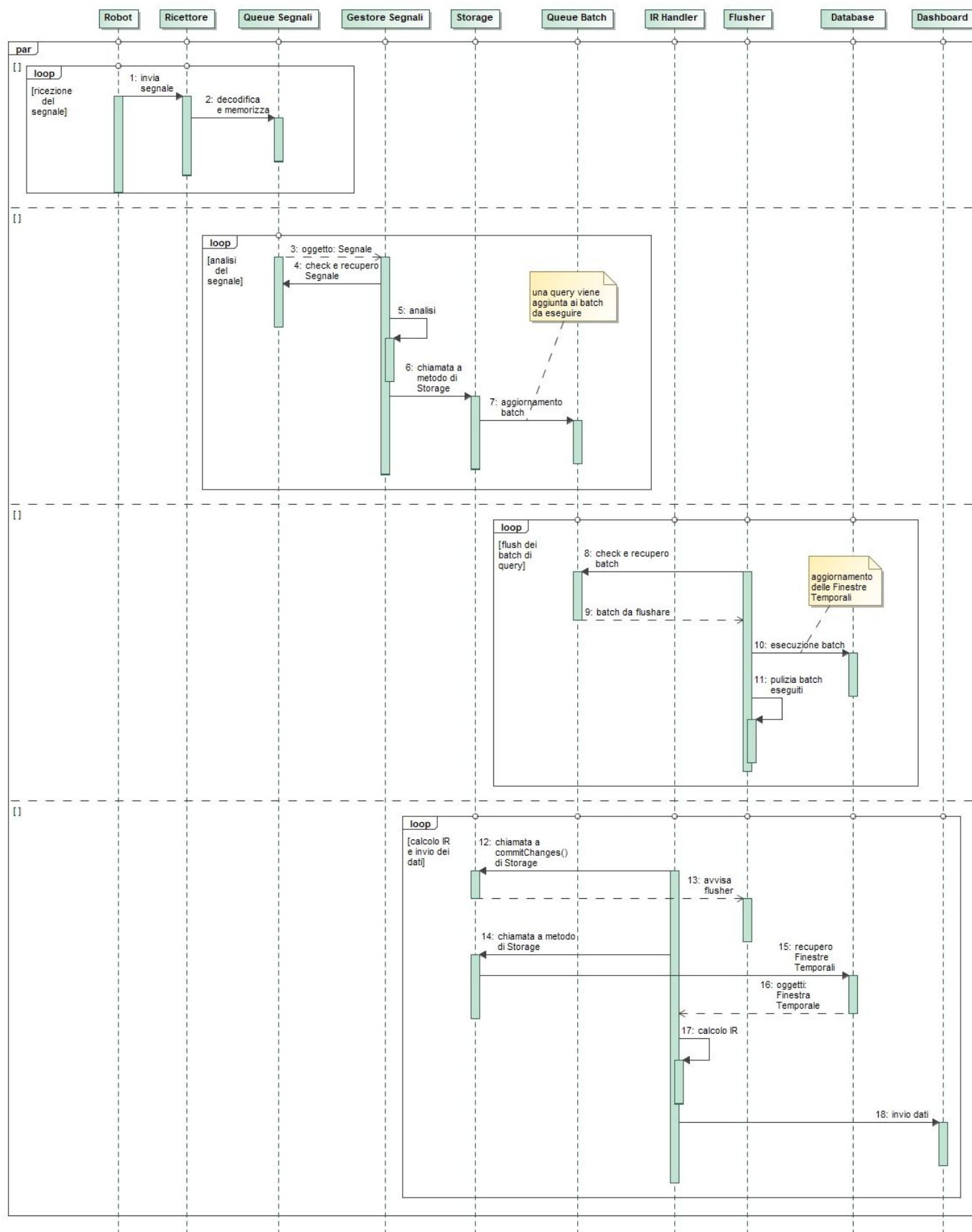
C.2 The dynamic view of the software architecture: Sequence Diagram



C.2.1 Sequence Diagram: Avvio del Sistema IRES

Il sequence diagram in figura C.2.1 rappresenta il modo in cui il sistema IRES inizializza tutte le sue componenti al momento dell'avvio.

La classe Storage è la prima a entrare in funzione, avviando automaticamente il thread Flusher, che poi opererà in modo autonomo. Dopodiché viene inizializzato Gestore Segnali, il quale si occupa di avviare gli altri tre thread: Ricettore, IR Handler e Gestore Connessioni;



C.2.2 Sequence Diagram: Esempio di funzionamento del Sistema Asincrono

La figura C.2.2 rappresenta il funzionamento del Sistema al momento di ricezione di un segnale in arrivo da un Robot. E' evidenziato il parallelismo tra le operazioni fondamentali del sistema.

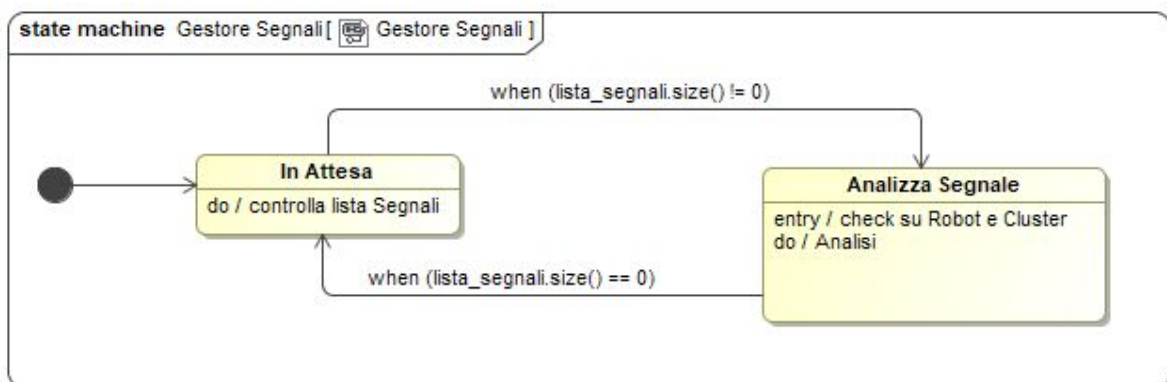
Il Ricettore è il primo sottosistema ad attivarsi: all'arrivo di un segnale lo decodifica e lo inserisce in una coda apposita. Il Gestore Segnali, indipendentemente, è già in attesa che la coda venga popolata di un qualche oggetto da recuperare ed analizzare. A seconda della tipologia di Segnale esaminata il Gestore svolge una funzione apposita sul database tramite la classe Storage.

Contemporaneamente un altro thread (Flusher) controlla se ci sono query da eseguire in blocco e, nel caso, le recupera e si occupa di effettuare le operazioni richieste. Dopodiché effettua un clear del blocco di query eseguite.

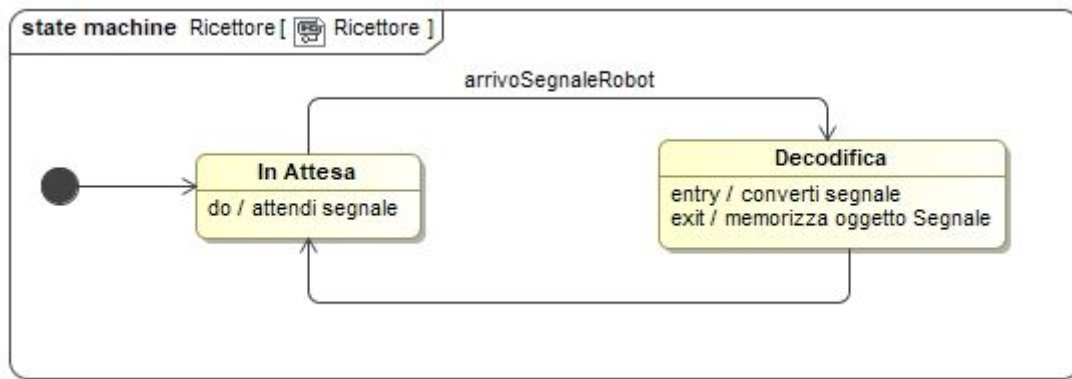
Un ultimo thread (IR Handler) si occupa invece di effettuare appena possibile il calcolo del dato IR che sarà inviato alle dashboard. Per prima cosa tramite Storage comunica al Flusher se c'è un nuovo blocco da eseguire, poi recupera

Il team ha deciso di rappresentare il percorso di un ipotetico segnale per evidenziare al meglio la sequenza di operazioni che avvengono all'interno del sistema all'arrivo del suddetto. Per evitare un sequence diagram troppo complesso la parte di Analisi è stata separatamente documentata tramite la state machine in figura C.2.4.

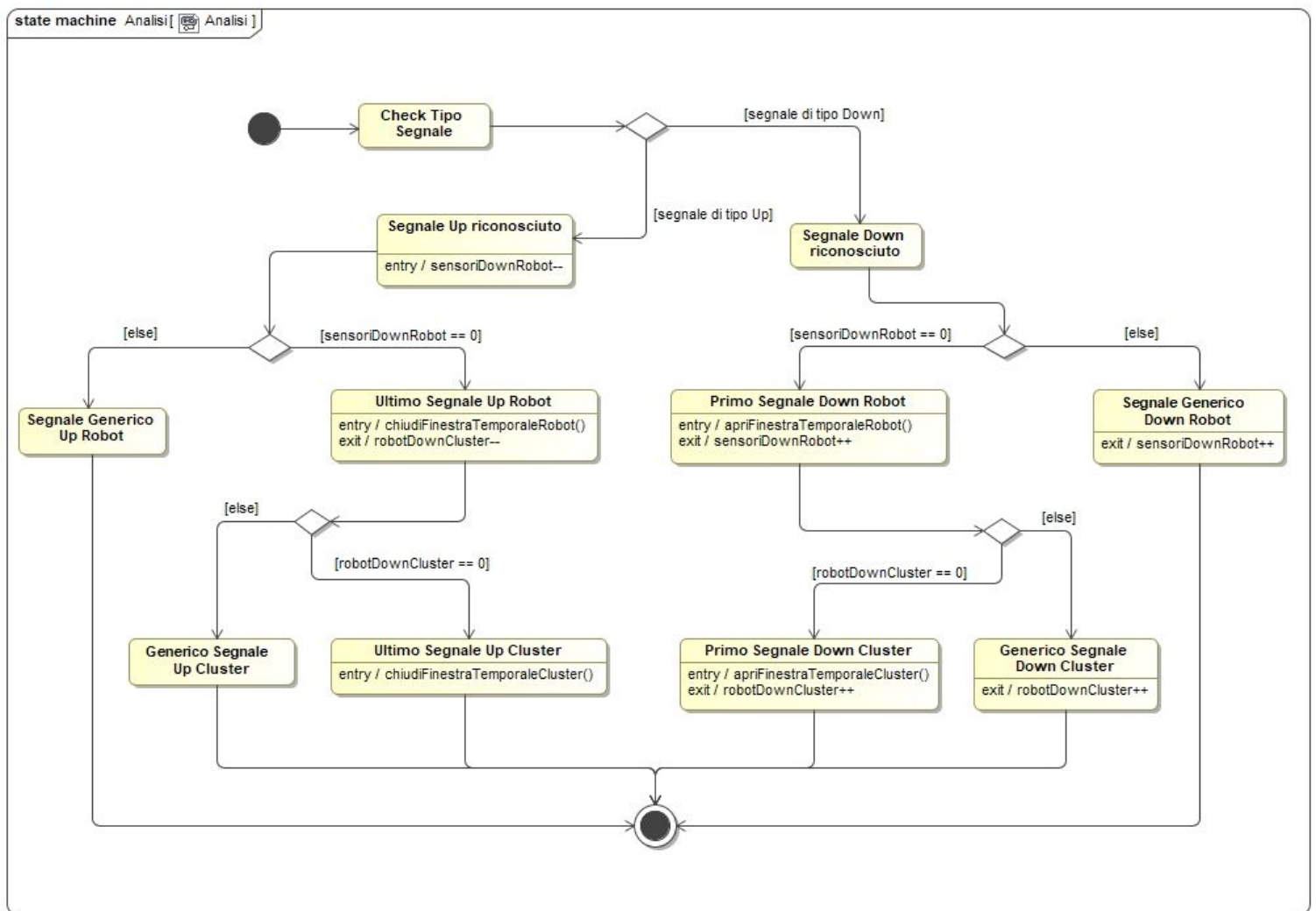
C.3 Overall State Machines



C.2.2 State Machine del Gestore Segnali



C.2.3 State Machine del Ricettore



C.2.4 State Machine dell'operazione di Analisi (di Gestore Segnali)

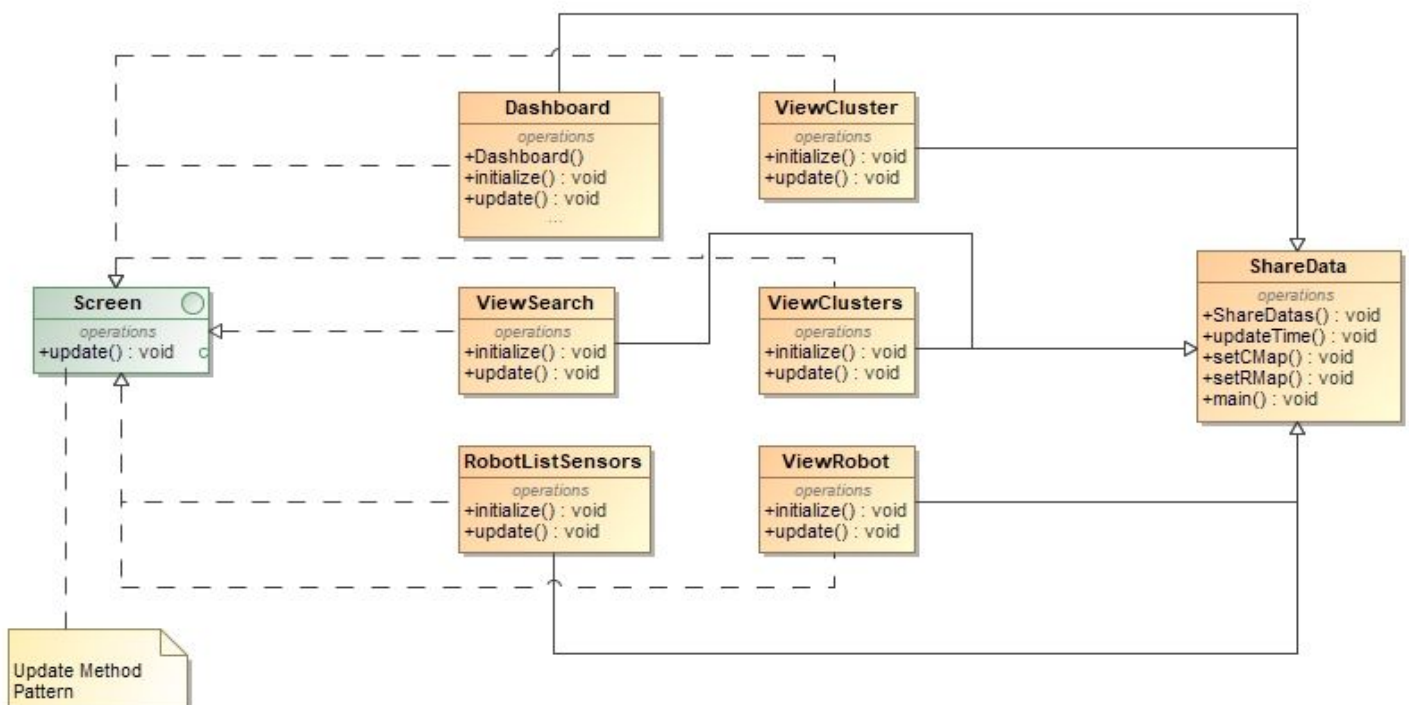
D. ER Design

Il Database è di tipo relazionale e implementato tramite tecnologia MySQL. Essendo costituito di un'unica tabella abbiamo giudicato lo schema entità-relazioni una notazione troppo pesante. Segue invece il codice SQL utilizzato in fase di creazione della tabella **finestra_temporale**:

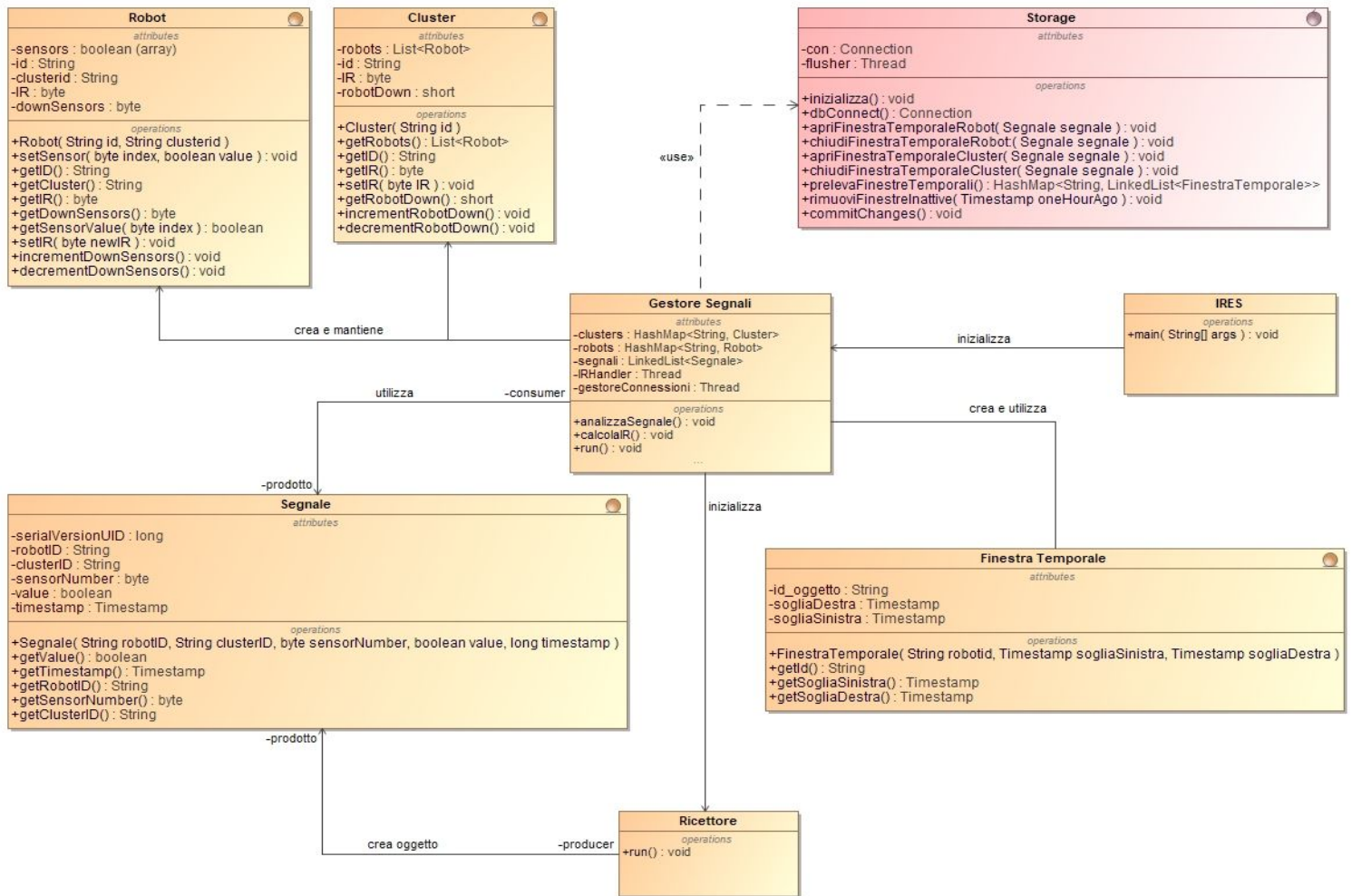
```
create table finestra_temporale(  
id int primary key auto_increment,  
id_oggetto varchar(7) default 'errore',  
sogliaSinistra timestamp,  
sogliaDestra timestamp);
```

La tabella raccoglie quindi tutte le finestre temporali create in fase di analisi dei segnali, sia per i cluster che per i robot. (L'associazione finestra-robot/cluster viene fatta dal sistema in fase di lettura dell'output restituito dalle query, come si può vedere dal codice della classe Storage)

E. Class Diagram of the Implemented System



E.1 Class Diagram della Dashboard



E.2 Class Diagram del Sistema IRES

F. Design Decisions

1. La design decision più importante è stata quella di adottare il parallelismo per ogni componente del nostro sistema. Inizialmente l'applicazione era stata progettata in tal senso ma, a seguito di incontri con i committenti, si era deciso di percorrere la strada opposta per due principali motivi: da una parte, strutturare il sistema, in particolare le operazioni di analisi dei segnali e di calcolo dell'IR, in maniera asincrona sarebbe potuto rivelarsi eccessivamente complesso; dall'altra c'era il sospetto che tale modifica non fosse affatto necessaria per garantire il soddisfacimento dei NFR del progetto. Realizzato il prototipo, che quindi eseguiva ogni operazione linearmente, ci si è ben presto resi conto che anche soltanto con una dashboard si verificava consistentemente un ritardo incrementale sull'analisi dei segnali. Questo voleva dire che il dato presentato sulla dashboard al momento X non era l'effettiva

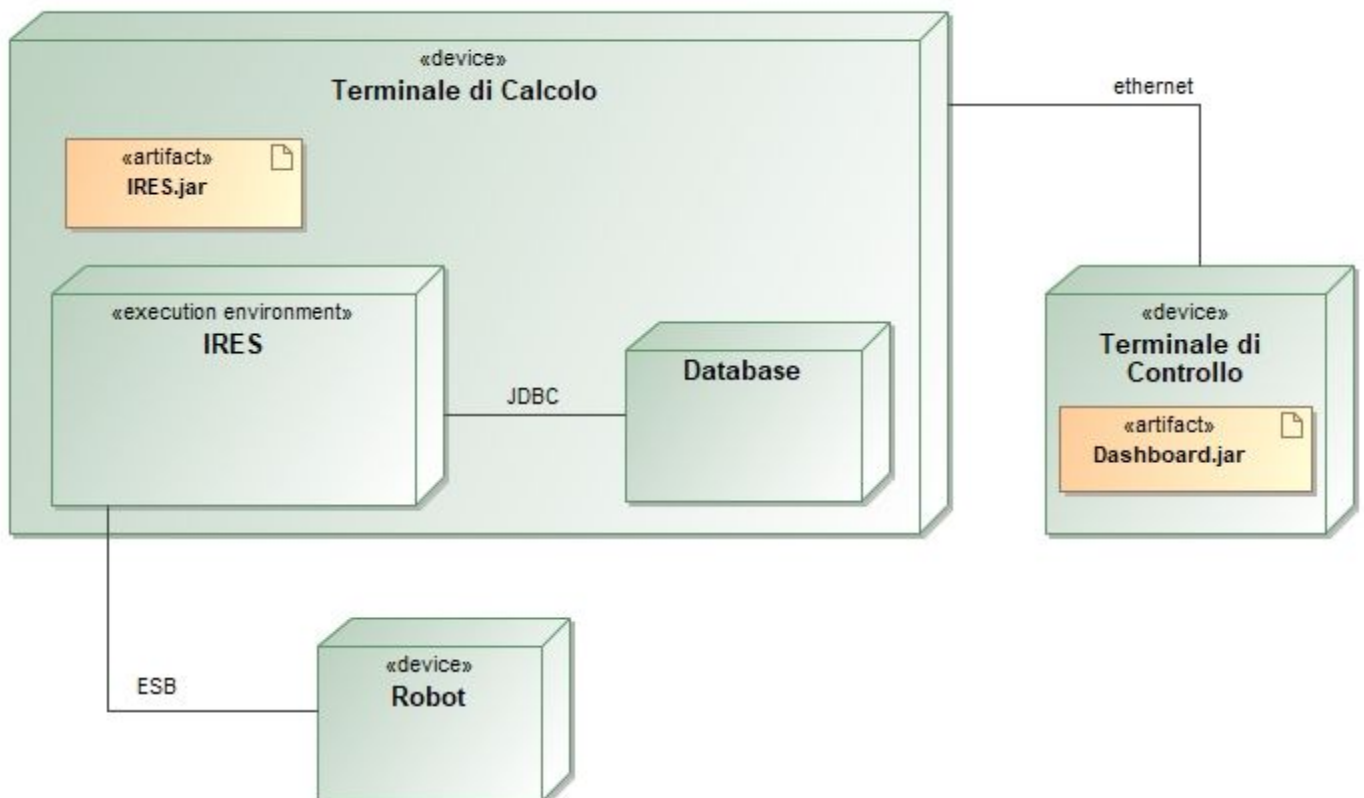
situazione di Robot e Cluster al momento X ma quella risultante dalla computazione dei segnali che il sistema era riuscito a calcolare fino al momento X (cioè un dato sfasato temporalmente). In quel momento il flusso del codice era uno e si passava per ogni fase: analisi, calcolo, query DB e così via in loop. Pertanto il team ha compiuto lo sforzo di introdurre parallelismo tra ognuno di questi tre aspetti del sistema, idea culminata nelle macro-modifiche esposte nella seguente tabella:

| | Sistema Sequenziale | Sistema Parallelo |
|--|---|---|
| Analisi dei segnali | L'analisi dei segnali aggiorna le strutture dati di Cluster e Robot e popola i batch che flusha al DB ogni 1000 query. | In maniera continua e indipendente, un thread apposito analizza i segnali e aggiorna le strutture dati di Cluster e Robot, popolando i batch. |
| Calcolo dell'IR | Ogni X secondi, lo stesso flusso di codice dell'analisi si interrompe per eseguire eventuali query rimanenti sul db, recuperare le finestre temporali, calcolare l'IR e inviarlo alle dashboard. | In maniera continua e indipendente, un thread apposito segnala a un ulteriore thread di effettuare le query sul DB e dopo aver recuperato le finestre temporali calcola l'IR e lo manda alle dashboard. |
| Interazione con il DB | Lo stesso flusso di codice (e quindi l'intero sistema) si interrompe per aspettare il DB a ogni query. | Un unico thread, il Flusher, comunica effettivamente con il DB. In questo modo soltanto una parte del sistema si mette in attesa. |
| Risultati dei test: NB: test effettuati con una sola dashboard in ascolto e analizzando i segnali provenienti da un tester progettato per mettere sotto sforzo il sistema (90.000+ segnali al minuto costanti). | <u>Tempo di aggiornamento:</u> X secondi tra un calcolo e l'altro dell'IR, più 5-10 secondi di esecuzione effettiva della procedura. N.B: il dato risulta falsato a causa dell'accumulo di segnali non ancora analizzati al momento del calcolo. | <u>Tempo di aggiornamento:</u> procedura in costante esecuzione (nessun gap fra le chiamate) la cui durata varia dai 2 ai 3 secondi. N.B: nessun segnale rimane da analizzare nella coda al momento del calcolo. |

2. Abbiamo deciso di sviluppare il sistema in due applicazioni:

- **IRES**, Inefficiency Rate EStimator, da eseguire su una sola macchina apposita, che calcola il dato che sarà poi mostrato agli ingegneri sui loro terminali: esso sarà suddiviso a sua volta in più componenti:
 - **Ricettore**, thread di **GestoreSegnali** che si occupa di ricevere i segnali, decodificarli e inserirli in una struttura dati;
 - **Storage**, classe che funge da interfaccia tra IRES e DB; contiene al suo interno il thread **Flusher**, che si occupa di eseguire e mantenere i batch di query del DB;
 - **IRHandler**, thread di **GestoreSegnali** che si occupa di calcolare l'IR di Cluster e Robot e inviarlo alle Dashboard;
 - **GestoreConnessioni**, thread listener di **GestoreSegnali** che stabilisce un canale di comunicazione con le dashboard che si connettono;
- **Dashboard**, da eseguire su ogni terminale degli ingegneri, che riceverà il dato via connessione Ethernet e lo mostrerà in un'interfaccia grafica adeguata.

La scelta è documentata nel diagramma seguente (figura F.1).



F.1 Deployment Diagram del Sistema

3. Abbiamo deciso di elevare il concetto di "Finestra Temporale di Down Time" a entità del sistema. Queste finestre vengono create per periodi di down sia per i Cluster che per i Robot. In particolare, quelle relative ai Robot vengono istanziate con il solo timestamp di inizio ("*timestamp sinistro*") nel momento in cui va down per la prima volta un sensore; quelle dei cluster seguono la stessa logica ma per i robot contenuti al loro interno. Queste finestre verranno modellate come oggetti e immagazzinate nel DB al fine di calcolare in maniera corretta e strutturata il tempo di Down di ogni Robot, e quindi il suo Tasso di Inefficienza.
4. Inizialmente venivano effettuati, per ogni dashboard connessa, due invii sequenziali sui socket: uno per la struttura dati dei cluster e l'altro per i robot, affidando la serializzazione degli oggetti alla libreria nativa di Java. Data la lentezza e la poca compatibilità della soluzione, il team ha invece adottato il formato JSON anche per la comunicazione con le dashboard, tramite l'implementazione realizzata da Google: Gson. E' stata quindi predisposta una classe apposita in cui wrappare le due strutture dati da convertire poi in formato JSON. Dai test effettuati è stato notato un guadagno di 1-2 secondi nel tempo totale di ricezione del dato.
5. Al fine di snellire i tempi di update, i dati relativi ai robot e ai cluster saranno tenuti esclusivamente in memoria RAM, così da non doverli recuperare ogni volta che le Dashboard si aggiornano. Le FinestreTemporali attive invece, per motivi di scalabilità, saranno tenute esclusivamente nel DB e portate in RAM tutte insieme quando richieste per calcolare l'IR; terminata la procedura di calcolo, verranno eliminate dalla memoria. Dati utilizzo memoria dai test fatti: Database, circa 2-3MB per 5 minuti di segnali, cioè massimo 35-40MB per un'ora di attività, oltre la quale comunque le finestre inattive vengono eliminate; Dashboard, non calcolata perché irrilevante; IRES, si attesta su una media di 1GB di spazio in RAM costante per l'esecuzione continua a pieno regime.

G. FRs & NFRs Solution

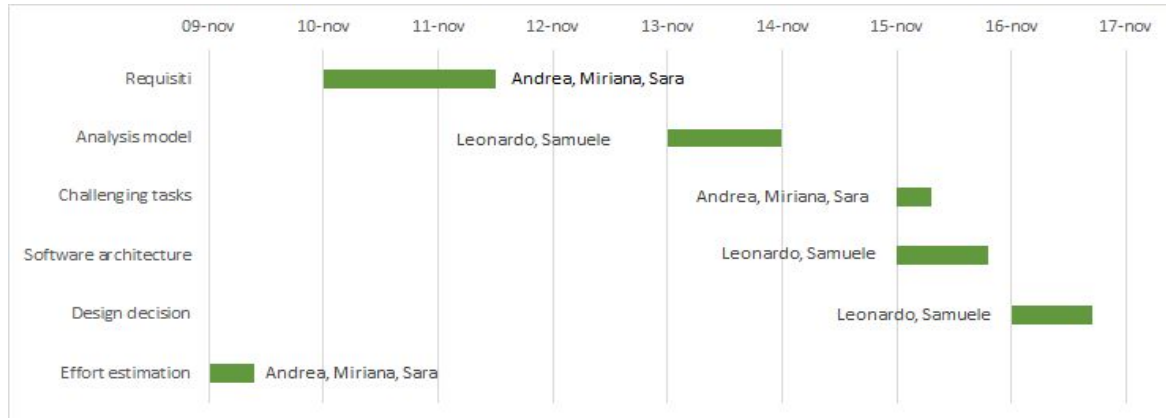
Mapping FR / NFR a Design Decision (DD)

- I FR 1 e 2 sono stati soddisfatti dall'implementazione dell'algoritmo operante sulle finestre temporali contenute nel DB.

- I FR dal 3 al 9 sono stati soddisfatti dall'inserimento di apposite sezioni/funzionalità nella Dashboard.
- Il FR 10 è garantito dai numerosi test di performance effettuati durante tutto il periodo di sviluppo nei quali la correttezza del dato è sempre stata mantenuta.
- Il FR 11 è soddisfatto dall'utilizzo di un DB dove i segnali dei Robot vengono immagazzinati in maniera strutturata.
- Il NFR 1 è soddisfatto dalle DDs 1, 3 e 5, cioè grazie a performance e persistenza.
- Il NFR 4 è soddisfatto dall'algoritmo di analisi dei segnali implementato che automaticamente registra nuovi Robot, e di conseguenza Cluster, al loro primo segnale inviato.
- Il NFR 6 è soddisfatto dai numerosi test effettuati sul prototipo finale (quindi quello strutturato parallelamente) nei quali il numero di segnali rimanenti da analizzare nella relativa coda al momento dell'inizio del calcolo dell'IR era sempre 0.
- Il NFR 7 è soddisfatto dall'associazione tanto in IRES quanto nella Dashboard di un codice del formato *RXXXXX* per i Robot e *CXXX* per i Cluster, implementato tramite String Java.
- I NFRs 2, 3 e 5 meritano un discorso a parte:
 Grazie all'impiego di una struttura asincrona e al formato Gson per lo scambio dei dati, il prototipo consegnato è in grado di fornire un aggiornamento del dato ogni pochi secondi in presenza di una singola dashboard e con un'affluenza esagerata di segnali dal nostro Tester. Tale performance non viene mantenuta se il numero di dashboard cresce, andando da un refresh rate rallentato a una decina di secondi con 3 dashboard fino ai quasi 2 minuti del test effettuato simulando 100 dashboard con dei thread a cui veniva inviato il dato. La bontà dell'aspetto computazionale del sistema, con particolare riguardo alla sua efficienza, è confermata dal fatto che il 95% del tempo è speso nella fase di invio, prototipata tramite socket Java ma che andrebbe, nella release finale del sistema, sostituita con un approccio producer-consumer basato sul protocollo HTTP come da specifica. Anziché serializzare e inviare il JSON su ogni socket, IRES potrebbe scrivere la corrispondente stringa su file che poi le dashboard recuperano tramite richiesta GET (eventualmente automatizzata con Web Socket). Questa strategia risulta altamente scalabile perché di fatto indipendente dal numero di dashboard connesse e performante poiché permette al sistema di consegnare un nuovo JSON, quindi una nuova versione dei dati, circa ogni secondo. (Questo soddisfa appieno il NFR 5. Un aumento anche ingente di Robot causerebbe un aumento nel tempo di POST del file JSON di qualche secondo).

H. Effort Recording

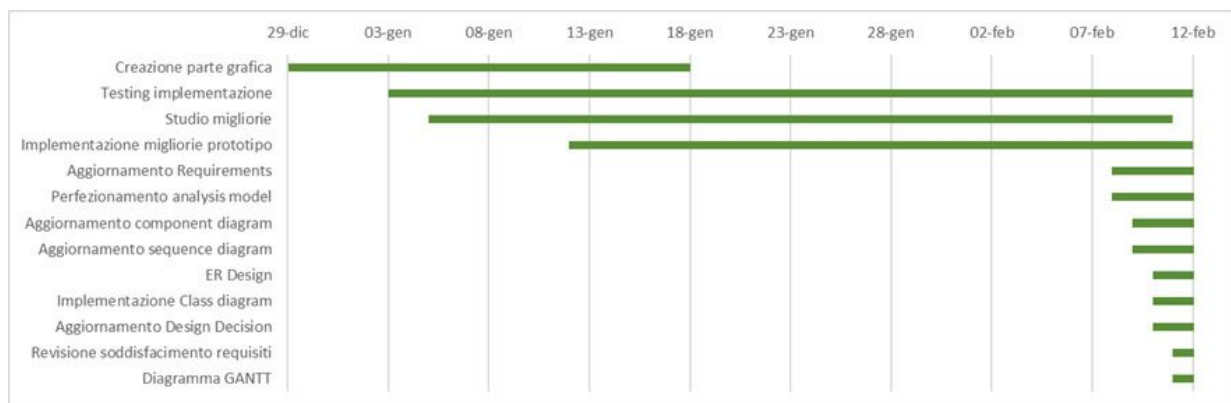
H.1 Gantt Charts



H.1.1 Carta di Gantt del Deliverable 1



H.1.2 Carta di Gantt del Deliverable 2



H.1.3 Carta di Gantt del Deliverable 3

Ore Totali D3: 50 ore su 5 persone

Appendix. Code Prototype

La dimostrazione più realistica del prototipo andrebbe effettuata con due computer, come da Deployment Diagram, uno che esegue IRES (con annesso Tester) e l'altro una Dashboard, connessi tramite Ethernet. Tuttavia, per comodità, includiamo le istruzioni anche per il caso di esecuzione su singola macchina.

Istruzioni per l'esecuzione del prototipo su due macchine (A - IRES; B - Dashboard):

- 1) Assicurarsi di avere un server MySQL attivo locale sulla macchina A in cui non sia presente un database di nome "dashboard";
- 2) Non avendo un dispositivo che automatizza il processo di riconoscimento dei computer connessi via Ethernet (come un modem, ad esempio), è necessario settare un indirizzo IP fisso che la macchina A adotterà così da essere riconoscibile dalla macchina B. Perciò nella macchina A, nelle proprietà della scheda di rete Ethernet (raggiungibile su Windows tramite pannello di controllo), alla voce "*Protocollo Internet versione 4 (TCP/IPv4)*" selezionare nuovamente *Proprietà* e nel box che appare selezionare "*Usa il seguente indirizzo IP:*". Digitare un indirizzo, ad esempio 192.168.0.1, il subnet mask dovrebbe apparire da solo altrimenti inserire 255.255.255.0 e lasciare vuoto lo spazio per i DNS in basso. Cliccare su OK per concludere la procedura;
- 3) Aprire due shell sulla macchina A da impostare sulla directory dei due file jar, *ires.jar* e *tester.jar*;
- 4) Nella prima shell della macchina A, digitare "*java -jar tester.jar <iterazioni> <segnali_per_iterazione>*" dove *iterazioni* è il numero di volte che verrà eseguito un loop all'interno del quale viene effettuato l'invio di *segnali_per_iterazione* segnali. Confermando il comando il Tester si mette in attesa di connettersi con IRES per inviare i segnali;
- 5) Aprire una shell sulla macchina B, navigare alla directory della dashboard e digitare il comando "*java -jar dashboard.jar <ires_address>*" dove *ires_address* rappresenta l'indirizzo IP su connessione Ethernet della macchina A impostato precedentemente;
- 6) Nella seconda shell della macchina A, digitare "*java -jar ires.jar <user_db> <pass_db>*" con le credenziali di accesso al server MySQL, e confermare il comando;
- 7) Dopo qualche istante, confermare il comando della dashboard sulla macchina B.

Istruzioni per l'esecuzione del prototipo su singola macchina:

- 1) Assicurarsi di avere un server MySQL attivo locale in cui non sia presente un database di nome "dashboard";
- 2) Aprire tre shell da impostare sulla directory dei tre file jar: *ires.jar*, *dashboard.jar* e *tester.jar*;
- 3) Nella prima shell digitare `"java -jar tester.jar <iterazioni> <segnali_per_iterazione>"` dove *iterazioni* è il numero di volte che verrà eseguito un loop all'interno del quale viene effettuato l'invio di *segnali_per_iterazione* segnali. Confermando il comando il Tester si mette in attesa di connettersi con IRES per inviare i segnali;
- 4) Nella seconda shell digitare il comando `"java -jar dashboard.jar <ires_address>"` dove *ires_address* può tranquillamente essere "localhost";
- 5) Nella terza shell digitare `"java -jar ires.jar <user_db> <pass_db>"` con le credenziali di accesso al server MySQL, e confermare il comando;
- 6) Dopo qualche istante, confermare il comando di esecuzione della dashboard.

Documentazione codice:

APX 1 Rif. codice del thread IRHandler, file GestoreSegnali.java, dalla riga 46 alla 83;

Commento:

Thread inizializzato da GestoreSegnali il cui scopo è calcolare l'IR e inviarlo alle Dashboard. Nel passo preliminare al ciclo infinito viene stabilita una connessione al Database; fondamentale per il corretto interfacciamento sistema-db è infatti l'utilizzo da parte di ogni thread di una connessione esclusiva con il database. I metodi chiamati nel ciclo, `commitChanges()` e `calcolaIR()`, sono gli stessi che prima venivano eseguiti in maniera sequenziale nel vecchio prototipo. I cambiamenti al loro codice sono dettagliati in sezioni successive della documentazione. Importante notare come all'inizio del ciclo venga controllata una variabile booleana, gestita dal Flusher, simboleggiante il possibile stato down del DB; in tal caso, IRHandler si metterà in attesa e ogni 5 secondi ripeterà il controllo.

APX 2 Rif. codice del metodo `calcolaIR()`, file GestoreSegnali.java, dalla riga 193 alla 210;

Commento:

ClientWrapper è una classe wrapper di comodo che racchiude al suo interno un Socket e un ObjectOutputStream aperto su tale socket; Data è una classe che funge da “pacchetto dati” da trasformare in formato JSON e serializzare sul socket al momento dell’invio alle dashboard.

Sostanzialmente si scorrono le dashboard connesse, registrate in precedenza dall’apposito thread, e si invia a ciascuna di loro un oggetto Data in formato JSON contenente le strutture dati di Cluster e Robot. Poiché il processo di serializzazione riscontra problemi di concorrenza con quello di modifica delle strutture dati coinvolte, l’operazione di invio è racchiusa in due costrutti java synchronized, che fungono da sezione critica e che sono presenti anche nel codice di analisi dei segnali, in modo da rendere invio e analisi mutuamente esclusive. Qualora una dashboard si disconnettesse, decretando quindi il fallimento dell’invio, la stessa viene semplicemente rimossa dalla relativa struttura dati.

APX 3 Rif. codice del thread che gestisce le connessioni con le dashboard, file GestoreSegnali.java, dalla riga 222 alla 235.

Commento:

Semplice procedura di check-in: il thread rimane in ascolto per eventuali connessioni e quando ne arriva una genera un oggetto ClientWrapper che va a popolare la struttura dati poi utilizzata da IRHandler.

APX 4 Rif. codice del thread Flusher, file Storage.java, dalla riga 37 alla 116.

Commento:

Dopo aver stabilito una connessione personale con il DB, il Flusher si occupa di eseguire le query che creano il database e la singola tabella necessaria allo storage dei dati. Vengono poi inizializzati i batch e le variabili booleane che fungono da flag per effettuare il flush. Il funzionamento della procedura di flush è questo: a ogni indice delle liste di batch corrisponde una variabile booleana, di default settata a false, che eventualmente setterà a true IRHandler quando verrà il momento di eseguire le query sul DB per avere i dati pronti da reperire e analizzare per il calcolo. Non appena ciò accade, il Flusher esegue i batch attivi e li resetta. Le liste sono circolari, nel senso che sono composte da 10 batch ciascuna che vengono via via resettati.

Ricapitolando:

L’analisi dei segnali popola i batch attivi di query che vengono inviate al DB dal Flusher su segnalazione di IRHandler, il quale inoltre rende attivi i prossimi batch in lista; conclusa la procedura i batch flushati vengono resettati e si aspetta di flushare i nuovi.

APX 5 Rif. codice del metodo `apriFinestraTemporaleRobot`, a titolo di esempio anche per le altre 3 operazioni sulle finestre temporali; file `Storage.java`, dalla riga 132 alla 142.

Commento:

Essendo molto simili tra di loro, riportiamo solo una delle 4 operazioni sulle finestre temporali a titolo di esempio. Viene salvato innanzitutto il valore dell'indice indicante i batch attivi di modo che non cambi tra una chiamata e l'altra ai metodi che operano sui batch. Dopodiché si inseriscono come parametri della query i dati provenienti dal Segnale analizzato.

APX 6 Rif. codice del metodo `commitChanges()`, file `Storage.java`, dalla riga 160 alla 163.

Commento:

Viene segnalato il flush ai batch correntemente attivi e si rendono attivi quelli successivi in lista tramite lo spostamento circolare dell'indice.

Differences between D3 and D2

Here we describe the major changes between the decisions taken in the D3 and those taken in the D2

- 1) Sicuramente la differenza più grande, da cui sono scaturite profonde modifiche all'intero impianto strutturale del progetto, risiede nel passaggio da sistema sequenziale a parallelo. E' stata una scelta necessaria, causata dal non soddisfacimento dei requisiti di correttezza del dato e di performance, che ha poi avuto ripercussioni su quasi ogni parte del sistema, diventata ora asincrona. Numerose parti della documentazione sono state riviste, aggiornate o corrette per riflettere il cambiamento.
- 2) Ricollegandosi al punto 1, per il D3 c'è stato un cambio di approccio rispetto al D2. Lo sviluppo è stato dettato molto di più dal prototipo e dai risultati ottenuti dai test preliminari, rispetto ai primi deliverable dove si era fatto un lavoro più che altro teorico per inquadrare il contesto (D1) e iniziare a strutturare il sistema con annesso algoritmo di calcolo dell'Inefficiency Rate (D2). C'è stata pertanto una convergenza dell'architettura messa in piedi inizialmente, le cui componenti essenziali sono rimaste di fatto inalterate (algoritmo di calcolo, algoritmo di analisi..), con la nuova struttura parallela e la conseguente divisione delle operazioni fra thread indipendenti.