

“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	<27/01/2018>
Deliverable	<Deliverable 3 - Finale>
Team (Name)	Unnamed Group

Team Members		
Name & Surname	Matriculation Number	E-mail address
Santi Filippo	232028	filippo.santi@student.univaq.it
De Ciantis Stefano	242860	stefano.deciantis@student.univaq.it
Ubaldi Davide	250668	davide.ubaldi@outlook.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*

Table of Contents of this deliverable

PAGINA	CONTENUTO
4	List of Challenging/Risky Requirements or Tasks
5	Requirements Collection
6	FR
7	Use Cases
8 – 9 – 10 - 11	GUI Requirements
12	Business Logic – DB Requirements
13	NFR
14	Content- Assumptions
15	Prioritization
16	Analysis Model – Robustness diagram
17	Robustness Diagram description
18	Software Architecture – Component Diagram
19	Component Diagram Description
20 – 21 – 22 - 23	Sequence Diagrams and relative descriptions
24 - 25	E R Design
26	Class Diagram of the implemented System
27 – 28 - 29	Class Diagram description
30-31	Design Decision
32 - 33	Explain how the FRs and NFRs are satisfied by design
34	Effort Recording
35 – 36 – 37 – 38 - 39	Appendix. Code

List of Challenging/Risky Requirements or Tasks

<In this section, you should describe using the table below the most challenging or discussed or risky design tasks, requirements, or activities related to this project. Please describe when the risk arised, when and how it has been solved.>

PLEASE FILL IN THIS TABLE AT EACH DELIVERABLE

Challenging Task	Date the task is identified	Date the challenge is resolved	Explanation on how the challenge has been managed
MongoDB - Insert Many	03/01/2018	12/01/2018	Approfondimento riguardante la funzione insert many su mongodb. Questa funzione risultava molto performante sulla carta inizialmente ma si sono rilevati alcuni problemi che influivano pesantemente sulle performance nella parte riguardante l'update dei robot. Grazie a un lavoro di documentazione online e un minuzioso lavoro di debug sul codice prodotto si è riuscito a isolare e risolvere completamente il problema ottenendo performance come da aspettative.
Java FX, .fxml file, editor Scene Builder	4/01/2018	5/01/2018	Installare in maniera funzionale il tool SceneBuilder, studio della documentazione offerta direttamente da Oracle (https://docs.oracle.com) e visione di tutorial online sulla funzione l'utilizzo e la gestione dei file FXML prodotti.
InputSimulation e update random dei segnali	12/01/2018	14/01/2018	Implementazione della nuova logica di generazione dei segnali attraverso la creazione di parametri utente.
Calcolo percentuale IR	15/01/2018	17/01/2018	Ricercare il giusto algoritmo applicabile sui dati, costantemente aggiornati, presenti sul database. Si è utilizzato il supporto di documentazioni online per utilizzare e gestire le liste ed estrapolarne soltanto i dati necessari al calcolo.

A. Requirements Collection

*In this section, you should describe both the application **features/functional** requirements as well as the **non functional** ones. You shall also document **constraints** and **rules**, if they apply.*

Che cosa deve fare il sistema?

Il sistema riceve in input una serie di segnali inviati da sensori presenti sui robot che si occupano del monitoraggio delle componenti hardware del robot stesso, i robot sono raggruppati in cluster che a loro volta sono raggruppati in aree, i robot che appartengono allo stesso cluster svolgono le stesse operazioni avendo il medesimo hardware. Ogni robot, cluster o area viene identificato univocamente attraverso un codice alfanumerico. Una volta memorizzati il sistema si occuperà di processare i dati calcolando l’Inefficiency Rate. Il sistema fornirà un’interfaccia (dashboard) che visualizza l’Inefficiency Rate di un cluster o di un singolo robot.

A.1 Functional Requirements

*<List, in bullet points, all the functional requirements your system shall implement>
<Describe functional requirements and stakeholders through Use Case Diagrams>.
<Then, prioritize them, and provide a table-based description of the most important requirements>
<Provide a table-based description of the most important requirements, using the Alistair Cockburn Use Case Template>*

PLEASE COPY HERE: (i) ALWAYS the diagram to be discussed, (ii) the text explaining the DECISIONS taken when creating the diagram (that is, do not spend time in EXPLAINING what the diagram says, but more on the decisions taken to create the diagram).

PRIORITIZE THEM

FOR EACH DIAGRAM (Use Case D, Analysis Model, Class, Sequence) add a number and a label to it (e.g., Figure 1: Sequence diagram of the xxx scenario)

<i>Functional Requirement</i>	Description
<i>Invio segnali</i>	Un robot dispone di una serie di sensori che inviano un flusso periodico di dati al sistema sotto forma di segnali, indicando se, in quel momento, per il robot è possibile eseguire una certa operazione o meno.
<i>Analisi real-time</i>	Il sistema effettua l'analisi real-time dei cambiamenti di stato trasmessi dai robot. L'analisi real-time prenderà decisioni automatizzate sullo stato operativo dei robot, che può essere sostanzialmente riassunto in: produzione o manutenzione, a seconda dello stato di UP o DOWN dell'hardware relativo ai sensori.
<i>Calcolo IR</i>	Il sistema effettua l'analisi real-time dei cambiamenti di stato trasmessi dai robot. L'analisi real-time prenderà decisioni automatizzate sullo stato operativo dei robot, che può essere sostanzialmente riassunto in: produzione o manutenzione, a seconda dello stato di UP o DOWN dell'hardware relativo ai sensori.
<i>Visualizzazione dati dashboard</i>	L'utente, attraverso la dashboard, deve essere in grado di visualizzare l'inefficiency rate di ogni robot dai 3 punti di vista: delle aree in generale, dal cluster in generale, oppure di un robot nello specifico. La dashboard permetterà all'utente di selezionare il cluster o l'area di cui si vogliono visualizzare le informazioni. La dashboard mostrerà la percentuale di Inefficiency Rate per ogni area, per ogni cluster e per ogni robot con un colore diverso (rosso se Inefficiency Rate > 0 = al valore es:20%, verde altrimenti).
<i>Data Storage</i>	Il cambio di stato in input proveniente da ogni singolo robot viene memorizzato nel database.

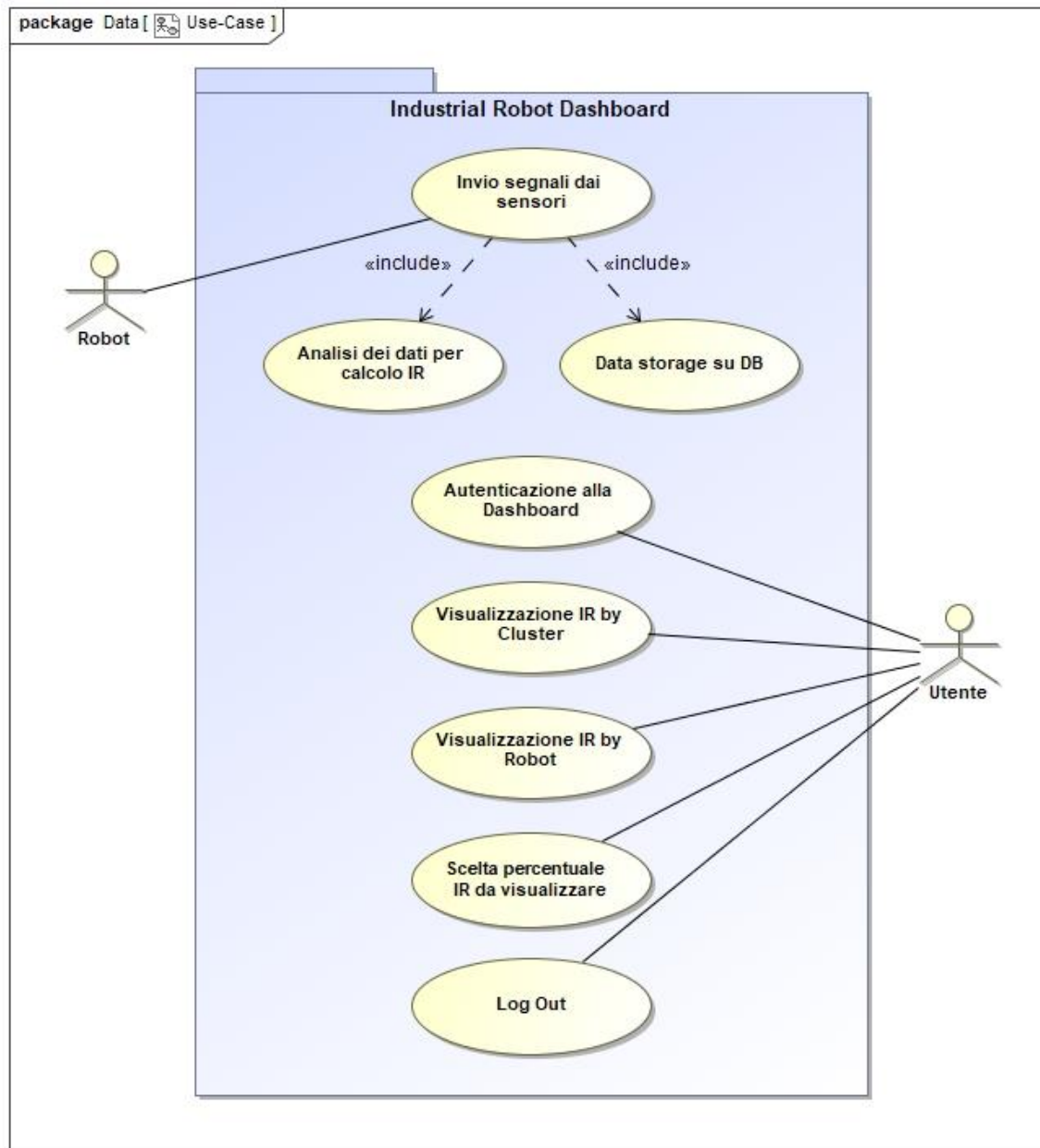


Figura 1- Use Case

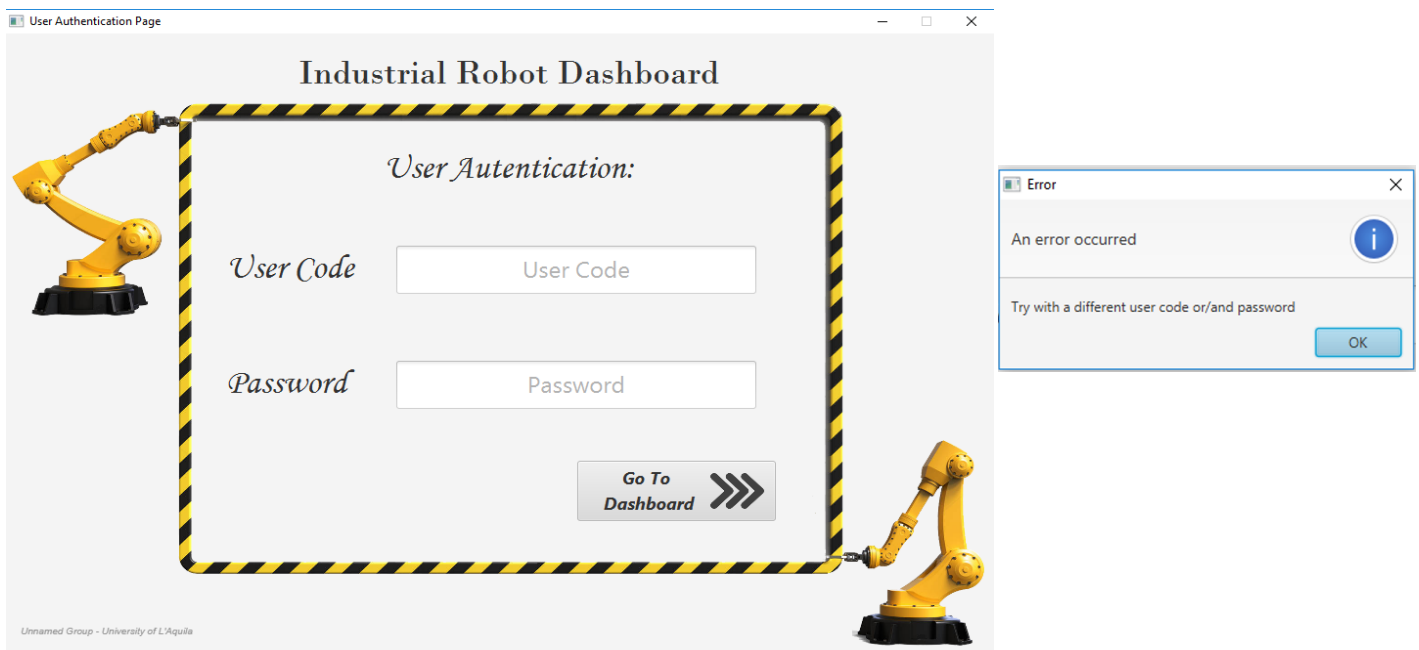
Abbiamo deciso di rappresentare “Analisi dei dati per il calcolo IR” e “Data Storage su DB” tramite la relazione `<<include>>` in quanto ci consente di completare lo use case precedente aggiungendo le logiche del calcolo dell’IR e dello storage dei dati al fine di fornire una panoramica completa su quello che è il ciclo dei segnali, dall’input alla memorizzazione.

Abbiamo, inoltre, deciso di rappresentare nel dettaglio tutte le azioni che un utente svolge o può svolgere in una sessione di lavoro standard, infatti l’ingegnere deve per prima cosa

autenticarsi con successo per accedere ai dati di monitoraggio della dashboard, quando esso ha ottenuto l'accesso può decidere di accedere alle varie visualizzazioni dell'IR e ha la possibilità di usufruire di tutte le funzionalità della dashboard. Una volta svolto il lavoro può effettuare il log out e permettere ad altri utenti di usufruire di quel terminale.

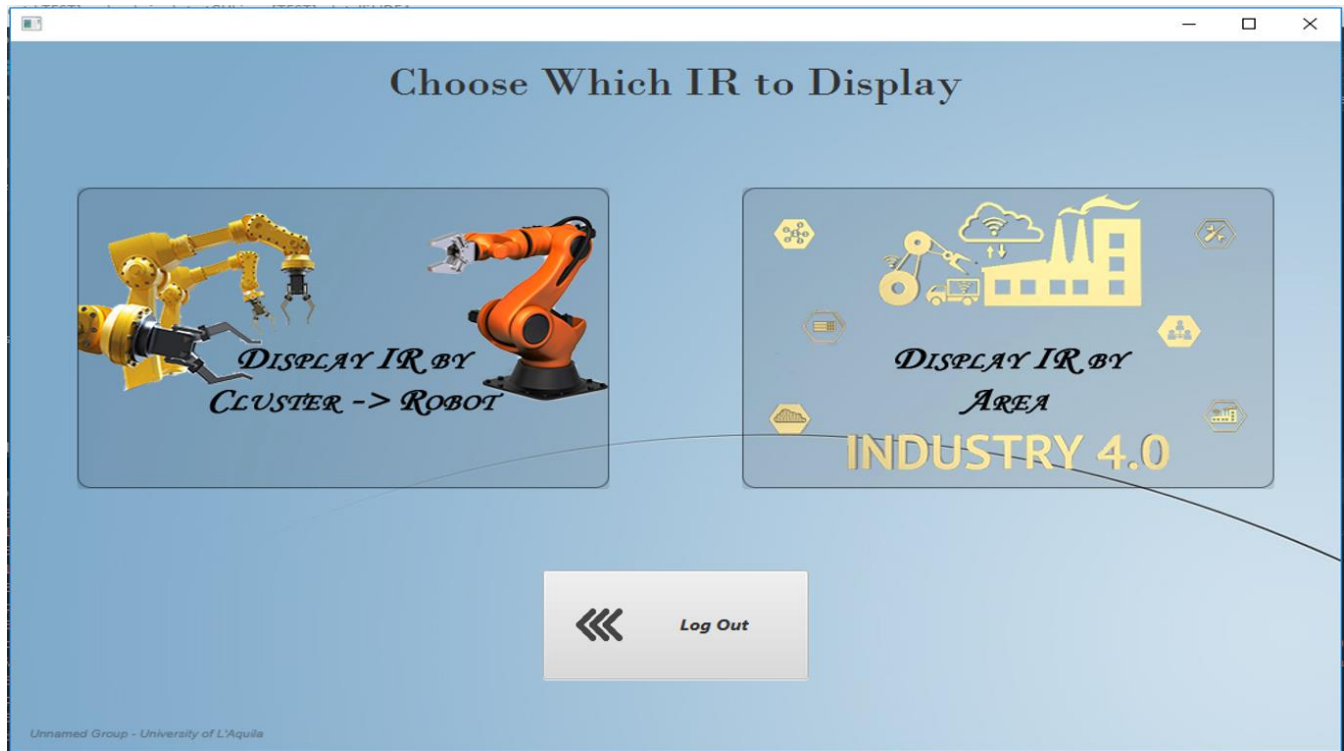
A1.1 GUI Requirements (da riempire a partire dalla Versione 2)

La GUI rappresenta il mezzo attraverso il quale il fruitore finale, ossia l'utente, si interfaccia con il sistema, nel nostro caso la GUI è rappresentata dalla DASHBOARD attraverso la quale l'ingegnere può monitorare lo stato di produttività di ogni singolo Robot da diverse "prospettive".



A1.1.1 - Screenshot – Scenario Autenticazione + messaggio di errore

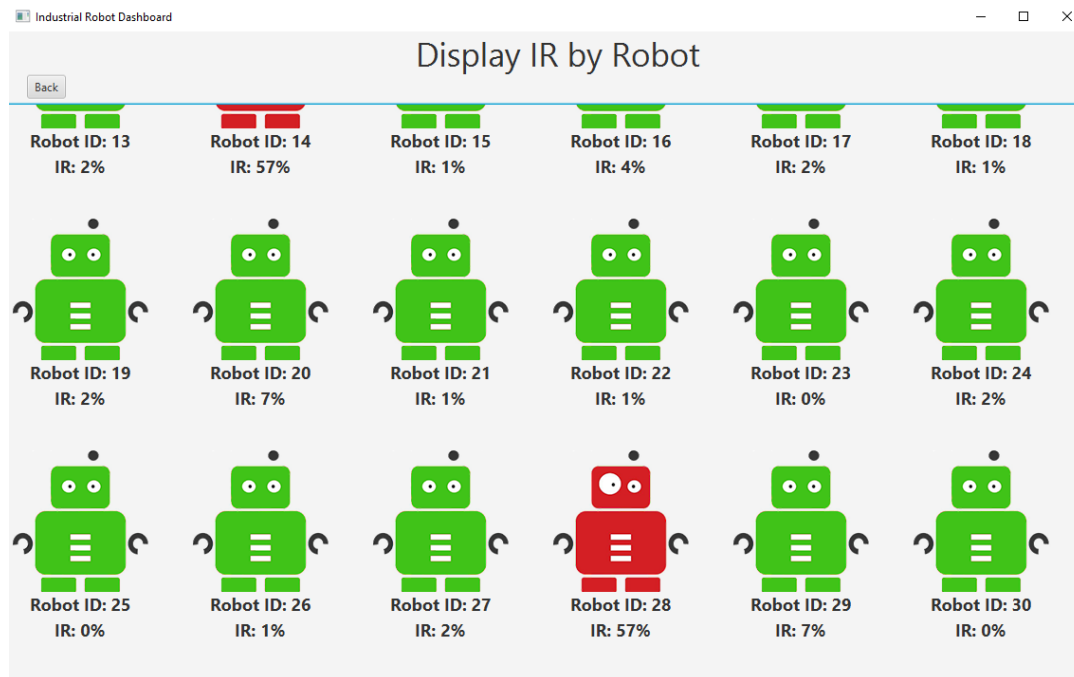
Nel momento in cui l'ingegnere decide di accedere alla Dashboard dovrà identificarsi nell'apposita finestra con i propri dati personali, solo nel caso in cui l'autenticazione avvenga con successo l'ingegnere può visualizzare i dati di monitoraggio (nel caso i dati non venano correttamente inseriti viene stampato un messaggio di errore e si dà possibilità all'utente di ritentare l'autenticazione).



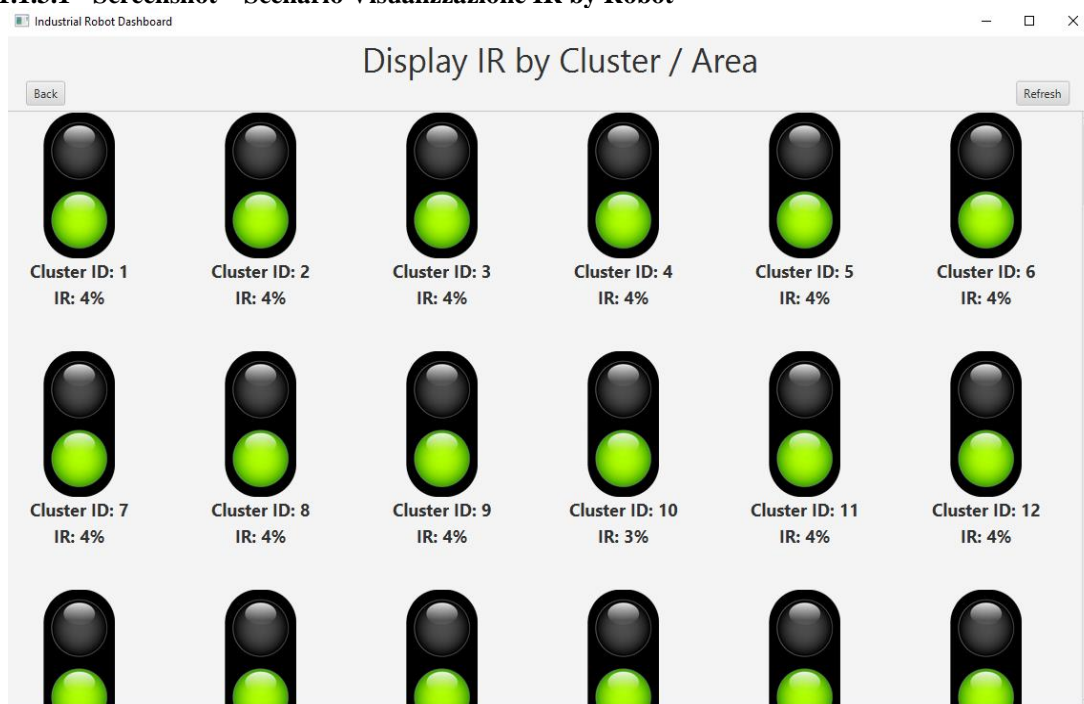
A1.1.2 - Screenshot – Scenario Scelta IR da Visualizzare

Una volta autenticato l'utente ha modo di scegliere la “prospettiva” dalla quale vuole monitorare l'efficienza dei Robot tramite degli appositi bottoni. Questi permettono la visualizzazione dell'inefficiency rate (IR) di ogni singolo Robot, di ogni singolo Cluster oppure la massima astrazione si ottiene con il monitoraggio delle Aree.

Per motivi di praticità e per facilitare la lettura dei dati all'utente finale si è scelto di raggruppare Cluster -> Robot, mentre invece le aree si possono monitorare direttamente dall'apposito bottone, in modo tale da avere una visualizzazione il più ordinata e precisa possibile (altrimenti visualizzare random circa 90'000 robot in un'unica finestra era forse più facile dal punto di vista implementativo ma poco utile allo scopo per cui la dashboard è stata realizzata) infatti scegliendo il bottone Cluster->Robot verranno visualizzati tutti i cluster con i relativi dati e cliccando su ognuno di essi sarà possibile visualizzare le informazioni di ogni robot presente all'interno di esso, mentre per visualizzare le informazioni relative alle aree basta fare click sull'altro bottone. L'utente avrà anche la possibilità di effettuare il log out per permettere a più ingegneri di utilizzare un singolo pc (es: cambio turno lavorativo).



A1.1.3.1 - Screenshot – Scenario Visualizzazione IR by Robot



A1.1.3.2 - Screenshot – Scenario Visualizzazione IR by Cluster oppure by Area

In questi scenari (A1.1.3.1 e A1.1.3.2) è possibile osservare nel dettaglio il compito che viene svolto dalla dashboard, ossia quello di monitorare e fornire dati all'ingegnere. Possiamo notare la presenza di 2 scenari differenti:

- **A1.1.3.1** è lo scenario rappresentante la visualizzazione by robot alla quale arrivo solo dopo aver navigato nei cluster, qui sono presenti i dati di monitoraggio di tutti i robot appartenenti al cluster precedentemente selezionato, sarà possibile fare click su ogni robot visualizzato per espanderne tutti i dettagli.
- **A1.1.3.2** è lo scenario rappresentante la visualizzazione by Cluster o Area, anche qui, come si può intuire, la lettura dei dati è facilitata dai colori utilizzati, rosso o verde secondo il caso, nella visualizzazione by area sarà possibile cliccare su ogni area presente in modo da espandere e controllare i cluster che la compongono, inoltre nella porzione destra sotto il titolo è posizionato il pulsante per fare il refresh della finestra e quindi rendere i dati aggiornati al preciso istante in cui se ne ha bisogno.

Sensors	Actual State
Laser - Calibrazione	UP
Piastra di Processo - Livello di Contaminazione	UP
Camera di Processo - Temperatura	DOWN
Camera di Processo - Livello di Contaminazione	UP
Camera di Processo - Pressione	UP
Injector Plasma - Pressione	UP
Injector Plasma - Temperatura	UP

Robot ID: 28
Appartenance Cluster ID: 1

* The data refers to the measurements of the last 60 minutes

A1.1.4 - Screenshot – Scenario Visualizzazione Dettagliata Singolo Robot

Per permettere all'ingegnere di apprendere quante più informazioni e dati relativi ad ogni singolo robot è stata progettata una finestra che visualizza nel dettaglio tutte le informazioni che si possono ottenere da ogni robot, ossia: lo specifico dei 7 sensori che riportano statistiche aggiuntive non visualizzabili altrimenti (es: lo stato attuale relativo ad ognuno dei 7 sensori), oltre ai dati relativi all'id del robot, al cluster di appartenenza e all'IR totale. Per accedere a questa view è necessario navigare nei cluster scegliendo quello che ci interessa facendoci click sopra, a quel punto verranno visualizzati tutti i robot che lo compongono, facendo click su uno specifico robot si visualizzerà la finestra in questione.

A1.2 Business Logic Requirements (da riempire a partire dalla Versione 2)

La logica di business presente nel nostro sistema ruota tutta intorno alla manipolazione dei dati ottenuti in input, ovvero ai cambi di stato dei robot, che verranno processati dopo essere stati estrapolati dal database. La Business Logic infatti svolge i seguenti compiti:

- Eseguire i controlli sul DB per verificare eventuali cambi di stato tra i segnali ottenuti in ingresso e nel caso sovrascriverne il valore attuale;
 - Eseguire i get() di tutti i valori dei segnali di tutti i robot per poter calcolare l'IR a 3 diversi livelli di astrazione (IR by Robot, IR by Cluster, IR by Area).
 - Fornire i dati ottenuti in output sulla GUI attraverso la quale l'utente si interfaccia al sistema per ottenere le informazioni che cerca;
 - Rieseguire tutti i passaggi precedenti a distanze di tempo regolari in modo da garantire dati il più possibile freschi e aggiornati all'utente.
-

A1.3 DB Requirements (da riempire a partire dalla Versione 2)

Il DB memorizza i dati provenienti dai robot durante un intervallo di tempo prestabilito, tenendo uno “storico” dei segnali di cambio stato ricevuti, consentendo in seguito di effettuare le operazioni necessarie al calcolo dell'IR.

Lo storage dei dati avviene attraverso una componente server dedicata, che ‘intercetta’ i segnali inviati dai robot ed effettua le operazioni di inserimento nel database. Sul database sarà memorizzato solo l'ultimo cambio di stato il tutto sovrascritto al relativo dato precedentemente memorizzato.

Oltre che ai segnali inviati dai robot, nel DB teniamo traccia anche del timestamp: dato di fondamentale importanza, il cui scopo è facilitare i calcoli relativi al periodo di inefficienza di un robot, e quindi semplificare il processo tramite cui si otterranno i valori di IR relativi ai cluster e alle aree.

A.2 Non Functional Requirements

<List and describe here the **most important** non functional requirements.>

BE CAREFUL NOT TO MAKE CONFUSION AMONG DIFFERENT NON FUNCTIONAL REQUIREMENTS. PLEASE FOCUS ON NO MORE THAN THE 3-4 MOST IMPORTANT NON FUNCTIONAL REQUIREMENTS-

Non Functional Requirement	Description
Usability	Bisogna coinvolgere gli utenti nel processo di progettazione ed eseguire test frequenti al fine di raccogliere feedback. Buona parte del giudizio sulla usability sarà determinato dall'utente in base alla qualità dell'interfaccia finale (in questo caso la dashboard).
Reliability	Il sistema deve essere sempre pronto a ricevere dati dai sensori dei robot e quindi rimanere costantemente online (24/7, 365gg/anno), nel nostro caso specifico si necessita di un minimo di up time pari al 98%. Inoltre deve fornire misurazioni costanti cercando, a contempo, di contenere il più possibile i costi, nel nostro caso è importante la precisione nel "timestamp" del cambio di stato ma non è richiesta particolare precisione nel valore del segnale essendo esso un semplice boolean.
Performance	Il sistema deve garantire dei tempi di risposta ottimali, più precisamente dovrà soddisfare 90.000 richieste al minuto provenienti da robot, senza aumentare eccessivamente i costi e/o i tempi di sviluppo.
Efficiency	Nel nostro caso l'architettura del sistema dovrà gestire fino a 90'000 robot, di conseguenza, supportare lo storage e il processing di almeno 90'000 messaggi al minuto. La dashboard sarà aperta su almeno 100 PC in linea e dovrà aggiornarsi automaticamente al massimo ogni 5 minuti (nel nostro caso siamo arrivati ad avere performance talmente alte da poter concedere all'utente il refresh ogni qualvolta lo desidera).

A.3 Content

*<Describe the **data provenance** (use of external API, web service, DB ...)>*

I dati vengono memorizzati in un database integrato nel Sistema, le tecnologie utilizzate sono MongoDB per quanto riguarda la base di dati e RoboMongo come DBMS.

A.4 Assumptions

<Briefly document, in this section, the most relevant requirement assumptions/decisions you had to made during your project>

- Viene dato per assunto che chiunque abbia progettato e/o scelto la parte software dei PC che verranno utilizzati dagli ingegneri e quindi i PC sopra i quali dovrà girare la nostra dashboard siano muniti di una versione del JDK pari a 1.8.0_152 o più recente per la corretta visualizzazione delle finestre e la corretta funzione della logica Java.
- Va inoltre dato per assunto che il sistema sia debitamente collegato in rete con ogni robot o dispositivo di ricezione segnali (quello che per noi è il Server) preferibilmente con una connessione cablata di tipo Gigabit Ethernet LAN in modo da garantire efficienza e velocità elevata.
- Abbiamo inoltre assunto la presenza di un server che funge sia da ricevitore dei segnali sia da bus dati per il trasferimento degli stessi al database ce ne esegue il relativo storage. Il Server si occupa solo della ricezione e salvataggio e NON dell'analisi dei dati.
- Al momento del primo avvio i robot dovranno avere una produttività del 100% ossia tutte le partizioni hardware dei robot devono rientrare nelle specifiche e quindi essere in fase di UP in quanto al primo arrivo i segnali saranno impostati di default a 1.

A.5 Prioritization

Requisiti listati secondo ordine di priorità.

<i>Requisiti</i>	Priorità	Commenti
<i>Invio dei dati</i>	Alta	Funzione indispensabile al corretto funzionamento dell'intero sistema
<i>Data storage</i>	Alta	È importante che i dati rimangano memorizzati sul database. Altrimenti sarebbe impossibile effettuare le operazioni di calcolo relative all'IR.
<i>Analisi real-time</i>	Media	Funzionalità indispensabile per l'analisi dei dati
<i>Calcolo IR</i>	Media	Funzionalità indispensabile per la produzione dell'inefficiency rate.
<i>Visualizzazione dati</i>	Media	Richiede che tutte le funzionalità precedenti siano eseguite correttamente, e consente all'utente di rapportarsi col sistema.

B. Analysis Model

<In this section, you shall report the Analysis Model produced for your system>

PLEASE REPORT: (i) ALWAYS the diagram to be discussed, (ii) the text explaining the decisions taken to create the diagram (that is, why did you create a certain analysis model design?)

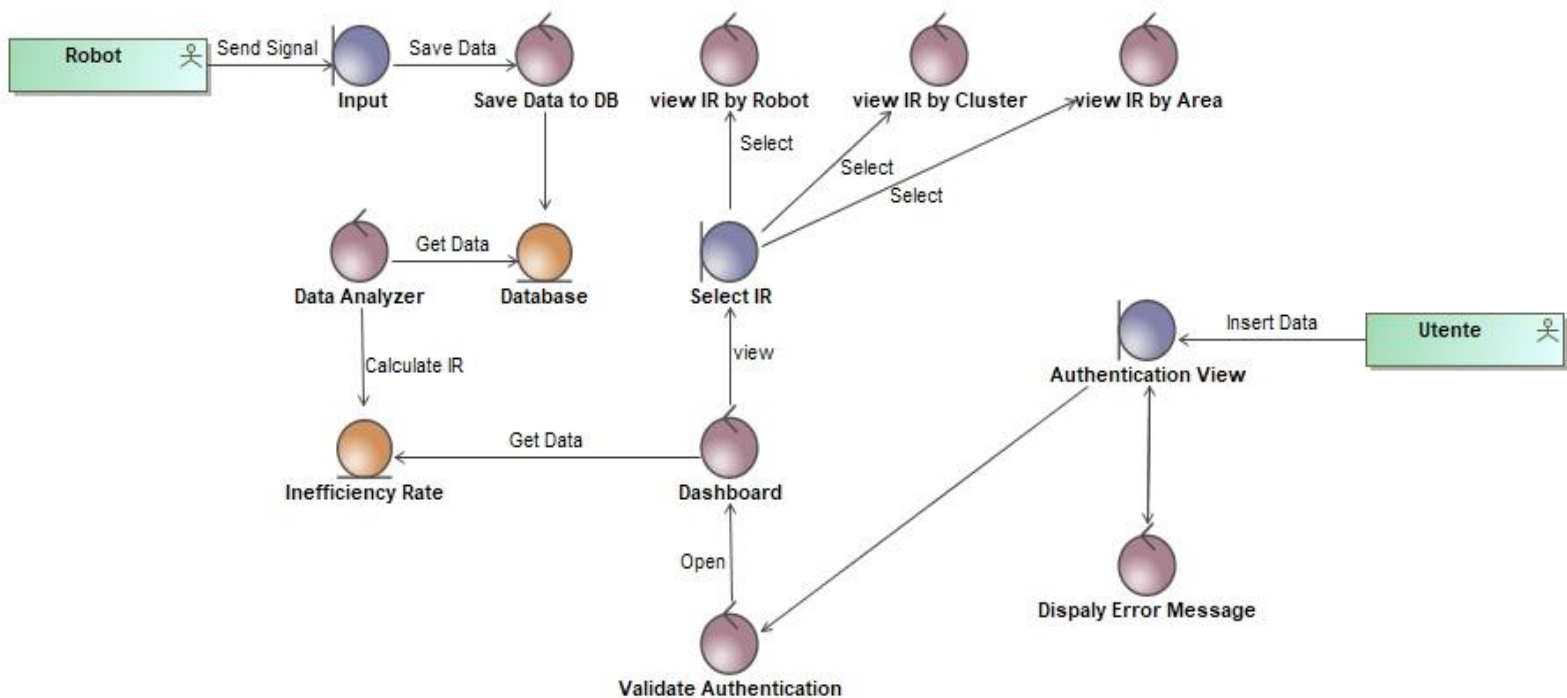


Figura 2- Robustness Diagram

Analizzando quindi gli Use Case generati in fase di collezione dei requisiti individuiamo le seguenti classi suddivise nelle tre tipologie previste da UML:

Boundary: Input, Select IR, Authentication View

Controller: Save Data to DB, Data Analyzer, Dashboard, View IR by Robot, View IR by Cluster, View IR by Area, Display Error Message, Validate Authentication

Entity: Data, Inefficiency Rate

L'utente si interfaccia con la schermata iniziale immettendo i dati di autenticazione, identificata da un oggetto boundary chiamato "Authentication View". Da qui, esitono due possibilità, la prima, rappresentata dall'oggetto di tipo controller "Display Error Message"

rappresenta il caso in cui l'autenticazione non vada a buon fine e quindi mostra a schermo un messaggio di errore eseguendo un refresh della pagina per dar modo all'utente di riprovare ad inserire i dati.

Nel secondo caso invece la fase di login è stata eseguita con successo attraverso l'oggetto di tipo controller "Validate Authentication" raggiungendo quindi la dashboard. Qui l'utente ha la possibilità di scegliere due modalità di visualizzazione relative all'inefficiency rate: IR by Robot, IR by Cluster oppure IR by Area.

Un robot ha la possibilità di interfacciarsi al sistema attraverso l'oggetto boundary "Input". Il controller "Save to Database" si occupa dello storage dei dati ricevuti all'interno della entity "Database".

Da questo punto, il controllo passa al controller "Data Analyzer", che recupera i dati necessari e li elabora, effettuando così il calcolo dell'IR. I risultati saranno poi presi dall'oggetto di tipo entity "Inefficiency Rate", pronti per essere recuperati dal controller "Dashboard".

C. Software Architecture

<Report here both the static and the dynamic view of your system design, in terms of a Component Diagram, Class Diagrams and their related Sequence Diagrams >

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

AVOID TO MAKE IT TOO COMPLEX AND FINE GRAINED. FOCUS MORE ON “HOW” THE COMPONENTS SHALL INTERACT IN ORDER TO SATISFY THE REQUIREMENTS

ADD INTERFACES and their parameters

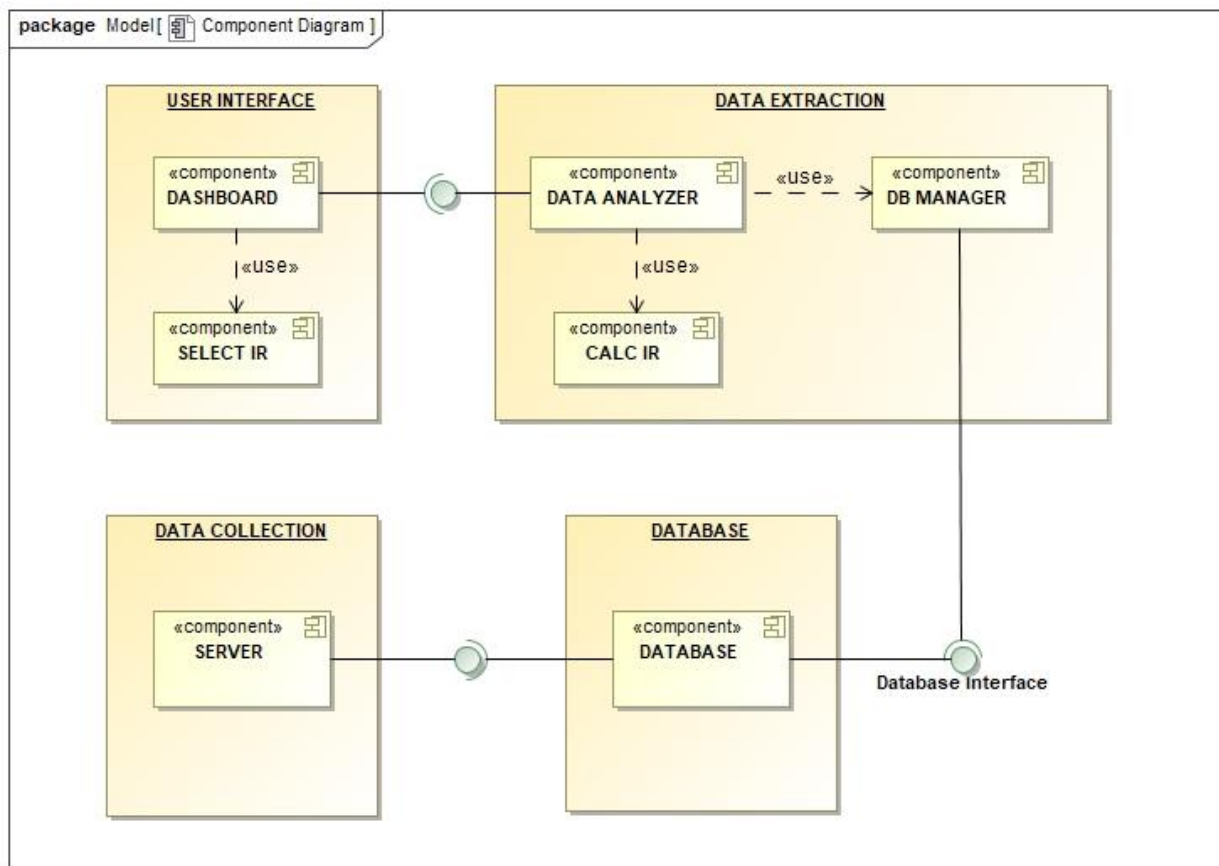


Figura 3- Component Diagram

Una delle componenti del nostro sistema, ossia l'USER INTERFACE, si interfaccia con la componente relativa alla DATA EXTRACTION che attraverso il DB Manager recupera i dati dal database per processarli. Il DATABASE riceve i dati della componente Server relativa alla DATA COLLECTION che ha come unico compito quello di memorizzare i dati ricevuti in input dai robot. Possiamo quindi distinguere il nostro sistema in 4 macro aree:

1. USER INTERFACE
2. DATA EXTRACTION
3. DATABASE
4. DATA COLLECTION

1- USER INTERFACE:

L'utente si relaziona al Sistema attraverso questa componente la quale è fornita di una interfaccia grafica (GUI) tramite cui un utente, scegliendo i dati che vuole ricevere tramite la componente SELECT IR, ottiene i dati calcolati dal DATA ANALYZER.

2- DATA EXTRACTION

La componente che si occupa della parte computazionale è il DATA ANALYZER. Il Data Analyzer si occupa dell'elaborazione dei dati ottenuti dal database tramite la componente DB Manager, sui dati ricevuti sarà eseguito il calcolo dell'IR con la relativa componente IR Calc. I dati saranno poi pronti per essere dati in risposta alla richiesta della Dashboard

3- DATABASE

Il Database riceve i dati da memorizzare dalla componente DATA COLLECTION e li fornisce, tramite l'interfaccia Database Interface, come risposta alle richieste del DB Manager.

4- DATA COLLECTION

E' la componente che si occupa di ricevere i dati dai robot e trasferirli al database per effettuarne la memorizzazione.

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

BE SURE THAT THE STRUCTURE IS SYNCHRONIZED WITH THE DYNAMIC VIEW

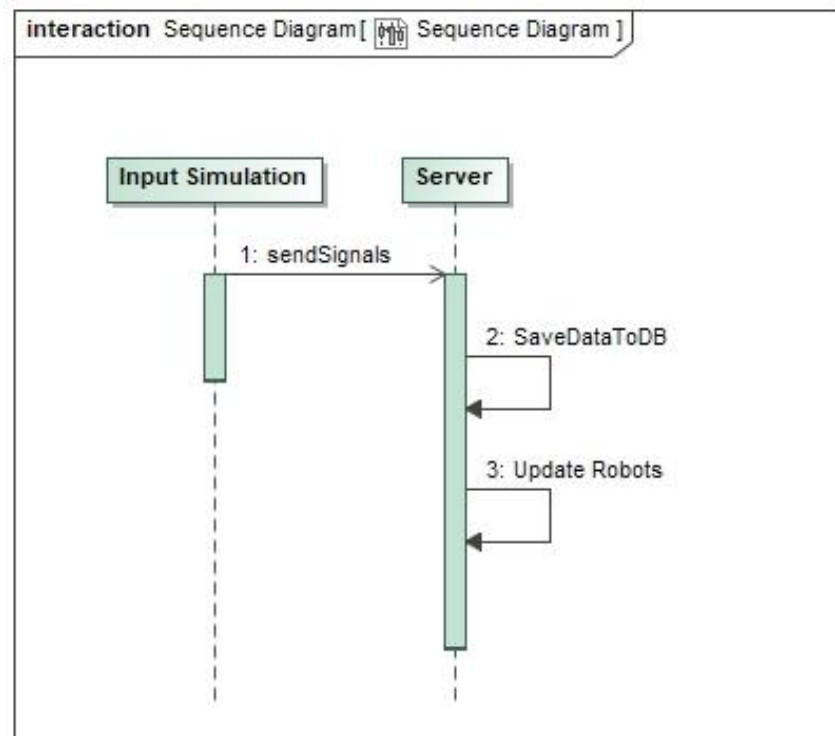


Figura 4- Sequence Diagram Scenario#1 (Data Storage)

Scenario che illustra il processo di invio dei segnali da parte dell'Input Simulation (come si evince dal nome questo processo simula l'invio dei dati dai robot) ed il salvataggio degli stessi.

1. sendSignal: Invio dei segnali simulati
2. SaveDataToDB: Invio dei dati al DB per eseguirne lo storage
3. Update Robots: Processo atto ad eseguire i controlli periodici sul DB per verificare la presenza di eventuali cambi di stato utili ai calcoli da eseguire.

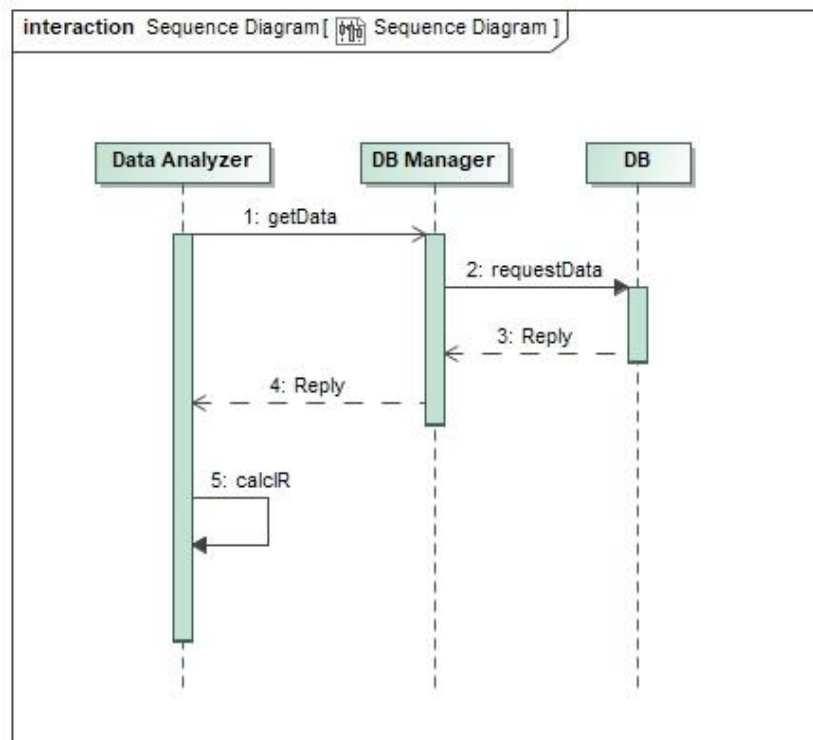


Figura 5- Sequence Diagram Scenario#2 (IR Calc)

Scenario che mostra il processo di data extraction, ovvero l’analisi dei dati prelevati dal database e produzione della percentuale di inefficiency rate.

1. **getData:** DataAnalyzer chiede i dati di cui ha bisogno al DBManager
2. **requestData:** Il DBManager si interfaccia con il database “inoltrandogli” la richiesta a sua volta ricevuta dal DataAnalyzer.
3. **Reply:** Il DB risponde alla richiesta del DBManager
4. **Reply:** Il DBManager a sua volta “inoltra” la risposta del DB al DataAnalyzer che era il richiedente originario dei dati
5. **calcIR:** il DataAnalyzer effettua il calcolo dell’Inefficiency Rate sui dati appena ricevuti.

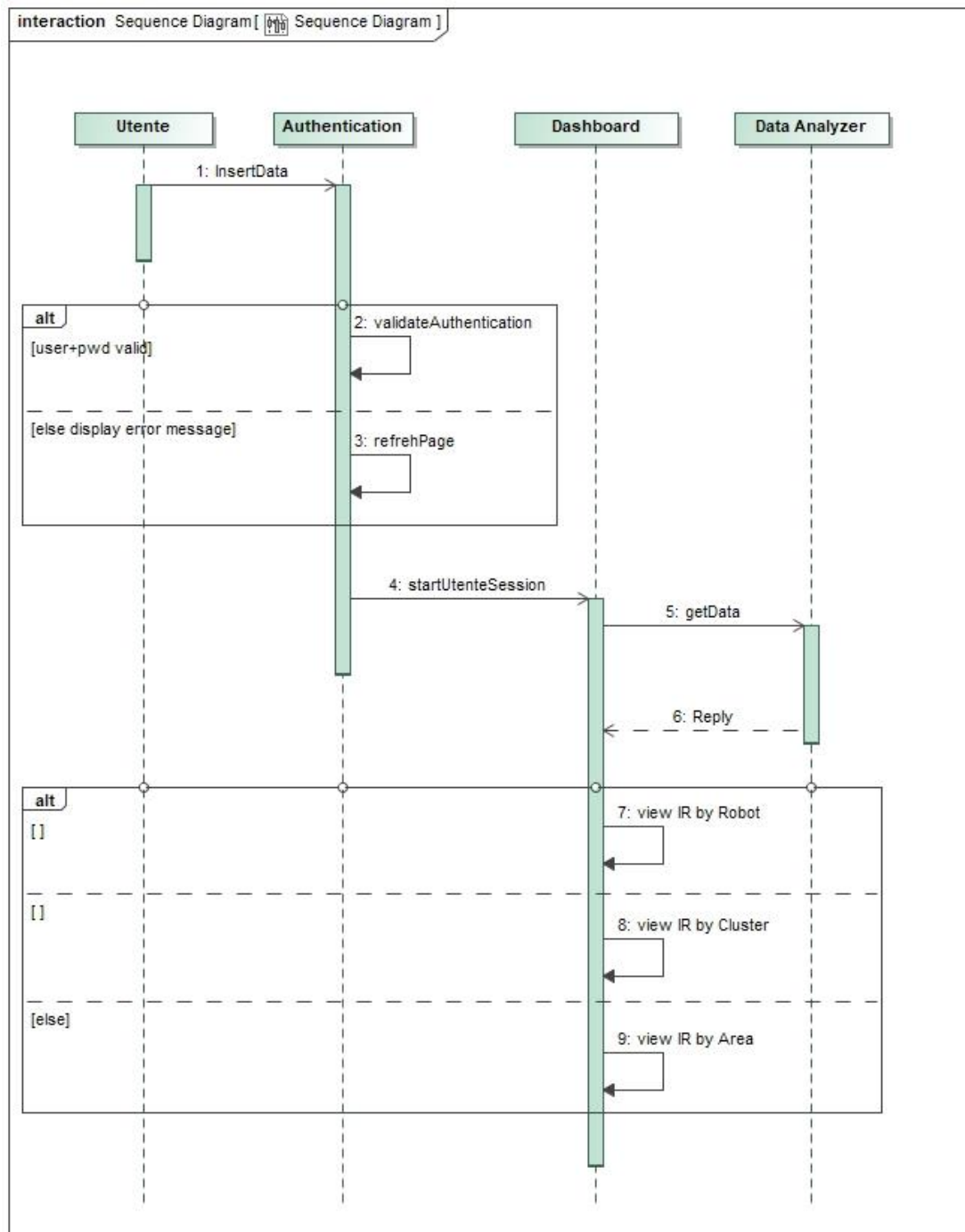


Figura 6- Sequence Diagram Scenario#3 (User)

Scenario che mostra l'interazione dell'utente con la Dashboard e la visualizzazione dei dati relativi all'IR.

1. `insertData`: L'utente inserisce nel form i dati necessari per l'autenticazione
2. `validateAuthentication`: (if) Controllo di validità sui dati inseriti per completare l'autenticazione.
3. `refreshPage`: (else) viene stampato un messaggio di errore che invita l'utente a reinserire i dati, viene eseguito un refresh della pagina per il nuovo tentativo
4. `startUtenteSession`: Avvio della sessione utente
5. `getData`: La dashboard effettua una richiesta per il recupero dei dati di IR dal `DataAnalyzer`.
6. `reply`: il `DataAnalyzer` fornisce i dati richiesti alla Dashboard.
7. `View IR by Robot` (alternative 1): L'utente invia una richiesta per visualizzare le informazioni desiderate (nella GUI per visualizzare l'IR by Robot bisogna comunque navigare nei cluster di conseguenza viene richiesto anche l'IR by Cluster anche se calcolati separatamente).
8. `View IR by Cluster` (alternative 2): L'utente invia una richiesta per visualizzare le informazioni relative ai cluster
9. `View IR by Area` (alternative 3): imposta la visualizzazione dell'IR by Area in modo tale da controllare tutti i cluster appartenenti ad una determinata area

D. ER Design

<Report here the Entity Relationship Diagram of the system DB>

Per questo progetto è stato scelto MongoDB, un database NoSQL.

Nel Db saranno inserite:

- Una collection chiamata “robot” contenente l’elenco di tutti i robot presenti nel database con le relative informazioni sui cluster e sulle aree. Nello specifico:
 - id: contiene l’id del Robot;
 - cluster: contiene l’id del cluster di appartenenza del robot in questione;
 - signal1: contiene lo stato relativo al sensore 1;
 - signal2: contiene lo stato relativo al sensore 2;
 - signal3: contiene lo stato relativo al sensore 3;
 - signal4: contiene lo stato relativo al sensore 4;
 - signal5: contiene lo stato relativo al sensore 5;
 - signal6: contiene lo stato relativo al sensore 6;
 - signal7: contiene lo stato relativo al sensore 7;
 - signal1Time: contiene il timestamp dell’ultimo cambiamento di stato relativo al signal1;
 - signal2Time: contiene il timestamp dell’ultimo cambiamento di stato relativo al signal2;
 - signal3Time: contiene il timestamp dell’ultimo cambiamento di stato relativo al signal3;
 - signal4Time: contiene il timestamp dell’ultimo cambiamento di stato relativo al signal4;
 - signal5Time: contiene il timestamp dell’ultimo cambiamento di stato relativo al signal5;
 - signal6Time: contiene il timestamp dell’ultimo cambiamento di stato relativo al signal6;
 - signal7Time: contiene il timestamp dell’ultimo cambiamento di stato relativo al signal7.

- Una collection chiamata “utenti” contenente l’elenco di tutte le coppie username e password degli utenti abilitati ad accedere alla dashboard. Nello specifico:
 - username: contiene il codice utente dell’ingegnere;
 - password: contiene la password relativa al codice utente dell’ingegnere.

E. Class Diagram of the implemented System

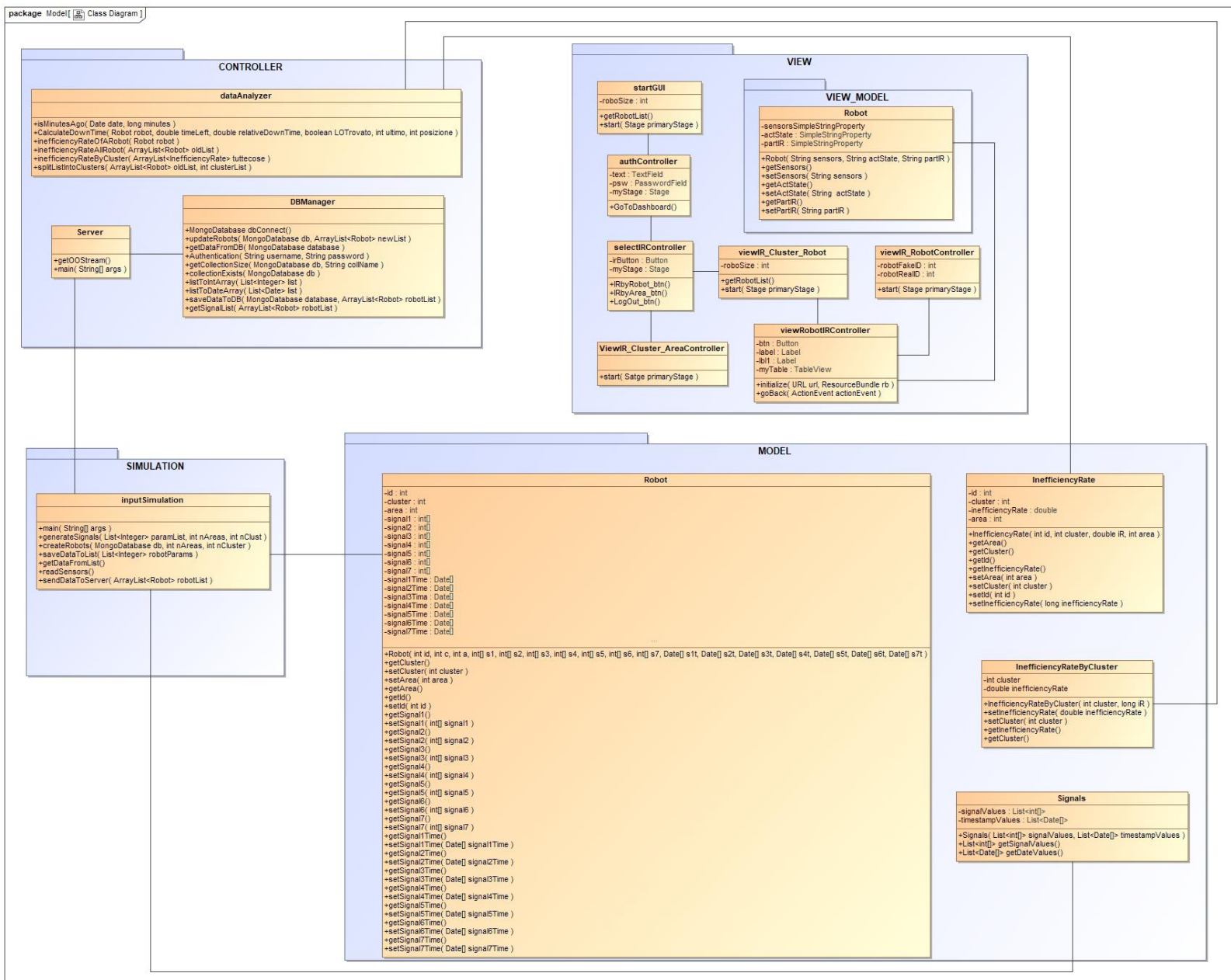


Figura 7- Class Diagram

Package Model:

Class Robot:

La classe Robot rappresenta un tipo di modello che ci permetterà di creare un oggetto di tipo robot composto dal proprio ID univoco, cluster di appartenenza, segnali e timestamp relativi ad ogni segnale.

Class InefficiencyRate:

La classe InefficiencyRate rappresenta un oggetto che associa ad un robot, identificato dal cluster e dall'area, la relativa Inefficiency Rate. Le funzioni principali di analisi restituiscono un array di questi oggetti dal quale è possibile estrapolare in maniera semplice il dato di Inefficiency rate di ogni robot da visualizzare.

Class InefficiencyRateByCluster:

La classe InefficiencyRateByCluster svolge le stesse funzioni della classe InefficiencyRate ma fornendo i dati di Inefficiency Rate per ogni singolo cluster.

Class Signals:

La classe Signals viene usata nell'inputSimulation per la generazione dei segnali random.

Package Controller:

Class dataAnalyzer:

La classe dataAnalyzer è composta da più funzioni tutte di elevata importanza che andremo a descrivere di seguito:

- La funzione isMinutesAgo prende in ingresso una data ed un long che rappresenta i minuti e restituisce TRUE se questa data è compresa tra: ora attuale e ora attuale-minutes. La funzione è utile per verificare se un segnale è cambiato durante l'ultima ora oppure in un arco temporale maggiore nel caso si decida di estendere il tempo di calcolo dell'IR.
- La funzione CalculateDownTime rappresenta la funzione principale dell'analizzatore e prende in ingresso un robot del quale verrà analizzata l'IR, un double rappresentante il tempo (in minuti) su cui calcolare l'IR e altri valori che devono essere inizializzati a 0 e saranno utili quando la funzione verrà richiamata in modo ricorsivo. La funzione analizza il primo cambio di stato dell'arco temporale e calcola il tempo di down parziale del segnale e viene richiamato in modo ricorsivo restituendo il tempo rimasto da analizzare, il tempo parziale di down appena calcolato.
- La funzione InefficiencyRateOfARobot calcola il tempo di down del robot attraverso al funzione precedente e ne restituisce la percentuale di inefficienza sulla base di 60 minuti.
- La funzione InefficiencyRateAllRobot restituisce un array con tutte le Inefficiency Rate di tutti i robot.

- InefficiencyRateByCluster dato l'array creato in precedenza restituisce l'IR di ogni cluster.

Class Server:

La classe Server riceve tutti i dati che vengono inviati dalla classe inputSimulation e attraverso il DBManager si occupa dell'interazione col database. Uno dei compiti principali di questa classe è quello di verificare se i segnali sono stati generati per la prima volta oppure se esiste già una collection nel database, in caso già esistano robot nel db il server farà riferimento al file params.tmp contenente tutte le info relative ai cluster di robot (dimensione e numero cluster), grazie a questo file infatti sarà possibile effettuare l'aggiornamento dei dati sul db basandosi sui parametri in questione.

Class DBManager:

Contiene tutte le funzioni per la gestione (inserimento, restituzione e cambio dati) del Database.

Package View:

Class startGUI:

La classe startGUI contiene il metodo per invocare la prima view della dashboard, ossia quella di autenticazione.

Class authController:

Questa classe è la prima ad essere invocata, contiene principalmente i controlli per l'autenticazione dell'utente e il messaggio di errore che viene stampato se l'utente inserisce una combinazione user code - password errata.

Class SelectIRController:

È la finestra che si aprirà nel caso in cui l'utente si sia autenticato con successo, in essa sono presenti i comandi per effettuare il log out oppure per scegliere tra le 2 visualizzazioni possibili dei dati.

Class ViewIR Cluster AreaController:

È la finestra che ci permette di avere una panoramica dell'inefficienza con la massima astrazione ossia visualizzandola by Area, facendo click su una delle aree sarà possibile espanderla fino a visualizzare l'inefficienza dei cluster che la compongono

Class viewIR Cluster Robot:

questa finestra ci permette, al contrario dell'altra di avere una panoramica dettagliata sui cluster e i singoli robot, accedendovi infatti partiremo col visualizzare i dati dei cluster, selezionandone uno lo esanderemo fino a visualizzare i dati di tutti i robot che lo compongono che a loro volta potranno essere selezionati e mostrati nel minimo dettaglio.

Class viewRobotIRController:

E' la finestra che, una volta selezionato un singolo robot, ci mostra nel dettaglio informazioni non visualizzabili diversamente riguardanti ognuno dei 7 segnali che il robot possiede. Infatti troveremo il parziale di inefficienza di ogni segnale e lo stato attuale (UP o DOWN) di ogni segnale.

Class ViewIR_RobotController:

La classe visualizza una lista di robot all'interno di un cluster e permette di visualizzarne il relativo tasso di IR. E' inoltre possibile visualizzare lo stato attuale dei sensori di un singolo robot tramite l'apposita finestra.

Package View_Model:

Class Robot:

L'oggetto robot contiene tutti i dati che serviranno alla finestra viewRobotIRController per visualizzare i dati di diagnostica specifici per ogni robot.

Package Simulation:

Class InputSimulation:

La classe InputSimulation è la classe che si occupa della generazione di robot e cambi stato casuali. I valori di ogni sensore vengono generati in un range specifico che può essere definito dall'utente, se inputSimulation non trova nessun file .tmp farà partire la generazione automatica dei robot per poi crear un file .tmp con i dati appena simulati, in questo modo durante la prima esecuzione del programma prima verrà generato il file e poi verranno simulati i cambi stato in loop sui robot all'interno del file .tmp all'interno di un intervallo di tempo definito. Questa classe contiene anche la funzione per inviare i dati rielaborati al database tramite TCP.

F. Design Decisions

<Document here the **5** most important design decisions you had to take. You can use both a textual or a diagrammatic specification.>

THIS IS A VERY IMPORTANT PART. BE SURE TO DOCUMENT THE 5 MOST IMPORTANT DECISIONS (related to your requirements and design) YOU MADE

1- Database: mongoDB

MongoDB è un DBMS non relazionale, orientato ai documenti. Classificato come un database di tipo NoSQL, MongoDB si allontana dalla struttura tradizionale basata su tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico (MongoDB chiama il formato BSON), rendendo l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce.

Abbiamo scelto di utilizzare un database non relazionale perchè presenta alcune caratteristiche utili allo sviluppo della nostra applicazione. In particolare, gode di elevata velocità computazionale, anche al crescere del volume dei dati, il che lo rende adatto a soddisfare i requisiti di performance e scalabilità. Inoltre, essendo basato su logiche di lettura dati più semplici rispetto a quelle dei database relazionali, porterebbe ad una riduzione significativa dei tempi di sviluppo.

La scelta sull'utilizzo di mongoDB è stata influenzata dal fatto che rappresenta uno dei più popolari database open-source, dispone di ampia documentazione ed è largamente supportato dai principali linguaggi di programmazione.

2 – Linguaggio di programmazione: Java

La scelta del linguaggio di programmazione è ricaduta su Java, in quanto rappresenta l'alternativa più adatta alle nostre esigenze di portabilità.

Java, essendo basato su macchina virtuale (JVM) garantisce un comportamento simile in contesti di esecuzione diversi. Sarebbe quindi possibile estendere il nostro sistema a target diversi (intesi come OS) senza dover effettuare modifiche importanti alla parte implementativa, massimizzando la portabilità del sistema.

3 – Pattern Architettuale: Model-View-Controller (MVC)

Il pattern architetturale scelto per la realizzazione del nostro sistema è MVC (Model View Controller), il pattern è basato sulla separazione dei compiti fra i component software che interpretano tre ruoli principali:

- il **model** fornisce I metodi per accedere ai dati utili all'applicazione;

- il **view** visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
- il **controller** riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

La separazione delle logiche garantisce una maggiore manutenibilità del sistema in seguito a modifiche o aggiunte future, senza dover cambiare radicalmente il design architetturale.

4 - Graphical User Interface (GUI): JavaFX 8

JavaFX 8 è una libreria grafica inclusa in Java SE 8 che consente di sviluppare Rich Client Application, delle applicazioni dotate di un'interfaccia utente costruita attraverso un insieme di componenti UI (User Interface) predefiniti.

Uno degli aspetti fondamentali di JavaFX è la facilità con cui viene implementato il pattern MVC (Model-View-Controller), integrandosi perfettamente con le scelte di design precedenti.

Inoltre il look and feel può essere modificato facilmente utilizzando un foglio di stile CSS senza la necessità di dover scrivere nulla nel codice che definisca l'aspetto grafico dei componenti visualizzati, se non le chiamate ad una classe specifica o agli id CSS. Con JavaFX 8 è anche possibile utilizzare Scene Builder un Visual Layout Tool, ovvero uno strumento che consente all'utente di progettare interfacce più rapidamente, infatti l'utente utilizzando il drag & drop può posizionare i componenti dell'interfaccia di cui ha bisogno all'interno di un area di lavoro, modificarne le proprietà e collegarli direttamente al controller o a file .css, Scene Builder genera automaticamente un file FXML contenente appunto la costruzione dell'interfaccia, le sue proprietà e le chiamate alle altre classi o alla logica usata all'interno.

5 – Architettura di sistema: Desktop Application

I motivi principali per cui abbiamo deciso di realizzare un'applicazione desktop sono i seguenti:

Flessibilità: Un'applicazione desktop consente maggiore integrazione con dispositivi hardware (es: sensori dei robot).

Performance: In genere, un'applicazione desktop consente una maggiore ottimizzazione relativa all'ambiente operativo e quindi un aumento significativo in termini di performance rispetto ad una web application, nel nostro caso specifico sappiamo che coniugandola con il linguaggio java perderemo quasi del tutto questo beneficio a favore però di eventuali futuri cambi o aggiornamenti di natura software o hardware.

Inoltre avendo già sviluppato desktop application questa scelta ha consentito al team di ridurre i tempi di sviluppo.

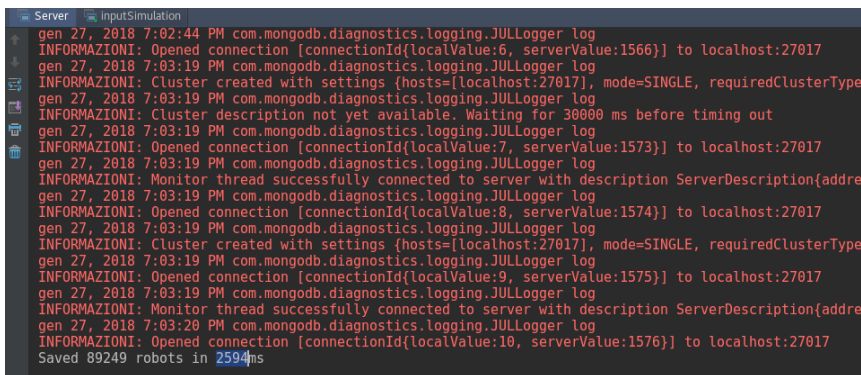
G. Explain how the FRs and the NFRs are satisfied by design

<Report in this section how the design you produced satisfies the FRs and the NFRs>

Con questa nuova versione del prototipo siamo riusciti a soddisfare i principali FR e NFR che avevamo prefissato. Il tutto rispettando le specifiche del progetto che richiedevano lo storage e il processing di circa 90'000 messaggi di cambi stato al minuto. Avendo scelto di utilizzare mongoDB, non abbiamo riscontrato particolari problemi di storage dei dati, essendo riusciti a sfruttare a pieno le potenzialità del database non relazionale. Inoltre, avendo utilizzato il protocollo di trasmissione dati TCP, siamo riusciti ad ottenere tempi ottimali di trasferimento dei dati verso il server.

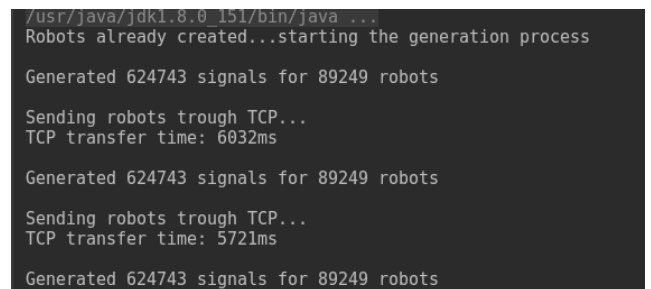
Nella nostra simulazione, il flusso di informazioni viene inviato ad un pc connesso in rete locale via ethernet, il ricevente dispone inoltre di un'istanza in esecuzione di mongodb per il salvataggio dei dati.

Sessione di invio dei dati verso il server (richiede in media 5000ms):



```
Server  InputSimulation
gen 27, 2018 7:02:44 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Opened connection [connectionId{localValue:6, serverValue:1566}] to localhost:27017
gen 27, 2018 7:03:19 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType
gen 27, 2018 7:03:19 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Cluster description not yet available. Waiting for 30000 ms before timing out
gen 27, 2018 7:03:19 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Opened connection [connectionId{localValue:7, serverValue:1573}] to localhost:27017
gen 27, 2018 7:03:19 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Monitor thread successfully connected to server with description ServerDescription{addre
gen 27, 2018 7:03:19 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Opened connection [connectionId{localValue:8, serverValue:1574}] to localhost:27017
gen 27, 2018 7:03:19 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType
gen 27, 2018 7:03:19 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Opened connection [connectionId{localValue:9, serverValue:1575}] to localhost:27017
gen 27, 2018 7:03:19 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Monitor thread successfully connected to server with description ServerDescription{addre
gen 27, 2018 7:03:20 PM com.mongodb.diagnostics.logging.JULLogger log
INFORMAZIONI: Opened connection [connectionId{localValue:10, serverValue:1576}] to localhost:27017
Saved 89249 robots in 2594ms
```

Figura 8- Screenshot Scenario update e storage dati



```
/usr/java/jdk1.8.0_151/bin/java ...
Robots already created...starting the generation process

Generated 624743 signals for 89249 robots

Sending robots trough TCP...
TCP transfer time: 6032ms

Generated 624743 signals for 89249 robots

Sending robots trough TCP...
TCP transfer time: 5721ms

Generated 624743 signals for 89249 robots
```

Figura 9- Screenshot Scenario generazione segnali

I dati vengono generati ad intervalli di tempo regolari definiti nella classe Start. Il tempo necessario a generare i segnali per 90000 robot, in media è di 500ms, invio escluso.

Per quanto riguarda il server, i tempi relativi alla memorizzazione sul DB sono di 5000/6000ms.

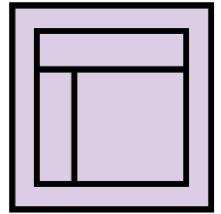
Di seguito troviamo lo schema dei dati memorizzati nel database:

Key	Value	Type
(1) ObjectId("5a6cbee8bfe0ed303c465921")	{ 17 fields }	Object
_id	ObjectId("5a6cbee8bfe0ed303c465921")	ObjectId
id	4001	Int32
cluster	5	Int32
signal1	[1 element]	Array
[0]	1	Int32
signal2	[1 element]	Array
signal3	[7 elements]	Array
signal4	[1 element]	Array
signal5	[1 element]	Array
signal6	[1 element]	Array
signal7	[1 element]	Array
signal1Time	[1 element]	Array
[0]	2018-01-27 14:50:01.750Z	Date
signal2Time	[1 element]	Array
signal3Time	[7 elements]	Array
signal4Time	[1 element]	Array
signal5Time	[1 element]	Array
signal6Time	[1 element]	Array
signal7Time	[1 element]	Array
(2) ObjectId("5a6cbee8bfe0ed303c465922")	{ 17 fields }	Object
(3) ObjectId("5a6cbee8bfe0ed303c465923")	{ 17 fields }	Object
(4) ObjectId("5a6cbee8bfe0ed303c465924")	{ 17 fields }	Object
(5) ObjectId("5a6cbee8bfe0ed303c465925")	{ 17 fields }	Object
(6) ObjectId("5a6cbee8bfe0ed303c465926")	{ 17 fields }	Object
(7) ObjectId("5a6cbee8bfe0ed303c465927")	{ 17 fields }	Object
(8) ObjectId("5a6cbee8bfe0ed303c465928")	{ 17 fields }	Object
(9) ObjectId("5a6cbee8bfe0ed303c465929")	{ 17 fields }	Object

Figura 11- Screenshot Scenario dati memorizzati nel DB

Riteniamo che i risultati raggiunti in fase definitiva siano ampiamente sotto la soglia indicata nella specifica del progetto.

H. Effort Recording



GANTT

Make a GANTT documenting the tasks and timing you expect to spend on the deliverable. Try to be as precise as possible. Check, after the deliverable deadline, if and how you satisfied (or not) the deadlines.

Logging

As you are working on the assignment, record what you are doing and how long you spent. As a rule of thumb, you should add a log entry every time you switch tasks. For example, if you do something for two hours straight, that can be one log entry. However, if you do two or three things in half an hour, you must have a log entry for each of them. You do not need to include time for logging, but should include the time spent answering the other parts of this question.

For this purpose, please use the **LogTemplate.xls** file.

Categorization

When logging the time spent on the project, please create different sub- categories. Specifically, it is important to clearly distinguish between two main categories: the time spent for “**learning**” (the modeling languages, the tools, etc.) from the time needed for “**doing**” (creating the models, taking the decisions, ...). Learning tasks are in fact costs to be paid only once, while doing costs are those that will be repeated through the project. For each category, please define sub-categories. Examples follow. You may add other sub-categories you find useful.

Learning

- Requirements Engineering
- Non functional Requirements
- Use Case Diagrams
- Tool study

Doing:

- Requirements discovery
- Requirements Modeling (UC diagrams)

Summary

Based on the attributes defined above, calculate the summary statistics of the time spent for “learning”, the time spent for “doing”, and the total time.

Statistics

Note: this Deliverable report shall document only the Summary Statistics for the different deliverables (D1, D2, and Final). Detailed information shall be reported in the Excel file.

COPY HERE (computed from the spreadsheet): i) the total number of hours spent by the group (that is, hours per task X number of people working on that task), ii) the time spent for LEARNING and for DOING

TOTAL TIME: 101h
LEARNING TIME: 16.5h
DOING TIME: 84.5h

Appendix. Code

<Report in this section a **documented** version of the produced code>

<Discuss how your code satisfies the FR and NFR>

<Show some screenshots of the code behavior>

Funzione presente nella classe inputSimulation incaricata di generare i segnali ad intervalli definiti dall'utente. Al primo avvio, crea un file .tmp contenente tutti dettagli relativi ai cluster. Così da permettere la generazione di segnali sempre per lo stesso numero di robot presenti nel DB.

```
File f = new File("params.tmp");

if (!f.exists()) {

    System.out.println("O_O There are no robots in the system O_O - STARTING SIMULATION NOW");
    // Create the robots and return the dimension of each cluster to generate signals
    List<Integer> robotsCount = createRobots(DBManager.dbConnect(), 100);
    saveDataToList(robotsCount);
    System.out.println("All done, ready to generate signals...");

} else System.out.println("Robots already created...starting the generation process");

Timer t = new Timer();
t.schedule(new TimerTask() {
    @Override
    public void run() {

        List<Integer> paramList = null;
        try {
            paramList = getDataFromList();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        try {

            // Start generating signals
            generateSignals(paramList);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}, 0, 36000);
}
```

La funzione `readSensors()` si occupa della generazione dei segnali casuali per i robot, modificando i parametri in essa contenuti, è possibile impostare il sistema a proprio piacimento, e quindi ottenere risultati diversi sulla dashboard.

```
    Signals sigArray = null;
    Date signalTime = null;
    Date dates[] = new Date[7];
    int signalsArray[] = new int[7];

    List<int[]> intList = new ArrayList<int[]>();
    List<Date[]> dateList = new ArrayList<Date[]>();

    int randomNum;

    // Every robot can send up to 7 signals
    for (int s = 0; s < 7; s++) {

        // Generate signal S1 and get its timestamp
        randomNum = ThreadLocalRandom.current().nextInt(0, 1000);
        if (randomNum > 998) signalsArray[0] = 0;
        else signalsArray[0] = 1;

        signalTime = new Date(System.currentTimeMillis());
        dates[0] = signalTime;

        // Generate signal S2 and get its timestamp
        randomNum = ThreadLocalRandom.current().nextInt(0, 200);
        if (randomNum > 198) signalsArray[1] = 0;
        else signalsArray[1] = 1;

        signalTime = new Date(System.currentTimeMillis());
        dates[1] = signalTime;

        // Generate signal S3 and get its timestamp
        randomNum = ThreadLocalRandom.current().nextInt(0, 100);
        if (randomNum > 98) signalsArray[2] = 0;
        else signalsArray[2] = 1;

        signalTime = new Date(System.currentTimeMillis());
        dates[2] = signalTime;

        // Generate signal S4 and get its timestamp
        randomNum = ThreadLocalRandom.current().nextInt(0, 200);
        if (randomNum > 198) signalsArray[3] = 0;
        else signalsArray[3] = 1;

        signalTime = new Date(System.currentTimeMillis());
        dates[3] = signalTime;

        // Generate signal S5 and get its timestamp
        randomNum = ThreadLocalRandom.current().nextInt(0, 100);
        if (randomNum > 98) signalsArray[4] = 0;
        else signalsArray[4] = 1;

        signalTime = new Date(System.currentTimeMillis());
        dates[4] = signalTime;

        // Generate signal S6 and get its timestamp
        randomNum = ThreadLocalRandom.current().nextInt(0, 100);
        if (randomNum > 98) signalsArray[5] = 0;
        else signalsArray[5] = 1;
```

```
signalTime = new Date(System.currentTimeMillis());
dates[5] = signalTime;

// Generate signal S7 and get its timestamp
randomNum = ThreadLocalRandom.current().nextInt(0, 100);
if (randomNum > 98) signalsArray[6] = 0;
else signalsArray[6] = 1;

signalTime = new Date(System.currentTimeMillis());
dates[6] = signalTime;
```

Funzione che si occupa dell'invio dati da client (inteso come inputSimulation) a server (controller/Server.java).

```
// Send the list to the server via sockets
public static void sendDataToServer(ArrayList<Robot> robotList) throws IOException {

    // Hostname and port configuration
    String host = "localhost";
    int port = 25001;

    InetAddress address = InetAddress.getByName(host);
    socket = new Socket(address, port);

    // Open the stream
    ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());

    // Measure execution time
    long startTime = System.currentTimeMillis();
    System.out.println("");
    System.out.println("Sending robots trough TCP...");

    // Measure execution time
    startTime = System.currentTimeMillis();

    // Write the entire robot list through sockets
    oos.writeObject(robotList);
    oos.close();

    // Record the execution time and display it on screen
    long stopTime = System.currentTimeMillis();
    long elapsedTime = stopTime - startTime;
    System.out.println("TCP transfer time: " + elapsedTime + "ms");
}
```

Codice del server che accetta la lista di robot generati dal client ed effettua le operazioni di salvataggio e aggiornamento. I controlli, individuano se è necessario fare l'update o inserimento dei dati inviati dal client (simulatore).

```
ServerSocket serverSocket = new ServerSocket(25001);
System.out.println("Server Started and listening to the port: 25000");
ObjectInputStream ois = null;

while (true) {

    //Reading the message from the client
```

```
socket = serverSocket.accept();
InputStream is = socket.getInputStream();
ois = new ObjectInputStream(is);

// Read the object from client
robotList = (ArrayList<Robot>) ois.readObject();

// Measure execution time
long startTime = System.currentTimeMillis();

File f = new File("params.tmp");

// Server-side check, if we find that the params.tmp is missing
// It means we have to save some robots to the db before generating signals for them...
boolean dbEmpty = DBManager.collectionExists(DBManager.dbConnect());

if (!dbEmpty) {

    // Save robots in the db
    DBManager.saveDataToDB(DBManager.dbConnect(), robotList);

    long stopTime = System.currentTimeMillis();
    long elapsedTime = stopTime - startTime;
    System.out.println("Saved " + robotList.size() + " robots in " + elapsedTime + "ms");
} else {

    // Update existing robots signals
    System.out.println("");
    System.out.println("Got the list from the client...updating robot entries in the db...");

    DBManager.updateRobots(DBManager.dbConnect(), robotList);
}
}
```

Il DBManager contiene tutto il codice relativo alle procedure di inserimento, aggiornamento e ottenimento delle informazioni dal DB.

In particolare, il team ha deciso di riportare qui sotto la funzione di update, che è stata oggetto di studio nelle fasi di progettazione del codice. La nostra funzione, invece che eseguire un normale aggiornamento tramite le procedure standard di update() di mongoDB (che in presenza di numerose informazioni si sono rivelate troppo lente). Effettua una copia in locale dell'intero database, riconosce i cambi di stato della nuova lista inviata dal server ed effettua un nuovo inserimento dell'intera collection nel DB.

Come studiato nella documentazione di mongo, spesso le operazioni di replace risultano più veloci di quelle di update. Nel nostro caso, abbiamo velocizzato ulteriormente il processo con l'insert. Il tempo di esecuzione dell'inserimento è in media di: 2000ms.

```
/* Update the list of robots in the db */
public static void updateRobots(MongoDatabase db, ArrayList<Robot> newList) throws IOException {

    ArrayList<Robot> oldList = DBManager.getDataFromDB(DBManager.dbConnect());

    // Delete the collection to insert a new one
    MongoCollection myCollection = db.getCollection("robot");
```

```
for (int i = 0; i < oldList.size(); i++) {  
  
    if (oldList.get(i).getSignal1()[oldList.get(i).getSignal1().length - 1]  
        != newList.get(i).getSignal1()[0]) {  
  
        int size = oldList.get(i).getSignal1().length;  
  
        int tempArray[] = new int[size + 1];  
        Date tempDate[] = new Date[size + 1];  
  
        for (int k = 0; k < oldList.get(i).getSignal1().length; k++) {  
            tempArray[k] = oldList.get(i).getSignal1()[k];  
        }  
  
        for (int k = 0; k < oldList.get(i).getSignal1Time().length; k++) {  
            tempDate[k] = oldList.get(i).getSignal1Time()[k];  
        }  
  
        tempArray[tempArray.length - 1] = newList.get(i).getSignal1()[0];  
        tempDate[tempDate.length - 1] = newList.get(i).getSignal1Time()[0];  
  
        oldList.get(i).setSignal1(tempArray);  
        oldList.get(i).setSignal1Time(tempDate);  
  
    }  
}
```

... abbiamo evitato di riportare tutto il codice, essendo il file composto da oltre 500 righe di codice. È possibile visualizzare la funzione completa ed il resto del codice su https://github.com/mikeshx/sw_eng/blob/master/src/controller/DBManager.java.

Altra componente fondamentale del sistema è rappresentata dal dataAnalyzer (che abbiamo già approfondito in precedenza), l'algoritmo si occupa del calcolo dell'IR su una lista ottenuta tramite apposite funzioni dal database. Non rappresentiamo qui l'intera funzione, in quanto risulterebbe troppo pesante per la lettura del documento. È possibile reperirla sul repository di progetto.

https://github.com/mikeshx/sw_eng/blob/master/src/controller/dataAnalyzer.java

Abbiamo cercato di riportare in questa sezione le parti di codice più significative, dato che sarebbe impossibile commentare tutto il lavoro svolto e dato che buona parte del codice è commentata e quindi di facile comprensione.

Il nostro repository contiene tutti i sorgenti aggiornati, una copia del database, e le librerie necessarie al funzionamento del programma. È inoltre disponibile un file readme che copre tutta la parte di testing dell'applicazione nel link che segue:

https://github.com/mikeshx/sw_eng