ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

**SCUOLA DI SCIENZE**
**Corso di Laurea in Informatica per il Management**

# Progettazione e sviluppo del pannello grafico in JavaFX per FREEDA


**Relatore:**
**Prof.**
**AMADINI ROBERTO**

**Presentata da:**
**ROSSI TOMMASO**

**Correlatore:**
**Dott.**
**GAZZA SIMONE**



**Sessione Unica**
**Anno Accademico 2023/2024**

# Introduzione

FREEDA (Failure-Resilient, Energy-Aware, and Explainable Deployment of Microservice-based Applications over Cloud-IoT infrastructures) è un progetto di ricerca avanzata del programma PRIN, realizzato in collaborazione con l'Università di Pisa e il Politecnico di Milano. L'obiettivo è sviluppare strumenti che facilitino la gestione adattiva delle applicazioni basate su microservizi (MSA) nei dispositivi IoT.

L'adozione di un'architettura a microservizi consente di realizzare sistemi modulari e scalabili, in grado di evolversi in base alle esigenze degli utenti e alle condizioni di rete. FREEDA si concentra sulla creazione di tecniche di deployment energeticamente efficienti e sulla fornitura di spiegazioni dettagliate riguardo alle decisioni prese durante la distribuzione e la gestione delle applicazioni.

Il progetto si inserisce nel contesto della crescente diffusione dell'IoT e dell'aumento della potenza di calcolo dei dispositivi di rete, fattori che hanno favorito lo sviluppo di infrastrutture Cloud distribuite. In tale modello, il calcolo avviene non solo nei datacenter centrali, ma anche ai margini della rete, sfruttando le risorse degli Edge devices. Questo approccio riduce la latenza e migliora l'efficienza energetica, pur presentando sfide significative in termini di resilienza e complessità gestionale, sfide che FREEDA intende affrontare con soluzioni innovative.

Un aspetto fondamentale del progetto è l'impiego di tecniche di continuous reasoning, che permettono di adattare in tempo reale la configurazio-

ne dei microservizi in base alle condizioni di rete e alla disponibilità delle risorse, garantendo così un deployment dinamico e ottimale anche in contesti complessi. Inoltre, l'integrazione di metodologie di explainability offre trasparenza nelle scelte di deployment, permettendo ai professionisti DevOps di monitorare e intervenire proattivamente per adattare il sistema alle esigenze operative.

In sintesi, FREEDA rappresenta un approccio innovativo per la gestione automatizzata e ottimizzata delle MSA su infrastrutture Cloud-IoT, contribuendo al progresso scientifico e all'adozione di pratiche IT sostenibili, in linea con gli obiettivi europei di crescita sostenibile e neutralità climatica.

# Obbiettivi della Tesi

Il principale obiettivo della presente tesi è la realizzazione di un'interfaccia grafica intuitiva e user-friendly, concepita per guidare l'utente nella progettazione della struttura di una MSA. Attraverso questa interfaccia, l'utente potrà inserire agevolmente tutti gli elementi necessari per definire il deployment della MSA, configurando in maniera semplice i microservizi, le relative dipendenze, nonché i vincoli legati alle risorse, alla resilienza e alla sostenibilità energetica.

Nell'ambito della tesi viene presentato lo sviluppo di un pannello grafico basato su JavaFX [JavaFX], progettato per rendere l'interfaccia dell'applicativo estremamente intuitiva. Tale pannello facilita la gestione delle configurazioni e delle scelte di deployment, offrendo una visualizzazione chiara e dettagliata delle opzioni disponibili. Le strutture dei dati salvate dall'interfaccia sono state appositamente progettate per agevolare la generazione automatica dei file YAML di configurazione.

Parallelamente è stata sviluppata una funzionalità, non di mia diretta competenza, che, partendo dall'interfaccia grafica, genera file di configurazione in formato YAML per FREEDA; questo modulo si integra perfettamente con l'interfaccia grafica, contribuendo a fornire un prodotto finale completo e funzionale, semplificando i processi di deployment in ambienti Cloud-IoT. Grazie a questa integrazione e mediante l'impiego di un solver adeguato, sarà possibile individuare la soluzione ottimale per il deployment delle MSA.

# Indice

# Capitolo 1

# Importanza dell'Interfaccia Grafica

## 1.1 L'importanza delle GUI nelle Applicazioni Software

Nell'era digitale, l'interfaccia grafica (Graphical User Interface, GUI) rappresenta il punto di contatto tra l'utente e il sistema. Un'interfaccia intuitiva e ben progettata è fondamentale per garantire un'interazione fluida e soddisfacente, migliorando l'accessibilità e riducendo la curva di apprendimento del software per gli utenti.

Le GUI svolgono un ruolo essenziale in molteplici settori, dai sistemi gestionali alle applicazioni consumer, offrendo strumenti visivi e interattivi che consentono agli utenti di eseguire operazioni complesse con semplicità. Questo è particolarmente rilevante nei contesti in cui la gestione di risorse, configurazioni o processi è cruciale, come nel caso del progetto FREEDA.

Un'interfaccia utente efficace non solo migliora l'esperienza dell'utente finale, ma contribuisce anche a ridurre il rischio di errori operativi, aumentando la produttività e l'efficienza. Questo aspetto è particolarmente importante

in ambienti professionali, dove decisioni rapide e accurate possono avere un impatto significativo.

## 1.2 L'UI in FREEDA

Nel progetto FREEDA, l'interfaccia grafica gioca un ruolo chiave, essendo il mezzo principale attraverso cui i professionisti DevOps possono interagire con il sistema per configurare, monitorare e gestire le applicazioni distribuite. Data la complessità dei task e la necessità di fornire un'esperienza utente semplice e accessibile, è stato indispensabile adottare un approccio al design che combinasse estetica e funzionalità.

L'obiettivo principale del pannello grafico di FREEDA è rendere l'interazione con il sistema intuitiva, supportando gli utenti nel prendere decisioni informate e monitorare le configurazioni in modo trasparente. Elementi come visualizzazioni grafiche delle connessioni tra microservizi, pulsanti ben definiti e strutture di navigazione coerenti sono stati progettati per migliorare l'usabilità complessiva. Inoltre è stato utilzzato l'elemento del colore sia per migliorare l'estetica e la percezione dell'interfaccia che per rendere maggiormente intuitivi gli elementi che la compongono.

# Capitolo 2

# Background

In questo capitolo viene delineato il contesto tecnologico che supporta il progetto, evidenziando come l'integrazione delle architetture IoT e Cloud IoT con metodologie di Continuous e Constraint Reasoning costituisca il fondamento della soluzione proposta. Si esaminano le principali tecnologie adottate per la gestione degli elementi e l'ottimizzazione dell'infrastruttura, concentrandosi sugli strumenti e le metodologie che hanno consentito un'efficace integrazione dei componenti e una gestione dinamica dell'ambiente applicativo.

Nel dettaglio, verranno approfonditi i sistemi IoT, che permettono la connessione e la comunicazione di dispositivi e sensori in ambienti eterogenei, e il ruolo strategico del Cloud, in grado di centralizzare l'elaborazione dei dati e garantire scalabilità e affidabilità. Parallelamente, si analizzeranno le tecniche di Continuous e Constraint Reasoning, che insieme abilitano processi decisionali rapidi e sofisticati, capaci di rispondere in tempo reale agli eventi e di risolvere complessi problemi applicativi mediante l'applicazione di vincoli predefiniti.

## 2.1 Panoramica Generale

La presente sezione offre un quadro generale delle tecnologie e delle metodologie su cui si fonda il progetto, evidenziando aspetti chiave come l'evoluzione delle architetture tecnologiche e l'approccio a metodi di ragionamento avanzato, elementi che supportano in modo flessibile i processi decisionali.

### 2.1.1 Architetture IoT e Cloud IoT

Le architetture IoT rappresentano un paradigma fondamentale per l'interconnessione di dispositivi e sensori distribuiti in ambienti estremamente eterogenei. Questi sistemi, attraverso la raccolta e la trasmissione in tempo reale dei dati, consentono un monitoraggio continuo e un'interazione intelligente con il mondo fisico, offrendo nuove prospettive di analisi e controllo in numerosi ambiti applicativi.

Un elemento cardine di queste architetture è l'integrazione con il Cloud. La centralizzazione dell'elaborazione e dell'analisi dei dati in piattaforme Cloud permette non solo di superare i limiti delle risorse locali, ma anche di garantire una scalabilità e un'affidabilità maggiori. In particolare, il Cloud favorisce la gestione di grandi volumi di dati e consente di eseguire analisi complesse in tempi ridotti, supportando così processi decisionali rapidi e informati.

L'infrastruttura di rete, caratterizzata da protocolli di comunicazione sicuri e affidabili, assicura una trasmissione protetta e tempestiva delle informazioni raccolte dai dispositivi IoT. Questa sinergia tra dispositivi, reti e infrastrutture Cloud genera un ecosistema dinamico e adattabile, capace di rispondere in maniera efficiente alle esigenze del contesto operativo e di promuovere una trasformazione digitale verso sistemi sempre più intelligenti e resilienti.

In sintesi, le architetture IoT, integrate con il Cloud, costituiscono un elemento strategico per l'innovazione tecnologica, fornendo la base necessaria per lo sviluppo di applicazioni avanzate e per l'ottimizzazione dei processi decisionali nell'era della digitalizzazione.

### 2.1.2 Continuous e Constraint Reasoning

Parallelamente alle tecnologie di raccolta e gestione dei dati, il progetto implementa metodologie di ragionamento avanzate per ottimizzare il processo decisionale. Il Continuous Reasoning si basa sull'analisi ininterrotta dei dati, permettendo di fornire risposte immediate agli eventi in tempo reale. D'altro canto, il Constraint Reasoning si focalizza sulla risoluzione di problemi complessi attraverso l'applicazione di vincoli predefiniti, contribuendo a definire soluzioni ottimali anche in scenari con numerose variabili e restrizioni. L'integrazione di questi due approcci garantisce un sistema decisionale robusto, capace di gestire sia situazioni dinamiche che problemi strutturati, favorendo una gestione efficace delle risorse e un miglioramento continuo delle prestazioni operative.

Questa panoramica introduttiva pone le basi per i capitoli successivi, nei quali verranno analizzate in maniera più dettagliata le tecnologie e le metodologie che, combinate, permettono di realizzare soluzioni innovative e altamente performanti nel contesto dei sistemi intelligenti.

## 2.2 Tecnologie Utilizzate

### 2.2.1 JavaFX

JavaFX [JavaFX] è un framework avanzato per lo sviluppo di interfacce grafiche in Java. Grazie alle sue potenti funzionalità ed alla capacità di creare interfacce moderne e responsive è stato scelto per realizzare l'interfaccia utente dell'applicazione. Quest'interfaccia guida l'utente nella configurazione della struttura della Microservices-based Applications, permettendo di definire in modo intuitivo microservizi, dipendenze e vincoli relativi alle risorse, resilienza e sostenibilità energetica.

**La scelta di JavFX** JavaFX è stato scelto come framework per lo sviluppo dell'interfaccia grafica di FREEDA è stata motivata da una serie di fattori tecnici e progettuali:

- **Flessibilità e Potenza Grafica**: JavaFX offre una vasta gamma di strumenti per creare interfacce grafiche moderne e responsive, grazie al suo potente motore grafico basato su hardware e al supporto per il rendering 2D e 3D.
- **Modularità e Scalabilità**: la sua struttura modulare lo rende adatto a progetti complessi come FREEDA, dove è necessario integrare funzionalità avanzate mantenendo un codice organizzato e manutenibile.
- **Compatibilità con Java**: essendo integrato con Java, facilita l'interazione con librerie e framework già in uso, garantendo coerenza e riutilizzo del codice.
- **Supporto per CSS e Scene Builder**: l'utilizzo di CSS per la personalizzazione grafica e Scene Builder per il drag-and-drop semplifica notevolmente lo sviluppo e accelera i tempi di implementazione.

- **Comunità e Risorse**: Una comunità attiva e una documentazione completa facilitano la risoluzione dei problemi e l'utilizzo.

**Vantaggi Specifici di JavaFX in FREEDA** Nel contesto di FREEDA, JavaFX ha permesso di implementare funzionalità avanzate, come la visualizzazione grafica delle connessioni tra i nodi e la generazione automatica di file YAML [YAML], mantenendo al contempo un'interfaccia chiara e user-friendly. La possibilità di rappresentare graficamente le dipendenze tra i microservizi, mediante elementi interattivi e personalizzabili, ha migliorato significativamente la comprensione del sistema da parte degli utenti finali.

Inoltre, l'utilizzo di JavaFX ha reso possibile integrare funzionalità dinamiche, come la visualizzazione in tempo reale delle modifiche ai parametri di configurazione e il salvataggio automatico dei dati in locale, rendendo il pannello grafico uno strumento indispensabile per la gestione delle applicazioni distribuite su infrastrutture Cloud-IoT.

### 2.2.2 YAML

Il formato YAML (YAML Ain't Markup Language) è stato scelto per rappresentare i file di configurazione grazie alla sua semplicità e leggibilità. Le strutture dati ottenute dall'interfaccia vengono convertite in file YAML conformi agli standard richiesti, facilitando l'integrazione con strumenti e piattaforme di orchestrazione su infrastrutture Cloud-IoT.

**Struttura e Sintassi dei File YAML** I file YAML si basano su una sintassi semplice ed intuitiva, che utilizza l'indentazione per definire le gerarchie dei dati, evitando l'uso di delimitatori complessi. Ad esempio, la rappresentazione di un dizionario o di una lista è immediata e visibile, come mostrato in Figura 2.1:

```yaml
configurazione:
   database:
     host: localhost
     port: 3306
     user: admin
     password: segreto

```

**Figura 2.1:** Codice di configurazione YAML

Questa struttura favorisce la leggibilità e la manutenzione, rendendo YAML particolarmente adatto per file di configurazione complessi.

**File YAML e Configurazioni** Nel contesto del progetto, i file YAML svolgono un ruolo fondamentale nel collegare l'interfaccia utente a FREEDA. I dati inseriti tramite l'interfaccia vengono automaticamente convertiti in file YAML, garantendo:

- Una rapida trasformazione dei dati in un formato standardizzato.
- L'interoperabilità con strumenti di orchestrazione e piattaforme Cloud-IoT.
- Una maggiore flessibilità, che permette di aggiornare facilmente le configurazioni senza modifiche strutturali invasive.

**SnakeYAML** La libreria *SnakeYAML* [SnakeYAML] rappresenta uno strumento fondamentale per la gestione dei file YAML all'interno del progetto. Sviluppata in linguaggio Java, essa consente di effettuare il parsing e l'emissione di file YAML in modo efficiente e conforme agli standard, facilitando l'integrazione delle configurazioni definite in YAML con il resto dell'applicazione.

### 2.2.3 MiniZinc

MiniZinc [MiniZinc07] è un linguaggio di modellazione ad alto livello, pensato per la definizione e la risoluzione di problemi di ottimizzazione. Nel progetto viene impiegato per tradurre le informazioni raccolte tramite l'interfaccia grafica in un modello matematico. Questo modello, elaborato da un solver appropriato, consente di individuare la soluzione ottimale per il deployment delle MSA.

**Struttura e Sintassi dei File MZN** I file con estensione `.mzn` contengono la definizione del modello MiniZinc e sono organizzati in modo da rendere espliciti i seguenti elementi:

- **Dichiarazione delle Variabili:** in cui vengono definiti i domini delle variabili decisionali.
- **Vincoli:** espressi in forma logica e matematica, i vincoli determinano le regole che la soluzione ottimale deve rispettare.
- **Funzione Obiettivo:** la funzione da ottimizzare, che può essere soggetta a minimizzazione o massimizzazione a seconda del problema.

**Uso di MiniZinc nel Progetto** I file `.dzn` (MiniZinc Data File Format) vengono generati automaticamente a partire dalle impostazioni definite tramite l'interfaccia utente. Il flusso operativo si articola in diverse fasi:

- **Generazione del Modello:** le configurazioni inserite dall'utente vengono tradotte in un modello matematico formalizzato in MiniZinc.
- **Esecuzione del Solver:** il file `.mzn` contenente il modello, insieme ai dati definiti nel file `.dzn`, viene processato dal solver MiniZinc. Durante questa fase, il solver applica algoritmi di risoluzione ottimizzata per individuare la soluzione che soddisfa tutti i vincoli specificati.

- **Interpretazione dei Risultati:** i risultati ottenuti vengono integrati nell'interfaccia utente, fornendo un feedback immediato e visivo sulla configurazione ottimale per il deployment delle MSA.

# Capitolo 3

# Struttura dei pannelli dell'UI

L'interfaccia grafica è organizzata in diverse aree, denominate pannelli, ognuna delle quali consente all'utente finale di configurare i parametri necessari per rappresentare il modello in maniera precisa e fedele. Complessivamente, sono stati sviluppati quattro pannelli distinti: Applicazione, Infrastruttura, Risorse e Deployment. La navigazione tra tali pannelli è resa intuitiva mediante un menù a schede posizionato nella parte superiore dell'interfaccia, che permette di selezionare agevolmente la sezione che si desidera aprire.

Questa struttura si ispira al modello adottato nel progetto FREEDA, dove i dati sono suddivisi in due gruppi principali: quelli relativi all'Applicazione e quelli relativi all'Infrastruttura.

## 3.1 Topologia Applicazione

Il pannello *Topologia Applicazione* (Figura 3.1) gestisce i dati relativi all'Applicazione. In questa sezione vengono definiti tutti i servizi che compongono il sistema, specificandone dettagliatamente le caratteristiche tecniche, i requisiti di risorse e le dipendenze reciproche.

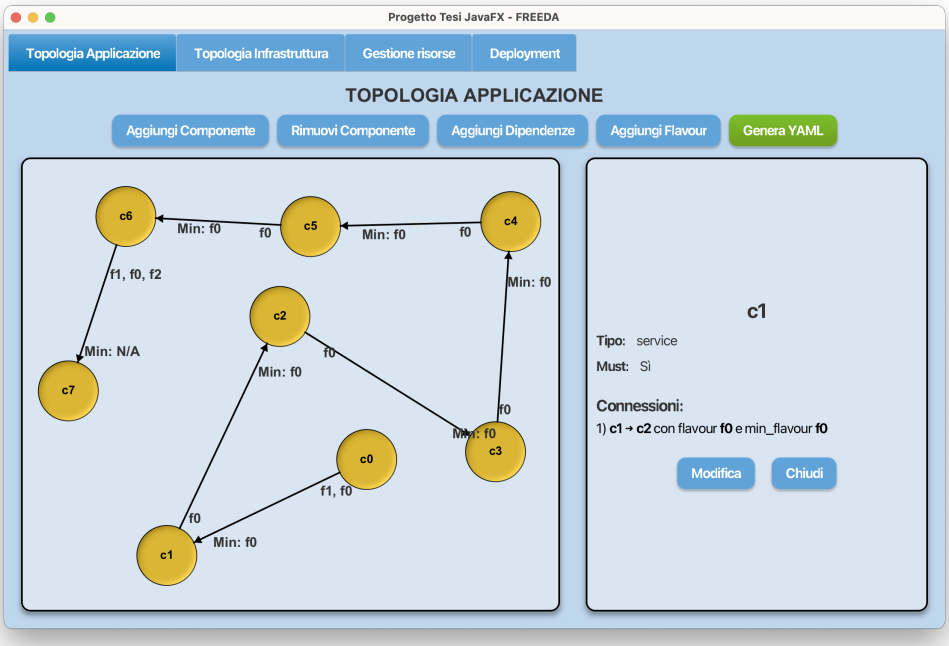


**Figura 3.1:** Pannello - Topologia Applicazione

### 3.1.1 Elementi principali

In questo pannello la parte principale è costituita da un riquadro contenente la topologia dei componenti, ovvero, la rappresentazione grafica delle configurazioni inserite fino a quel momento. La topologia è composta dai seguenti elementi:

**Componenti** Visibili come nodi gialli del grafo in Figura 3.1, sono rappresentati da cerchi colorati con all'interno il loro nome, sono gli elementi principali di questa sezione. Al click sul componente si apre una schermata a lato della topologia che riassume tutte le caratteristiche e le informazioni riguardante se stesso; inolte premendo il bottone "Modifica" è possibile cambiare i dati relativi al componente selezionato e premendo il bottone "Chiudi" si chiude la schermata dei dettagli.

**Flavour** Indicano le differenti modalità o varianti con cui un componente può essere configurato. In sostanza, i flavour rappresentano i differenti "gusti" di un componente, ciascuno dei quali introduce specifiche configurazioni e comportamenti. Ogni componente può infatti essere declinato in più flavour permettendo di personalizzare l'architettura del sistema, adattando le caratteristiche di ogni componente alle differenti esigenze funzionali e operative. Questo meccanismo consente pertanto di personalizzare e modulare l'architettura del sistema in modo da rispondere alle diverse esigenze funzionali e progettuali, garantendo flessibilità e adattabilità.

**Dipendenze** Le dipendenze definiscono le connessioni tra i componenti del sistema e possono essere suddivise in due categorie: dipendenze funzionali e dipendenze di risorse. Queste relazioni sono essenziali per garantire l'integrazione e il corretto funzionamento complessivo dell'architettura.

- **Dipendenze funzionali:** le dipendenze funzionali rappresentano le relazioni operative tra i componenti, determinando come e in quale misura un componente necessiti dell'interazione con altri per erogare le proprie funzionalità. Ad esempio, un modulo frontend potrebbe dipendere da un modulo backend per l'elaborazione delle richieste o il recupero dei dati, mentre un servizio di autenticazione potrebbe essere richiesto per abilitare l'accesso a funzionalità riservate. Que-

ste dipendenze assicurano la coerenza logica e l'efficienza operativa del sistema.

- **Dipendenze di risorse:** le dipendenze di risorse riguardano le connessioni che associano specifiche risorse ad uno o più componenti in base ai flavour configurati. Questo tipo di dipendenza consente di modellare in modo accurato il fabbisogno delle risorse e di ottimizzare la distribuzione in funzione delle differenti varianti (flavour) adottate.

**Min-flavour** Il parametro min-flavour, identificato da `Min:` in Figura 3.1 viene impiegato per definire il livello minimo (per importanza) di configurazione richiesto per una dipendenza. In altre parole, quando un componente è configurato in una determinata variante, è possibile indicare che il componente a cui fa riferimento debba essere presente almeno in una variante (flavour) minima definita.

**Connessioni Unidirezionali** Ogni componente può essere connesso con gli altri componenti presenti nella topologia e queste connessioni sono rappresentate graficamente come linee nere solide come in Figura 3.1. Inoltre le connessioni tra i componenti sono unidirezionali, ovvero, hanno un verso/direzione. L'orientamento delle connessioni è indicato da una freccia posta all'estremità della linea che identifica la direzione stabilita.

**Risorse** Le risorse rappresentano parametri fondamentali utilizzati per definire i componenti dell'applicazione e configurare l'infrastruttura. Esse costituiscono requisiti essenziali che caratterizzano i componenti e devono essere soddisfatti sia dai nodi su cui verranno implementati, sia dalle interconnessioni che li collegano.

### 3.1.2 Operazioni sui componenti

Per consentire l'inserimento dei vari elementi sopra descritti sono stati implementati i seguenti bottoni che al click aprono le relative finestre di inserimento o cancellazione degli elementi/configurazioni:

- **Aggiungi Componente:** apre una finestra in cui è possibile inserire un nuovo componente, indicando le varie configurazioni che lo compongono (nome, tipo, must, risorse assegnate, connessioni, flavour, min-flavour).

- **Rimuovi Componente:** apre una finestra in cui si sceglie tramite un elenco a discesa il componente che si vuole eliminare.

- **Aggiungi Dipendenze:** apre una finestra in cui vengono elencate tutte le connessioni già presenti, con la possibilità di aggiungere ad esse una o più dipendenze cliccando il bottone "Aggiungi Dipendenza" posto alla fine di ogni riga della tabella.

- **Aggiungi Flavour:** apre una finestra in cui è possibile aggiungere un nuovo flavour al sistema, indicando, nome, descrizione e priorità.

- **Modifica Componente:** per la modifica di ciascun componente è stato implementato un ulteriore bottone, visibile dopo la selezione del componente desiderato. Una volta cliccato, sulla destra si apre una schermata che mostra i dettagli del componente ed il bottone di modifica come in Figura 3.1. Una volta premuto questo tasto si apre una nuova finestra in cui vengono visualizzate tutte le configurazioni attuali del componente con possibilità di modificare o rimuovere parte di esse, così da adattare il componente secondo le proprie esigenze.

### 3.1.3 Generazione file configurazione dell'Applicazione

Il file di configurazione dei componenti in formato YAML [YAML] viene generato quando viene premuto il bottone "Genera YAML", con il quale si apre una schermata dove è possibile selezionare il percorso di salvataggio ed il nome del file. Una volta data conferma viene generato il file in formato `.yaml`.

## 3.2 Topologia Infrastruttura

Il pannello *Topologia Infrastruttura* (Figura 3.2) si focalizza sui dati relativi all'infrastruttura. Qui vengono descritte tutte le informazioni riguardanti i nodi su cui i servizi possono essere distribuiti, includendo dettagli come le risorse hardware disponibili, le configurazioni di rete e le relazioni tra i vari nodi tramite connessioni non orientate.

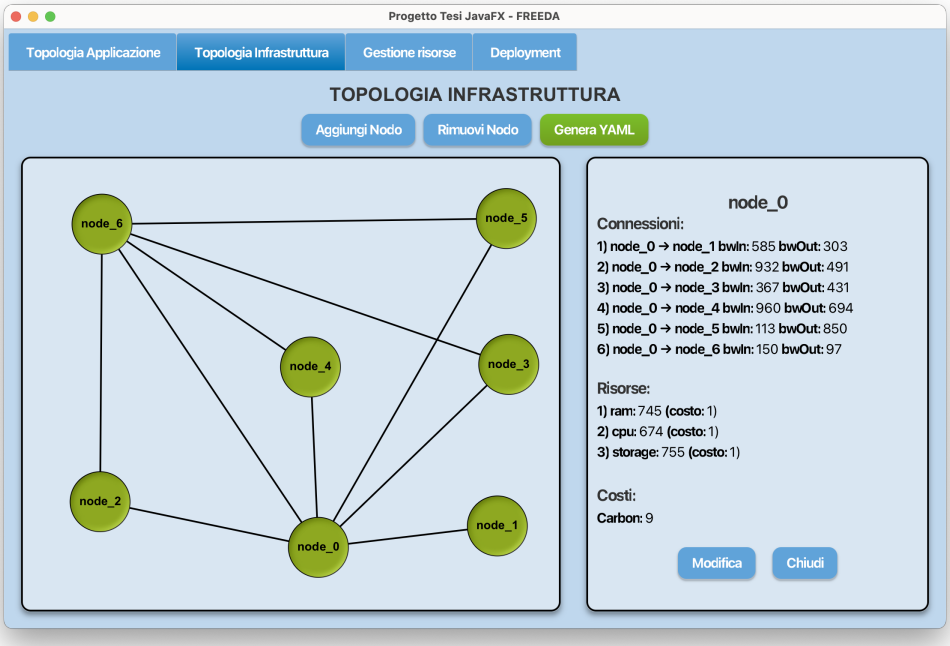


**Figura 3.2:** Pannello - Topologia Infrastruttura

### 3.2.1 Elementi principali

**Nodi** Sono rappresentati da cerchi colorati con all'interno il loro nome come in Figura 3.2, sono gli elementi principali di questa sezione. Un nodo può essere immaginato come un server, virtuale, fisico o un dispositivo IoT su cui possono essere eseguiti uno o più servizi. Ad ogni nodo sono associati dei costi per il suo utilizzo, suddivisi in due categorie principali:

- `carbon`: che rappresenta l'impatto ambientale in termini di emissioni di $CO_2$;

- `cost`: indica i costi economici.

Inoltre, i nodi possono essere connessi tra loro tramite connessioni non orientate, favorendo l'interoperabilità e la distribuzione dei servizi.

**Connessioni** Ogni nodo può essere connesso con gli altri nodi presenti nella topologia e queste connessioni sono rappresentate graficamente come linee nere solide come in Figura 3.2. Queste connessioni non sono orientate, ovvero, non hanno un verso (quindi, diversamente dai componenti, non hanno una freccia che ne identifica il senso). Le connessioni tra nodi hanno due campi:

- `bwIn`: larghezza di banda in ingresso;

- `bwOut`: larghezza di banda in uscita.

**Risorse** Le risorse rappresentano parametri fondamentali utilizzati per definire i componenti dell'applicazione e configurare l'infrastruttura. Esse costituiscono requisiti essenziali che caratterizzano i componenti e devono essere soddisfatti sia dai nodi su cui verranno implementati, sia dalle interconnessioni che li collegano.

### 3.2.2 Operazioni sui nodi

- **Aggiungi Nodo:** apre una finestra in cui è possibile compilare le varie configurazioni per creare un nuovo nodo come: nome, costo (carbon), assegnare risorse al nodo, selezionare eventuali connessioni specificando latency ed availability di ognuna di esse.

- **Rimuovi Nodo:** apre una finestra in cui è possibile selezionare, tramite un elenco a dicesa, il nodo che si desidera eliminare e successivamente confermare la cancellazione.

- **Modifica Nodo:** per la modifica di ciascun nodo è stato implementato un ulteriore bottone, visibile dopo la selezione del nodo desiderato. Una volta cliccato, sulla destra si apre una schermata che mostra i dettagli del nodo ed il bottone di modifica come in Figura 3.2. Una volta premuto questo tasto si apre una nuova finestra in cui vengono visualizzate tutte le configurazioni attuali del nodo con possibilità di modificare o rimuovere parte di esse, così da adattare il componente secondo le proprie esigenze.

### 3.2.3 Generazione file di configurazione dell'Infrastruttura

Nel momento in cui si preme il pulsante "Genera YAML", il sistema avvia la procedura di creazione del file di configurazione dell'infrastruttura in formato YAML. Contestualmente, viene visualizzata una finestra di dialogo che consente all'utente di selezionare il percorso di salvataggio e di specificare il nome del file. Una volta confermate tali impostazioni, il file in formato `.yaml` viene effettivamente generato.

## 3.3 Gestione Risorse

Il pannello *Gestione Risorse* (Figura 3.3) fa riferimento alle risorse che rappresentano parametri fondamentali utilizzati sia per definire i componenti dell'applicazione sia per configurare l'infrastruttura. Esse costituiscono requisiti essenziali che caratterizzano i componenti e che devono essere soddisfatti dai nodi su cui essi verranno implementati, nonché dalle interconnessioni che li collegano. In quest'ottica, il pannello dedicato alle risorse consente di visualizzare l'insieme delle risorse attualmente disponibili e di aggiungerne di nuove. Una volta inserite, tali risorse vengono rese accessibili anche negli altri pannelli, permettendo così la loro associazione sia ai componenti sia ai nodi e garantendo una gestione modulare e coerente del sistema.

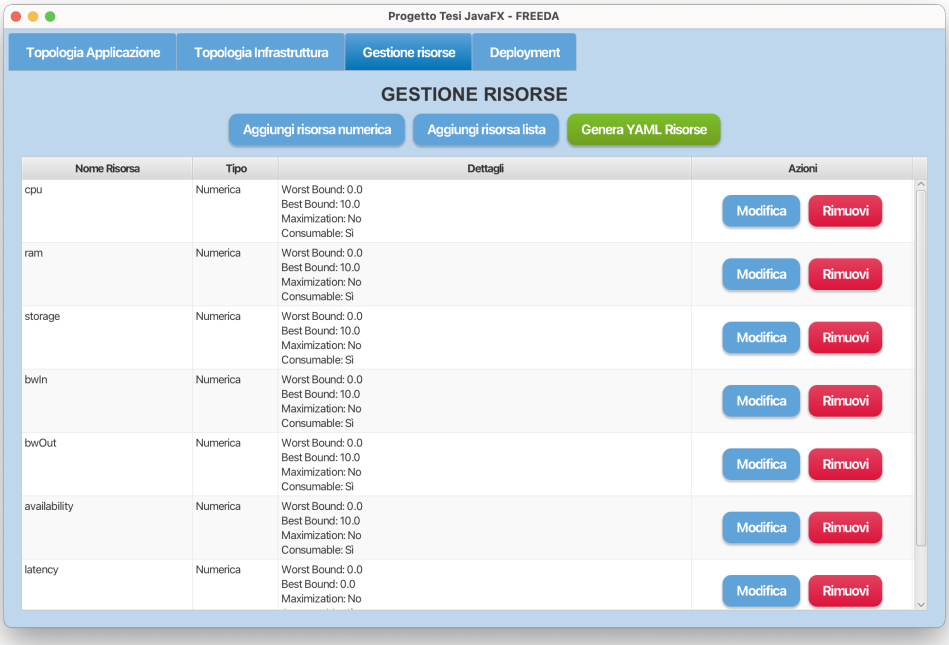


**Figura 3.3:** Pannello - Gestione Risorse

### 3.3.1 Elementi principali

Il progetto FREEDA distingue tra risorse di tipo numerico e risorse di tipo lista, differenza che verrà illustrata in dettaglio nelle sezioni successive.

**Risorsa Numerica:** queste risorse sono risorse di tipo numerico e sono dotate dei seguenti campi: nome, worst bound, best bound, flag massimizzazione, flag consumable.

**Risorsa Lista:** queste risorse sono risorse di tipo lista e sono dotate dei seguenti campi: nome, lista di valori della risorsa (valori di tipo stringa che vanno a formare la lista).

### 3.3.2 Operazioni sulle risorse

- **Inserisci risorsa numerica:** apre una finestra in cui è possibile inserire i parametri per creare una nuova risorsa di tipo numerico, ovvero: nome, worst bound, best bound, flag massimizzazione, flag consumable.

- **Inserisci risorsa lista:** apre una finestra in cui è possibile inserire i parametri per creare una nuova risorsa di tipo lista, ovvero: nome, lista dei valori associati alla risorsa lista.

- **Modifica Risorsa:** per la modifica di ciascuna risorsa è stato implementato un bottone che viene visualizzato nell'ultima colonna di ogni riga della tabella contenente le risorse inserite come in Figura 3.3. Quando questo bottone viene premuto si apre una finestra in cui vengono visualizzate tutte le configurazioni attuali della risorsa con possibilità di modificarle, rimuoverle o aggiugerle, così da adattare il nodo secondo le proprie esigenze. Il pannello cambia opportunamente se la risorsa è di tipo lista o di tipo numerico.

### 3.3.3 Generazione file di configurazione delle risorse

Premendo il pulsante "Genera YAML", viene avviato il processo di creazione del file di configurazione delle risorse in formato YAML [YAML]. Contestualmente, si apre una finestra di dialogo che consente di selezionare il percorso di salvataggio e di specificare il nome del file. Una volta confermati questi parametri, il file in formato `.yaml` viene generato.

## 3.4 Deployment

Il *Pannello Deployment* (Figura 3.4) costituisce la sezione dedicata alla fase conclusiva del processo di configurazione, in cui i componenti vengono effettivamente distribuiti (deploy) sui nodi dell'infrastruttura. Tramite questa interfaccia, l'utente può avviare il calcolo dell'allocazione ottimale, nonché visualizzare i risultati in forma sia grafica sia testuale.

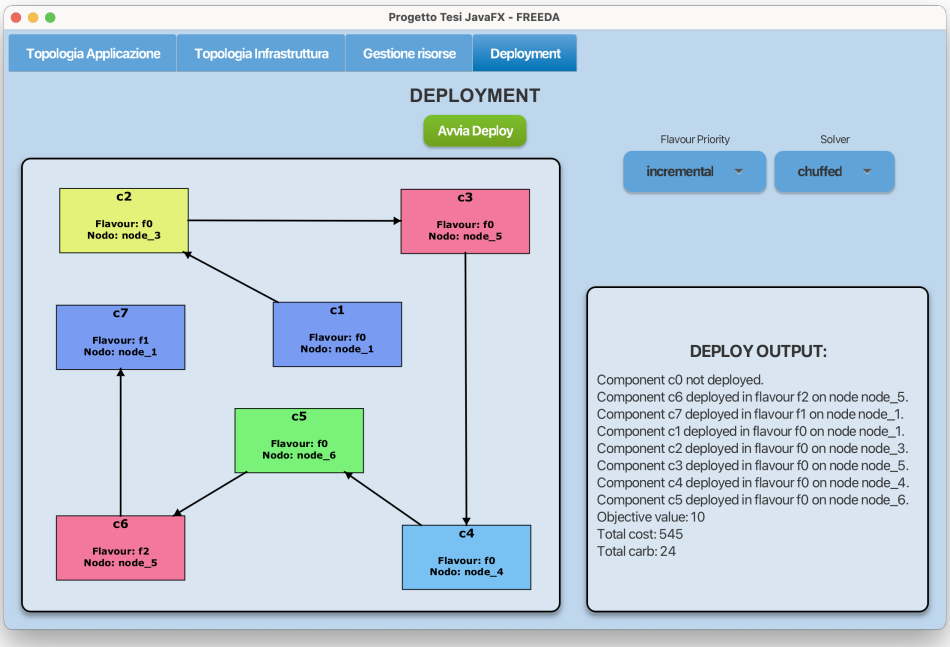


**Figura 3.4:** Pannello - Topologia Deployment

### 3.4.1 Elementi principali

All'interno del pannello è possibile individuare diversi elementi funzionali:

**Area grafica di Deployment** Occupa la parte a sinistra dello schermo e mostra un grafo in cui i componenti (rappresentati da riquadri di diverso

colore) sono associati ai nodi in cui vengono distribuiti. Le connessioni tra i riquadri indicano le dipendenze tra i componenti.

**Output testuale** Posizionato sulla destra, riporta un riepilogo dell'operazione di deploy. Vengono elencati:

- I componenti dispiegati e i rispettivi *flavour*;
- I nodi di destinazione;
- Eventuali componenti non dispiegati;
- I valori finali di metriche rilevanti, come costi e impatto ambientale (carbon).

**Pulsante di avvio (Avvia Deploy)** Collocato nella parte superiore del pannello, permette all'utente di avviare il calcolo dell'allocazione ottimale in base alle configurazioni impostate. Una volta premuto, il sistema esegue i processi di ottimizzazione e aggiorna in tempo reale sia la visualizzazione grafica sia l'output testuale.

### 3.4.2 Settaggio del flavour priority e solver

Come visibile in Figura 3.4, nella parte superiore del pannello sono presenti due *ComboBox* (elenchi a discesa) che consentono di specificare alcuni parametri fondamentali per l'operazione di deploy:

- **Flavour Priority:** permette di indicare la strategia di priorità da applicare ai diversi *flavour* dei componenti. In base alla scelta effettuata, il sistema seleziona in modo diverso le varianti di ciascun componente durante la fase di ottimizzazione. I valori selezionabili dall'elenco a tendina sono: *incremental*, *manual*, *lexicographic*, *reversed*.

- **Solver:** consente di scegliere il risolutore (solver) da utilizzare per il calcolo dell'allocazione ottimale. A seconda del solver selezionato, il sistema potrebbe adottare algoritmi differenti e fornire soluzioni più o meno rapide o accurate. I valori selezionabili dall'elenco a tendina sono: *chuffed*, *gecode*, *highs*, *lcg*.

Dopo aver impostato il *flavour priority* e il *solver*, l'utente può procedere con il pulsante di avvio per innescare l'esecuzione del processo di deploy. Una volta conclusa l'elaborazione, il sistema aggiorna sia la sezione grafica sia l'output testuale, fornendo una visione immediata della distribuzione finale dei componenti sui nodi dell'infrastruttura. I risultati includono, inoltre, eventuali metriche di costo e sostenibilità, offrendo un riscontro chiaro sugli esiti dell'ottimizzazione.

Nel caso il processo fornisca più di un risultato la rapresentazione grafica dei risultati cambia opportunamente quando il sistema trova una soluzione migliore o più performante.

# Capitolo 4

# Architettura Interna

In questo capitolo si espone in maniera approfondita il funzionamento interno del sistema, con particolare attenzione alla logica implementativa e all'interazione tra le varie componenti. L'obiettivo è quello di fornire una descrizione rigorosa delle scelte progettuali adottate e delle strategie implementative utilizzate per garantire il corretto funzionamento e la coerenza del sistema.

## 4.1 Diagramma ER

I diagrammi ER (Entity-Relationship) sono rappresentazioni grafiche che illustrano le entità presenti in un sistema e le relazioni che intercorrono tra di esse. Questi diagrammi facilitano l'analisi e la progettazione della struttura dei dati, fornendo una visione chiara delle connessioni logiche all'interno del sistema.

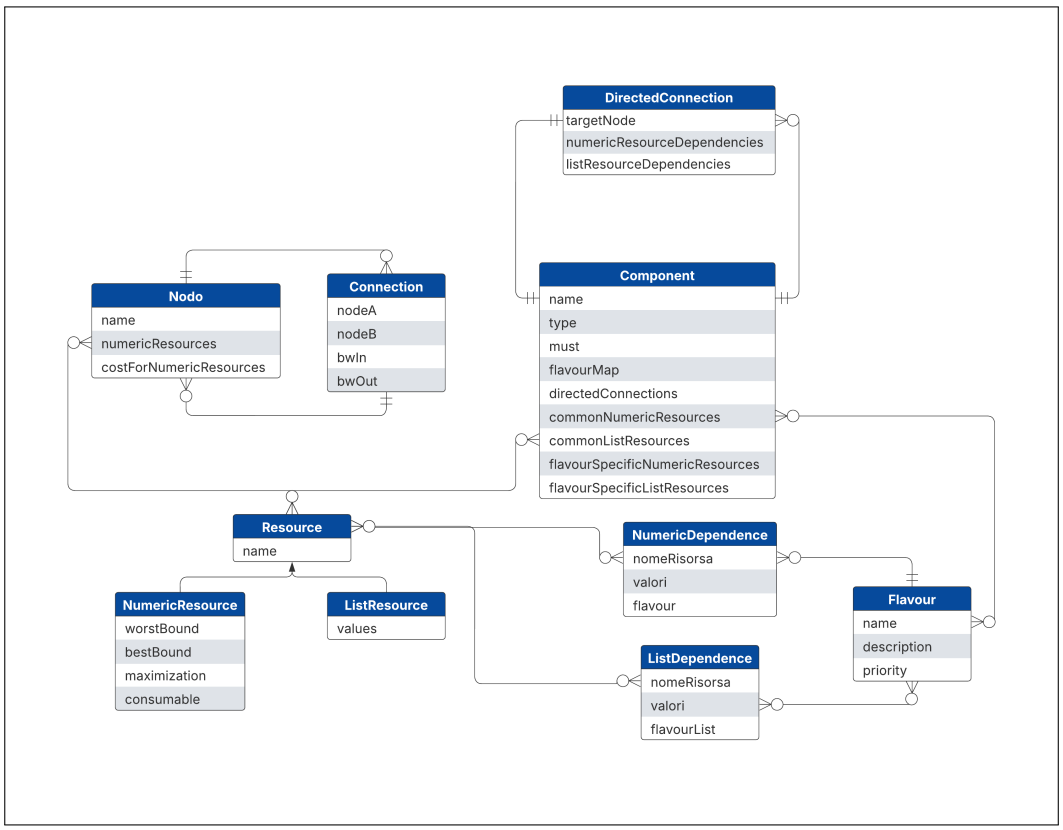


**Figura 4.1:** Diagramma ER del progetto

**Descrizione diagramma ER** In Figura 4.1 è rappresentato il diagramma ER del progetto; questo consente di mostrare come sono interconnesse le varie entità che lo compongono. Il diagramma esamina l'integrazione di elementi quali `Nodo`, `Connection`, `Component`, `DirectedConnection`, risorse e dipendenze, con particolare attenzione al concetto di "flavour". In particolare, l'entità `Nodo` rappresenta un punto di rete o un elemento di topologia, caratterizzato da un nome e dalla possibilità di disporre di risorse numeriche, alle quali è associato un costo. Le connessioni tra i nodi sono definite da `Connection`, che collega due `Nodo` (indicati come `nodeA` e `nodeB`) specificando parametri relativi alla banda in ingresso e in uscita. Il ruolo centrale è svolto da `Component`, un'entità dotata di nome, tipo e flag `must` che, attraverso l'uso delle `DirectedConnection`, si connette ad altri elementi tramite un `targetNode` e incorpora vincoli legati a risorse numeriche e a lista, indicate rispettivamente con `numericResourceDependencies` e `listResourceDependencies`. Il concetto di "flavour" viene ulteriormente definito dall'entità `Flavour`, dotata di nome, descrizione e priorità, e consente di gestire configurazioni diverse dello stesso `Component`. Infine, la gestione delle risorse è strutturata gerarchicamente a partire da `Resource`, entità generica contraddistinta dal nome, che si specializza in `NumericResource` e `ListResource`. La prima definisce attributi quali `worstBound`, `bestBound`, la possibilità di essere consumabile e la massimizzazione, mentre la seconda si occupa di risorse basate su insiemi di valori discreti. A completare il modello, `NumericDependence` e `ListDependence` formalizzano i vincoli e le relazioni logiche tra le risorse, contribuendo a definire un sistema flessibile e modulare.

## 4.2 Descrizione delle Entità

### 4.2.1 Component

La classe `Component` visibile in Codice 4.1 rappresenta un'entità fondamentale nel sistema, definendo le caratteristiche e le relazioni di un componente all'interno dell'architettura. Gli oggetti di questa classe presentano i seguenti attributi:

- `name`: una stringa che identifica il nome del componente.
- `type`: una stringa che specifica il tipo del componente.
- `must`: un valore booleano che indica se il componente è obbligatorio.
- `flavourMap`: una lista di associazioni tra istanze della classe `Flavour`.
- `directedConnections`: una lista di oggetti di tipo `DirectedConnection` che rappresentano le connessioni dirette con altri componenti.
- `commonNumericResources` e `commonListResources`: mappe che associano chiavi di tipo `String` a risorse condivise (`Resource`), rispettivamente in forma numerica e come lista di stringhe.
- `flavourSpecificNumericResources` e `flavourSpecificListResources`: strutture analoghe alle precedenti, specifiche per determinati `Flavour`.

Il costrutture inizializza gli attributi `name`, `type` e `must` e crea nuove istanze delle collezioni (`ArrayList` e `HashMap`) per garantire una corretta gestione dei dati.
Il metodo `toString()` è sovrascritto per restituire il nome del componente come rappresentazione testuale dell'oggetto.

```java
public class Component {
    private String name;
    private String type;
    private Boolean must;
    private List<Map<Flavour, Flavour>> flavourMap;
    private List<DirectedConnection> directedConnections;
    private Map<String, Map<Resource, Integer>>
    commonNumericResources;
    private Map<String, Map<Resource, List<String>>>
    commonListResources;
    private Map<String, Map<Resource, Integer>>
    flavourSpecificNumericResources;
    private Map<String, Map<Resource, List<String>>>
    flavourSpecificListResources;

    // Costrutture
    public Component(String name, String type, Boolean must)
    {
        this.name = name;
        this.type = type;
        this.must = must;
        this.directedConnections = new ArrayList<>();
        this.commonNumericResources = new HashMap<>();
        this.commonListResources = new HashMap<>();
        this.flavourSpecificNumericResources = new HashMap
    <>();
        this.flavourSpecificListResources = new HashMap<>();
    }

    @Override
    public String toString(){
        return name;
    }
}
```

**Codice 4.1:** Codice classe Component.java

### 4.2.2 DirectedConnection

La classe `DirectedConnection` visibile in Codice 4.2 modella una connessione diretta tra componenti, indicando le dipendenze che si instaurano verso un componente specifico. Gli oggetti di questa classe presentano i seguenti attributi:

- `targetNode`: istanza della classe `Component` che rappresenta il nodo di destinazione.
- `numericResourceDependencies`: lista di oggetti `NumericDependence` che memorizza le dipendenze numeriche associate.
- `listResourceDependencies`: lista di oggetti `ListDependence` che contiene le dipendenze di tipo lista.

Il costruttore riceve un parametro di tipo `Component` e lo assegna a `targetNode`. Il metodo `removeNumericDependence` rimuove una dipendenza numerica dalla lista. Il metodo `removeListDependence`, invece, rimuove una dipendenza di tipo lista dalla lista.

```java
public class DirectedConnection {
    private Component targetNode;
    private List<NumericDependence>
    numericResourceDependencies = new ArrayList<
    NumericDependence>();
    private List<ListDependence> listResourceDependencies =
    new ArrayList<ListDependence>();

    // Costruttore
    public DirectedConnection(Component targetNode) {
        this.targetNode = targetNode;
    }

    public void removeNumericDependence(NumericDependence
    dependence) {
        numericResourceDependencies.remove(dependence);
    }

    public void removeListDependence(ListDependence
    dependence) {
        listResourceDependencies.remove(dependence);
    }
}
```

**Codice 4.2:** Codice classe DirectedConnection.java

### 4.2.3 Flavour

La classe `Flavour` visibile in Codice 4.3 definisce una particolare tipologia o variante di componente, utile per differenziare configurazioni e comportamenti. Gli oggetti di questa classe presentano i seguenti attributi:

- `name`: nome della variante.
- `description`: descrizione dettagliata della variante.
- `priority`: intero che determina il livello di priorità.

Il costruttore inizializza gli attributi con i valori forniti in input. Il metodo `toString()` restituisce il nome della variante.

```java
public class Flavour {
    private String name;
    private String description;
    private int priority;

    // Costruttore
    public Flavour(String name, String description, int
    priority) {
        this.name = name;
        this.description = description;
        this.priority = priority;
    }

    @Override
    public String toString() {
        return name;
    }
}
```

**Codice 4.3:** Codice classe Flavour.java

### 4.2.4 NumericDependence

La classe `NumericDependence` visibile in Codice 4.4 rappresenta una dipendenza numerica associata a una risorsa, includendo un valore numerico e una caratterizzazione tramite un oggetto `Flavour`. Gli oggetti di questa classe presentano i seguenti attributi:

- `nomeRisorsa`: stringa che identifica il nome della risorsa.
- `valore`: intero che rappresenta il valore della dipendenza.
- `flavour`: istanza di `Flavour` che attribuisce una specifica caratteristica.

Il costruttore inizializza gli attributi `nomeRisorsa`, `valore` e `flavour`.

```
1  public class NumericDependence {
2      private String nomeRisorsa;
3      private int valore;
4      private Flavour flavour;
5
6      // Costruttore
7      public NumericDependence(String nomeRisorsa, int valore,
       Flavour flavour) {
8          this.nomeRisorsa = nomeRisorsa;
9          this.valore = valore;
10         this.flavour = flavour;
11     }
12 }
```

**Codice 4.4:** Codice classe NumericDependence.java

### 4.2.5 ListDependence

La classe `ListDependence` visibile in Codice 4.5 modella una dipendenza basata su liste, associando a una risorsa un insieme di valori testuali e varianti. Gli oggetti di questa classe presentano i seguenti attributi:

- `nomeRisorsa`: stringa che identifica il nome della risorsa.
- `valori`: lista di stringhe contenente i valori associati.
- `flavourList`: lista di oggetti `Flavour` che specifica le varianti correlate.

Il costruttore inizializza gli attributi con i valori forniti.

```java
public class ListDependence {
    private String nomeRisorsa;
    private List<String> valori;
    private List<Flavour> flavourList;

    // Costruttore
    public ListDependence(String nomeRisorsa, List<String>
   valori, List<Flavour> flavourList) {
        this.nomeRisorsa = nomeRisorsa;
        this.valori = valori;
        this.flavourList = flavourList;
    }
}
```

**Codice 4.5:** Codice classe ListDependence.java

### 4.2.6 Resource

La classe astratta `Resource` visibile in Codice 4.6 rappresenta l'astrazione di una risorsa nel sistema, definendo l'attributo comune `name`.

```java
public abstract class Resource {
    private String name;

    // Costruttore
    public Resource(String name) {
        this.name = name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

**Codice 4.6:** Codice classe Resource.java

### 4.2.7 NumericResource

La classe `NumericResource` visibile in Codice 4.7 estende `Resource` per rappresentare risorse di natura numerica. Gli oggetti di questa classe presentano i seguenti attributi:

- `worstBound`: limite minimo (peggiore) che la risorsa può assumere.
- `bestBound`: limite massimo (migliore) raggiungibile dalla risorsa.
- `maximization`: flag booleano che indica se la risorsa deve essere massimizzata.
- `consumable`: flag booleano che specifica se la risorsa è consumabile.

Il costruttore inizializza gli attributi con i valori forniti.
Il metodo `toString()` fornisce una rappresentazione testuale completa dell'oggetto.

```java
public class NumericResource extends Resource {
    private Double worstBound;
    private Double bestBound;
    private boolean maximization;
    private boolean consumable;

    // Costruttore
    public NumericResource(String name, double worstBound,
    double bestBound, boolean maximization, boolean consumable
    ) {
        super(name);
        this.worstBound = worstBound;
        this.bestBound = bestBound;
        this.maximization = maximization;
        this.consumable = consumable;
    }

    @Override
    public String toString() {
        return "NumericResource{" +
                "name='" + getName() + '\'' +
                ", worstBound=" + worstBound +
                ", bestBound=" + bestBound +
                ", maximization=" + maximization +
                ", consumable=" + consumable +
                '}';
    }
}
```

**Codice 4.7:** Codice classe NumericResource.java

### 4.2.8 ListResource

La classe `ListResource` visibile in Codice 4.8 estende `Resource` ed è progettata per gestire risorse costituite da una lista di valori. Gli oggetti di questa classe presentano un unico attrubuto `values` che identifica una lista di stringhe contenente i possibili valori associati alla risorsa.

Il costruttore inizializza il nome della risorsa e la lista dei valori.
Il metodo `toString()` fornisce una rappresentazione sintetica e facilmente interpretabile della risorsa.

```java
public class ListResource extends Resource {

    private List<String> values;

    // Costruttore
    public ListResource(String name) {
        super(name);
        this.values = new ArrayList<>();
    }

    @Override
    public String toString() {
        return "ListResource{name='" + getName() + "', values
    =" + values + "}";
    }
}
```

**Codice 4.8:** Codice classe ListResource.java

### 4.2.9 Nodo

La classe `Nodo` visibile in Codice 4.9 rappresenta un'entità fondamentale per la modellazione dell'infrastruttura. Essa incapsula le informazioni relative a un nodo, comprendendo il suo nome, le risorse associate e le connessioni con altri nodi. Gli oggetti di questa classe presentano i seguenti attributi:

- `name`: nome del nodo.
- `numericResources`: mappa delle risorse numeriche.
- `costForNumericResources`: mappa dei costi associati alle risorse numeriche.
- `listResources`: lista delle risorse di tipo lista.
- `costForListValues`: mappa dei costi per i valori delle risorse di tipo lista.
- `carbon`: parametro per il monitoraggio dell'impatto ambientale.
- `connections`: lista dei nodi connessi.

Il costruttore inizializza il nome del nodo e le strutture dati per la gestione delle risorse e delle connessioni.

```java
public class Nodo {

    private String name;
    private final Map<NumericResource, Integer>
    numericResources;
    private final Map<NumericResource, Integer>
    costForNumericResources;
    private final List<ListResource> listResources;
    private final Map<ListResource, Map<String, Integer>>
    costForListValues;
    private int carbon = 0;
    private final List<Nodo> connections;

    // Costruttore
    public Nodo(String name) {
        this.name = name;
        this.numericResources = new HashMap<>();
        this.costForNumericResources = new HashMap<>();
        this.listResources = new ArrayList<>();
        this.costForListValues = new HashMap<>();
        this.connections = new ArrayList<>();
    }
}
```

**Codice 4.9:** Codice classe Nodo.java

### 4.2.10 Connection

La classe `Connection` visibile in Codice 4.10 rappresenta il collegamento tra due nodi dell'infrastruttura, modellando la connessione tra due istanze della classe `Nodo`. Gli oggetti di questa classe presentano i seguenti attributi:

- `nodeA`: il primo nodo della connessione.
- `nodeB`: il secondo nodo della connessione.
- `bwIn`: larghezza di banda in ingresso.

- `bwOut`: larghezza di banda in uscita.

Il costruttore Inizializza la connessione assegnando i nodi `nodeA` e `nodeB` e impostando i valori di larghezza di banda.

```
1  public class Connection {
2      private Nodo nodeA;
3      private Nodo nodeB;
4      private int bwIn;
5      private int bwOut;
6
7      // Costruttore
8      public Connection(Nodo nodeA, Nodo nodeB, int bwIn, int
       bwOut) {
9          this.nodeA = nodeA;
10         this.nodeB = nodeB;
11         this.bwIn = bwIn;
12         this.bwOut = bwOut;
13     }
14 }
```

**Codice 4.10:** Codice classe Connection.java

## 4.3 Implementazione del Deployment

Quando l'utente preme il bottone `Avvia Deploy` nel pannello in Figura 3.4 si innesca un processo articolato in più fasi, finalizzato alla configurazione, esecuzione e visualizzazione del deployment. Di seguito viene descritto in dettaglio il procedimento.

### 4.3.1 Creazione dei file di configurazione YAML

Il sistema raccoglie le informazioni inserite dall'utente relative ai componenti e all'infrastruttura e le organizza in due file di configurazione: `components.yaml` e `infrastructure.yaml`. Questi file vengono generati e salvati nella directory temporanea del sistema, ottenuta tramite il comando Java:

```
System.getProperty("java.io.tmpdir")
```

In questo modo il processo risulta compatibile con tutti i sistemi operativi.

### 4.3.2 Esecuzione del Solver in Python

Dopo la creazione dei file YAML [YAML], il sistema esegue uno script Python che trasforma tali configurazioni in un file dati in formato `.dzn` (in questo caso, denominato `test.dzn`). Il comando eseguito è il seguente:

```
python3 /percorso-del-file/main.py
/percorso-del-file/components.yaml
/percorso-del-file/infrastructure.yaml -p <valore-tendina>
```

Descrizione del comando:

- I parametri `/percorso-del-file/components.yaml` e `/percorso-del-file/infrastructure.yaml` indicano i file di configurazione appena creati.

- L'opzione `-p <valore-tendina>`, visibile in Figura 3.4, consente di passare il valore selezionato nella tendina relativa al `Flavour Priority`.

L'esecuzione dello script Python genera il file `test.dzn` nella cartella temporanea, che contiene i dati strutturati necessari per la fase di ottimizzazione.

### 4.3.3 Esecuzione del comando MiniZinc

Con il file `test.dzn` disponibile, il sistema costruisce ed esegue il comando per avviare MiniZinc [MiniZinc07]:

```
minizinc /percorso-del-file/freeda-model-v0.2.mzn -d
/percorso-del-file/data.dzn --solver <valore-tendina>
```

Descrizione del comando:

- `/percorso-del-file/freeda-model-v0.2.mzn` è il modello matematico del deployment, copiato nella cartella temporanea.

- L'opzione `-d /percorso-del-file/data.dzn` (che corrisponde al file `test.dzn`) fornisce i dati necessari.

- L'opzione `--solver <valore-tendina>`, visibile in Figura 3.4 consente di selezionare il solver, come specificato dall'utente.

MiniZinc esegue il modello, trovando inizialmente una prima soluzione iniziale. Successivamente, verifica la presenza di eventuali soluzioni migliori e, se ne trova, le calcola e le restituisce in output. In questo modo il sistema è in grado di gestire e presentare più soluzioni, la cui ultima, al termine del calcolo, è quella ottimale.

### 4.3.4 Elaborazione e Visualizzazione dei Risultati

Una volta eseguito il comando precedente, l'output viene catturato e suddiviso in blocchi, ciascuno corrispondente a una soluzione. Il processo di parsing utilizza linee delimitatrici (una sequenza di trattini) per identificare i confini tra le soluzioni. Successivamente, viene implementato un meccanismo basato su una timeline che alterna la visualizzazione delle diverse soluzioni, aggiornando dinamicamente la topologia di deployment rappresentata graficamente. In questo modo, l'utente può esaminare e confrontare le soluzioni ottimali, mentre il sistema aggiorna le strutture dati interne per riflettere la mappatura tra componenti, nodi e flavour.

In sintesi, il processo di deployment integra:

1. La generazione di file YAML e la loro memorizzazione nella directory temporanea.
2. L'esecuzione di uno script Python che produce il file dati `test.dzn`.
3. L'invocazione di MiniZinc, che risolve il modello matematico e restituisce una o più soluzioni ottimali.
4. L'elaborazione e la visualizzazione dinamica dei risultati, che consente di gestire il caso in cui siano presenti più soluzioni ottimali.

# Capitolo 5

# Conclusioni

## 5.1 Conclusioni Finali e Contributi della Tesi

Il lavoro svolto ha permesso di raggiungere i seguenti risultati:

- **Implementazione struttura**: implementazione della struttura dati per gestire le configurazioni richieste dal progetto FREEDA.
- **Sviluppo della GUI**: creazione di una GUI intuitiva e user-friendly che faciliti l'inserimento e la gestione delle configurazioni necessarie al deploy di MSA e alla configurazione di sistemi complessi, riducendo la curva di apprendimento per gli operatori.
- **Modularità e Scalabilità**: progettazione di un'architettura modulare che consenta di estendere facilmente il sistema con nuove funzionalità e tipologie di risorse, garantendo coerenza nell'interfaccia e nella gestione dei dati.
- **Automazione e Integrazione**: realizzazione di un flusso di lavoro automatizzato per la generazione di file di configurazione in formato YAML [YAML].

- **Rielaborazione soluzioni**: visualizzazione migliorata delle soluzioni ottenute dall'esecuzione del modello FREEDA, del solver e dei file di configurazione, per renderle più leggibili ed intuitive.

## 5.2 Limiti e Sviluppi Futuri

Nonostante i risultati ottenuti, il progetto presenta alcuni limiti che offrono interessanti spunti per futuri sviluppi:

- **Aggiunta e Integrazione di un Database:** valutare l'introduzione di un database per la gestione centralizzata dei dati. Tale sviluppo consentirebbe un migliore tracciamento e analisi delle informazioni, facilitando il recupero dei dati in tempo reale oltre a migliorare la scalabilità e la sicurezza del sistema.

- **Miglioramento dell'Usabilità:** svolgere approfonditi test con utenti finali per identificare ulteriori aree di miglioramento nell'interfaccia grafica e nei flussi interattivi.

- **Integrazione con Soluzioni Esterne:** ampliare la compatibilità con altri sistemi e piattaforme di orchestrazione, garantendo una maggiore interoperabilità e un'integrazione più fluida in ambienti eterogenei.

# Bibliografia


[SnakeYAML] Andrey Asomov. *SnakeYAML - YAML parser for Java*. 2025. URL: `https://github.com/snakeyaml/snakeyaml`.

[JavaFX] *JavaFX Official Documentation*. URL: `https://openjfx.io/`.

[MiniZinc07] Nicholas Nethercote et al. "MiniZinc: Towards a Standard CP Modelling Language". In: *Principles and Practice of Constraint Programming – CP 2007*. A cura di Christian Bessière. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 529–543. ISBN: 978-3-540-74970-7.

[YAML] *YAML Official Website*. URL: `https://yaml.org/`.

# Appendice A

# File FXML

## A.1 Main FXML

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<?import java.net.URL?>
<AnchorPane xmlns:fx="http://javafx.com/fxml" fx:controller="
    org.progettotesi.controller.MainController">
     <stylesheets>
         <URL value="@css/FREEDA_style.css"/>
     </stylesheets>
     <TabPane fx:id="tabPane" AnchorPane.topAnchor="0"
    AnchorPane.bottomAnchor="0" AnchorPane.leftAnchor="0"
    AnchorPane.rightAnchor="0">
         <tabs>
             <Tab fx:id="tabUses" text="Topologia Applicazione
    " closable="false"/>
             <Tab fx:id="tabInfrastructure" text="Topologia
    Infrastruttura" closable="false"/>
```

```
15              <Tab fx:id="tabResources" text="Gestione risorse"
     closable="false"/>
16              <Tab fx:id="tabDeploy" text="Deployment" closable
    ="false"/>
17          </tabs>
18      </TabPane>
19  </AnchorPane>
```

## A.2 Pannello Applicazione

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.scene.control.*?>
4  <?import javafx.scene.layout.*?>
5
6  <?import java.net.URL?>
7  <AnchorPane xmlns:fx="http://javafx.com/fxml" fx:controller="
     org.progettotesi.controller.UsesPaneController">
8      <stylesheets>
9          <URL value="@css/FREEDA_style.css"/>
10     </stylesheets>
11     <children>
12         <HBox alignment="CENTER"
13               AnchorPane.topAnchor="14.0"
14               AnchorPane.leftAnchor="0.0"
15               AnchorPane.rightAnchor="0.0">
16             <children>
17                 <Label text="TOPOLOGIA APPLICAZIONE" style="-
     fx-font-size: 22px; -fx-font-weight: bold; -fx-font-
     family: 'Arial';"/>
18             </children>
19         </HBox>
20
```

```
21        <!-- Barra dei bottoni principali, ancorata sotto il titolo e espandibile in orizzontale -->
22        <VBox alignment="CENTER" spacing="10"
23              AnchorPane.topAnchor="50.0"
24              AnchorPane.leftAnchor="20.0"
25              AnchorPane.rightAnchor="20.0">
26            <children>
27                <HBox spacing="10" alignment="CENTER">
28                    <children>
29                        <Button fx:id="bottoneAggiungiComponente" text="Aggiungi Componente" />
30                        <Button fx:id="bottoneRimuoviComponente" text="Rimuovi Componente" />
31                        <Button fx:id="bottoneAggiungiDipendenza" text="Aggiungi Dipendenze" />
32                        <Button fx:id="bottoneAggiungiFlavour" text="Aggiungi Flavour" />
33                        <Button fx:id="bottoneGeneraYaml" text="Genera YAML" onAction="#handleGeneraYaml" />
34                    </children>
35                </HBox>
36                <Region fx:id="spacer" />
37            </children>
38        </VBox>
39
40        <!-- Pannello centrale per la visualizzazione della topologia -->
41        <Pane fx:id="pane" style="-fx-border-color: black; -fx-border-width: 2px;"
42              AnchorPane.topAnchor="100.0"
43              AnchorPane.leftAnchor="20.0"
44              AnchorPane.bottomAnchor="20.0"
45              AnchorPane.rightAnchor="450.0"
46              prefWidth="450"
47              prefHeight="300"/>
```

```
48
49          <!-- Pannello laterale per i dettagli del componente
    -->
50          <VBox fx:id="dettagliComponentePane" visible="false"
51                style="-fx-border-color: black; -fx-padding:
    10;"
52                alignment="CENTER"
53                AnchorPane.topAnchor="100.0"
54                AnchorPane.rightAnchor="20.0"
55                AnchorPane.bottomAnchor="20.0"
56                prefWidth="400.0">
57              <children>
58                  <Label fx:id="labelNomeComponente" style="-fx
    -font-size: 16px; -fx-font-weight: bold;" />
59                  <VBox fx:id="flavoursList" spacing="5" />
60                  <VBox fx:id="connectionsList" spacing="5" />
61                  <HBox spacing="20" alignment="CENTER">
62                      <children>
63                          <Button fx:id="
    bottoneModificaComponente" text="Modifica" />
64                          <Button fx:id="
    bottoneChiudiDettaglioComponente" text="Chiudi" />
65                      </children>
66                  </HBox>
67              </children>
68          </VBox>
69      </children>
70  </AnchorPane>
```

## A.3 Pannello Infrastruttura

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.scene.control.*?>
```

```
<?import javafx.scene.layout.*?>

<?import java.net.URL?>
<AnchorPane xmlns:fx="http://javafx.com/fxml" fx:controller="
    org.progettotesi.controller.InfrastructurePaneController">
     <stylesheets>
         <URL value="@css/FREEDA_style.css"/>
     </stylesheets>
     <children>
         <HBox alignment="CENTER"
               AnchorPane.topAnchor="14.0"
               AnchorPane.leftAnchor="0.0"
               AnchorPane.rightAnchor="0.0">
             <children>
                 <Label text="TOPOLOGIA INFRASTRUTTURA"
                        style="-fx-font-size: 22px; -fx-font-
    weight: bold; -fx-font-family: 'Arial';"/>
             </children>
         </HBox>

         <VBox alignment="CENTER" spacing="10"
               AnchorPane.topAnchor="50.0"
               AnchorPane.leftAnchor="20.0"
               AnchorPane.rightAnchor="20.0">
             <children>
                 <HBox spacing="10" alignment="CENTER">
                     <children>
                         <Button fx:id="bottoneAggiungiNodo"
    text="Aggiungi Nodo" />
                         <Button fx:id="bottoneRimuoviNodo"
    text="Rimuovi Nodo" />
                         <Button fx:id="bottoneGeneraYaml"
    text="Genera YAML" />
                     </children>
                 </HBox>
```

```
34                <Region fx:id="spacer" />
35            </children>
36        </VBox>
37
38        <!-- Pannello centrale per la visualizzazione della
    topologia -->
39        <Pane fx:id="pane" style="-fx-border-color: black; -
    fx-border-width: 2px;"
40              AnchorPane.topAnchor="100.0"
41              AnchorPane.leftAnchor="20.0"
42              AnchorPane.bottomAnchor="20.0"
43              AnchorPane.rightAnchor="450.0" />
44
45        <!-- Pannello laterale per i dettagli del nodo -->
46        <VBox fx:id="dettagliNodoPane" visible="false"
47              alignment="CENTER"
48              AnchorPane.topAnchor="100.0"
49              AnchorPane.rightAnchor="20.0"
50              AnchorPane.bottomAnchor="20.0"
51              prefWidth="400.0">
52            <children>
53                <Label fx:id="nomeNodoLabel" style="-fx-font-
    size: 16pt; -fx-font-weight: bold;" />
54                <VBox fx:id="connectionsList" spacing="5" />
55                <VBox fx:id="resourcesList" spacing="5" />
56                <HBox spacing="20" alignment="CENTER">
57                    <children>
58                        <Button fx:id="modificaNodoButton"
    text="Modifica" />
59                        <Button fx:id="
    chiudiDettagliNodoButton" text="Chiudi" />
60                    </children>
61                </HBox>
62            </children>
63        </VBox>
```

```
64      </children>
65  </AnchorPane>
```

## A.4 Pannello Risorse

```
1   <?xml version="1.0" encoding="UTF-8"?>
2
3   <?import javafx.collections.FXCollections?>
4   <?import javafx.scene.control.Button?>
5   <?import javafx.scene.control.CheckBox?>
6   <?import javafx.scene.control.ComboBox?>
7   <?import javafx.scene.control.Label?>
8   <?import javafx.scene.control.TableColumn?>
9   <?import javafx.scene.control.TableView?>
10  <?import javafx.scene.control.TextField?>
11  <?import javafx.scene.layout.AnchorPane?>
12  <?import javafx.scene.layout.HBox?>
13  <?import javafx.scene.layout.VBox?>
14
15  <?import java.lang.String?>
16  <AnchorPane xmlns="http://javafx.com/javafx"
17              xmlns:fx="http://javafx.com/fxml"
18              fx:controller="org.progettotesi.controller.
    ResourcesPaneController"
19              prefHeight="600.0" prefWidth="800.0">
20      <children>
21
22          <HBox alignment="CENTER"
23                AnchorPane.topAnchor="14.0"
24                AnchorPane.leftAnchor="0.0"
25                AnchorPane.rightAnchor="0.0">
26              <children>
27                  <Label text="GESTIONE RISORSE" style="-fx-
    font-size: 22px; -fx-font-weight: bold; -fx-font-family: '
```

```
Arial';"/>
28              </children>
29          </HBox>
30
31          <VBox alignment="CENTER" spacing="10"
32                AnchorPane.topAnchor="50.0"
33                AnchorPane.leftAnchor="20.0"
34                AnchorPane.rightAnchor="20.0">
35              <children>
36                  <HBox alignment="CENTER" spacing="10">
37                      <children>
38                          <Button fx:id="
bottoneAggiungiRisorsaNumerica" text="Aggiungi risorsa
numerica" />
39                          <Button fx:id="
bottoneAggiungiRisorsaLista" text="Aggiungi risorsa lista"
 />
40                          <Button fx:id="generateYamlButton"
text="Genera YAML Risorse" />
41                      </children>
42                  </HBox>
43              </children>
44          </VBox>
45
46          <VBox fx:id="numericResourceConfig" spacing="10"
47                AnchorPane.topAnchor="20.0"
48                AnchorPane.leftAnchor="250.0"
49                visible="false">
50              <children>
51                  <Label text="Configura Risorsa Numerica"
style="-fx-font-size: 14px; -fx-font-weight: bold;" />
52                  <HBox spacing="10">
53                      <children>
54                          <Label text="Nome:" />
```

```
55                      <TextField fx:id="
    numericResourceNameField" promptText="Inserisci il nome" /
    >
56                  </children>
57              </HBox>
58              <HBox spacing="10">
59                  <children>
60                      <Label text="Worst Bound:" />
61                      <TextField fx:id="
    numericWorstBoundField" promptText="Inserisci il Worst
    Bound" />
62                  </children>
63              </HBox>
64              <HBox spacing="10">
65                  <children>
66                      <Label text="Best Bound:" />
67                      <TextField fx:id="
    numericBestBoundField" promptText="Inserisci il Best Bound
    " />
68                  </children>
69              </HBox>
70              <HBox spacing="10">
71                  <children>
72                      <Label text="Tipo:" />
73                      <ComboBox fx:id="numericTypeComboBox"
    >
74                          <items>
75                              <FXCollections fx:factory="
    observableArrayList">
76                                  <String fx:value="
    Massimizzazione" />
77                                  <String fx:value="
    Minimizzazione" />
78                              </FXCollections>
79                          </items>
```

```
80                          </ComboBox>
81                      </children>
82                  </HBox>
83                  <HBox spacing="10">
84                      <children>
85                          <Label text="Consumable:" />
86                          <CheckBox fx:id="
    numericConsumableCheckBox" />
87                      </children>
88                  </HBox>
89                  <Button fx:id="saveNumericResourceButton"
    text="Salva Risorsa" />
90              </children>
91          </VBox>
92
93          <VBox fx:id="listResourceConfig" spacing="10"
94                AnchorPane.topAnchor="250.0"
95                AnchorPane.leftAnchor="250.0"
96                visible="false">
97              <children>
98                  <Label text="Configura Risorsa di Tipo Lista"
     style="-fx-font-size: 14px; -fx-font-weight: bold;" />
99                  <HBox spacing="10">
100                     <children>
101                         <Label text="Nome:" />
102                         <TextField fx:id="
    listResourceNameField" promptText="Inserisci il nome" />
103                     </children>
104                 </HBox>
105                 <Label text="Valori della Lista:" />
106                 <VBox fx:id="listValuesContainer" spacing="5"
    >
107                     <children>
108                         <HBox spacing="10">
109                             <children>
```

```
110                                 <TextField fx:id="
    newListValueField" promptText="Inserisci un valore" />
111                                 <Button fx:id="
    addListValueButton" text="Aggiungi Valore" />
112                             </children>
113                         </HBox>
114                     </children>
115                 </VBox>
116                 <Button fx:id="saveListResourceButton" text="
    Salva Risorsa" />
117             </children>
118         </VBox>
119
120         <TableView fx:id="resourcesTable"
121                    AnchorPane.topAnchor="100.0"
122                    AnchorPane.leftAnchor="20.0"
123                    AnchorPane.rightAnchor="20.0"
124                    AnchorPane.bottomAnchor="20.0">
125             <columns>
126                 <TableColumn fx:id="resourceNameColumn" text=
    "Nome Risorsa" prefWidth="200" />
127                 <TableColumn fx:id="resourceTypeColumn" text=
    "Tipo" prefWidth="100" />
128                 <TableColumn fx:id="resourceDetailsColumn"
    text="Dettagli" prefWidth="460" />
129                 <TableColumn fx:id="resourceActionsColumn"
    text="Azioni" prefWidth="200" />
130             </columns>
131         </TableView>
132     </children>
133 </AnchorPane>
```

## A.5 Pannello Deploy

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.collections.FXCollections?>

<?import java.net.URL?>
<AnchorPane xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml" fx:controller="org.progettotesi.controller.DeployPaneController" prefHeight="400.0" prefWidth="600.0">
    <stylesheets>
        <URL value="@css/FREEDA_style.css"/>
    </stylesheets>
    <children>
        <!-- Header -->
        <HBox alignment="CENTER"
              AnchorPane.topAnchor="14.0"
              AnchorPane.leftAnchor="0.0"
              AnchorPane.rightAnchor="0.0">
            <children>
                <Label text="DEPLOYMENT" style="-fx-font-size: 22px; -fx-font-weight: bold; -fx-font-family: 'Arial';"/>
            </children>
        </HBox>

        <!-- Contenitore per il bottone Avvia Deploy (a sinistra) -->
        <VBox alignment="CENTER" spacing="10"
```

```
28              AnchorPane.topAnchor="50.0"
29              AnchorPane.leftAnchor="20.0"
30              AnchorPane.rightAnchor="20.0">
31          <children>
32              <HBox spacing="10" alignment="CENTER">
33                  <children>
34                      <!-- Bottone di avvio deploy -->
35                      <Button fx:id="bottoneAvviaDeploy"
    text="Avvia Deploy"/>
36                  </children>
37              </HBox>
38              <!-- Spacer se necessario -->
39              <Region fx:id="spacer" />
40          </children>
41      </VBox>
42
43      <!-- Pannello centrale per la visualizzazione della
    topologia -->
44      <Pane fx:id="pane" style="-fx-border-color: black; -
    fx-border-width: 2px;"
45            AnchorPane.topAnchor="100.0"
46            AnchorPane.leftAnchor="20.0"
47            AnchorPane.bottomAnchor="20.0"
48            AnchorPane.rightAnchor="450.0" />
49
50      <!-- HBox per i drop down posizionati in alto a
    destra -->
51      <HBox spacing="10" alignment="CENTER"
52            AnchorPane.topAnchor="70.0"
53            AnchorPane.rightAnchor="60.0">
54          <children>
55              <!-- Tendina per il Flavour Priority -->
56              <VBox spacing="5" alignment="CENTER">
57                  <Label text="Flavour Priority"/>
```

```
58                  <ComboBox fx:id="comboFlavourPriority" value="incremental">
59                      <items>
60                          <FXCollections fx:factory="observableArrayList">
61                              <String fx:value="incremental"/>
62                              <String fx:value="manual"/>
63                              <String fx:value="lexicographic"/>
64                              <String fx:value="reversed"/>
65                          </FXCollections>
66                      </items>
67                  </ComboBox>
68              </VBox>
69              <!-- Tendina per il Solver -->
70              <VBox spacing="5" alignment="CENTER">
71                  <Label text="Solver"/>
72                  <ComboBox fx:id="comboSolver" value="chuffed">
73                      <items>
74                          <FXCollections fx:factory="observableArrayList">
75                              <String fx:value="chuffed"/>
76                              <String fx:value="gecode"/>
77                              <String fx:value="highs"/>
78                              <String fx:value="lcg"/>
79                          </FXCollections>
80                      </items>
81                  </ComboBox>
82              </VBox>
83          </children>
84      </HBox>
85
```

```
86          <!-- Pannello laterale per i dettagli del deploy (
    altezza ridotta) -->
87          <VBox fx:id="dettagliDeploy" visible="true"
88                style="-fx-border-color: black; -fx-padding:
    10;"
89                alignment="CENTER"
90                AnchorPane.topAnchor="250.0"
91                AnchorPane.rightAnchor="20.0"
92                AnchorPane.bottomAnchor="20.0"
93                prefWidth="400.0">
94              <children>
95                  <Label text="DEPLOY OUTPUT:" fx:id="
    deploymentOutputTitle" style="-fx-font-size: 20px; -fx-
    font-weight: bold;" />
96                  <!-- Output del deploy -->
97                  <VBox fx:id="deploymentOutput" spacing="5"
    style="-fx-padding: 10px 0 0 0;" />
98              </children>
99          </VBox>
100      </children>
101 </AnchorPane>
```

# Appendice B

# Esempi file YAML generati

## B.1 Esempio file YAML dei componenti

```yaml
name: app
components:
  c0:
    type: service
    must: false
    flavours:
      f0: { uses: [ { component: c1, min_flavour: f0 } ] }
      f1: { uses: [ { component: c1, min_flavour: f0 } ] }
    importance_order: [f0, f1]
  c1:
    type: service
    must: true
    flavours:
      f0: { uses: [ { component: c2, min_flavour: f0 } ] }
    importance_order: [f0]
  c2:
    type: service
    must: true
    flavours:
      f0: { uses: [ { component: c3, min_flavour: f0 } ] }
```

```
21      importance_order: [f0]
22    c3:
23      type: service
24      must: false
25      flavours:
26        f0: { uses: [ { component: c4, min_flavour: f0 } ] }
27      importance_order: [f0]
28    c4:
29      type: service
30      must: true
31      flavours:
32        f0: { uses: [ { component: c5, min_flavour: f0 } ] }
33      importance_order: [f0]
34    c5:
35      type: service
36      must: false
37      flavours:
38        f0: { uses: [ { component: c6, min_flavour: f0 } ] }
39      importance_order: [f0]
40    c6:
41      type: service
42      must: false
43      flavours:
44        f0: { uses: [] }
45        f1: { uses: [] }
46        f2: { uses: [ { component: c7, min_flavour: f0 } ] }
47      importance_order: [f0, f1, f2]
48    c7:
49      type: service
50      must: true
51      flavours:
52        f0: { uses: [] }
53        f1: { uses: [] }
54      importance_order: [f0, f1]
55  requirements:
```

```yaml
components:
  c0:
    common: { storage: { value: 128 } }
    flavour-specific:
      f0: { ram: { value: 67 } }
      f1: { ram: { value: 17 } }
  c1:
    common: { ram: { value: 133 } }
    flavour-specific:
      f0: { cpu: { value: 4 } }
  c2:
    common: {}
    flavour-specific:
      f0: { ram: { value: 25 } }
  c3:
    common: {}
    flavour-specific:
      f0: { ram: { value: 11 } }
  c4:
    common: {}
    flavour-specific:
      f0: { ram: { value: 61 } }
  c5:
    common: { storage: { value: 121 } }
    flavour-specific:
      f0: { ram: { value: 58 } }
  c6:
    common: { ram: { value: 104 } }
    flavour-specific:
      f0: { cpu: { value: 4 } }
      f1: { cpu: { value: 4 } }
      f2: { cpu: { value: 3 } }
  c7:
    common: {}
    flavour-specific:
```

```
91          f0: { ram: { value: 74 } }
92          f1: { ram: { value: 28 } }
93    dependencies:
94      c0:
95        f1:
96          c1:
97            bwIn: { value: 102 }
98            bwOut: { value: 98 }
99      c1:
100       f0:
101         c2:
102           bwIn: { value: 17 }
103           bwOut: { value: 42 }
104     c2:
105       f0:
106         c3:
107           bwIn: { value: 37 }
108           bwOut: { value: 64 }
109     c3:
110       f0:
111         c4:
112           bwIn: { value: 104 }
113           bwOut: { value: 84 }
114     c4:
115       f0:
116         c5:
117           bwIn: { value: 77 }
118           bwOut: { value: 27 }
119     c5:
120       f0:
121         c6:
122           bwIn: { value: 9 }
123           bwOut: { value: 73 }
124     c6:
125       f2:
```

```
        c7:
          bwIn: { value: 89 }
          bwOut: { value: 66 }
  budget: { cost: 2000000, carbon: 2000000 }
```

## B.2 Esempio file YAML dell'Infrastruttura

```
name: infrastructure
nodes:
  node_0:
    capabilities:
      cpu: 674
      ram: 745
      storage: 755
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 9
  node_1:
    capabilities:
      cpu: 127
      ram: 178
      storage: 580
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 6
  node_2:
    capabilities:
      cpu: 195
      ram: 245
      storage: 178
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 3
  node_3:
```

```
    capabilities:
      cpu: 134
      ram: 348
      storage: 580
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 8
  node_4:
    capabilities:
      cpu: 672
      ram: 328
      storage: 671
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 2
  node_5:
    capabilities:
      cpu: 882
      ram: 670
      storage: 542
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 7
  node_6:
    capabilities:
      cpu: 813
      ram: 722
      storage: 774
    profile:
      cost: { cpu: 1, ram: 1, storage: 1 }
      carbon: 6
links:
  - connected_nodes: [ node_0, node_1 ]
    capabilities: { bwIn: 585, bwOut: 303 }
  - connected_nodes: [ node_0, node_2 ]
```

```
    capabilities: { bwIn: 932, bwOut: 491 }
  - connected_nodes: [ node_0, node_3 ]
    capabilities: { bwIn: 367, bwOut: 431 }
  - connected_nodes: [ node_0, node_4 ]
    capabilities: { bwIn: 960, bwOut: 694 }
  - connected_nodes: [ node_0, node_5 ]
    capabilities: { bwIn: 113, bwOut: 850 }
  - connected_nodes: [ node_0, node_6 ]
    capabilities: { bwIn: 150, bwOut: 97 }
  - connected_nodes: [ node_2, node_6 ]
    capabilities: { bwIn: 907, bwOut: 457 }
  - connected_nodes: [ node_3, node_6 ]
    capabilities: { bwIn: 868, bwOut: 876 }
  - connected_nodes: [ node_4, node_6 ]
    capabilities: { bwIn: 377, bwOut: 136 }
  - connected_nodes: [ node_5, node_6 ]
    capabilities: { bwIn: 347, bwOut: 802 }
```

## B.3 Esempio file YAML delle risorse

```
cpu:
  type: consumable
  optimization: minimization
  worst_bound: 0.0
  best_bound: 10.0
ram:
  type: consumable
  optimization: minimization
  worst_bound: 0.0
  best_bound: 10.0
storage:
  type: consumable
  optimization: minimization
  worst_bound: 0.0
```

```
15    best_bound: 10.0
16  bwIn:
17    type: consumable
18    optimization: minimization
19    worst_bound: 0.0
20    best_bound: 10.0
21  bwOut:
22    type: consumable
23    optimization: minimization
24    worst_bound: 0.0
25    best_bound: 10.0
26  availability:
27    type: consumable
28    optimization: minimization
29    worst_bound: 0.0
30    best_bound: 10.0
31  latency:
32    type: consumable
33    optimization: minimization
34    worst_bound: 0.0
35    best_bound: 0.0
36  security:
37    choices:
38    - ssl
39    - firewall
40    - encrypted_storage
41    optimization: minimization
```