Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

DESIGN, IMPLEMENTAZIONE E VALUTAZIONE DI UN PARADIGMA DI FUNZIONI ANNIDATE IN UN LINGUAGGIO A BLOCCHI PER SERVIZI IOT

Relatore:
Dott.
FEDERICO MONTORI

Presentata da:
VINCENZO ARMANDI

Sessione II
Anno Accademico 2019/2020

Sommario

L'avvento di Internet ha rivoluzionato totalmente il modo in cui le persone comunicano e svolgono qualsiasi tipo di attività. Negli ultimi anni, inoltre, si sta diffondendo sempre più velocemente il paradigma dell'Internet of Things, il cui scopo è quello di migliorare la vita delle persone facilitando una moltitudine di azioni che spaziano tra svariati ambiti, da quello domestico a quello riguardante imprese ed enti governativi, mediante il connubio tra i dati raccolti da sensori e la loro analisi ed elaborazione attraverso Internet. In questo contesto si sviluppa l'IoT Collaborativo, specializzazione dell'IoT, basato sul concetto di condivisione e collaborazione delle parti interessate ai fini della raccolta e dell'elaborazione dei dati. L'attenzione di questa tesi si focalizza su SenSquare, piattaforma utile per la creazione di servizi personalizzati mediante l'uso di dati reperiti da fonti più o meno affidabili e da campagne di crowdsensing. Il contributo della tesi verterà sull'ampliamento delle funzionalità e sull'ottimizzazione delle performance offerte dalla piattaforma. In particolare, verrà potenziata l'espressività del linguaggio che questa utilizza per la creazione di servizi utili ai suoi utenti e verranno introdotte nuove funzionalità ai fini di conferire all'utente un ruolo sempre più centrale all'interno di essa. In seguito al lavoro di tesi l'utente avrà la possibilità di creare servizi congeniali alle sue esigenze servendosi di quelli creati in precedenza. Avrà inoltre la possibilità di eliminare e/o modificare i propri template, oltre a poterli eventualmente riparare se non più funzionanti. Allo stesso modo, queste azioni potranno essere eseguite anche in merito ai servizi stessi. Infine, verrà modificata l'architettura del modulo incaricato di eseguire la computazione relativa ai servizi creati dagli utenti, per fare in modo che la loro esecuzione avvenga in parallelo ai fini di massimizzare l'efficienza e l'affidabilità della piattaforma.

Indice

Elenco delle figure

Elenco dei listati

Introduzione

L'Internet of Things, termine coniato da Kevin Ashton nel 1999, ma la cui prima applicazione risale a circa un decennio prima grazie al primo oggetto reso smart, un tostapane [13], è l'insieme dei sensori integrati in oggetti fisici in grado di comunicare tra loro attraverso la rete. Se fino a 20 anni fa, il paradigma IoT era conosciuto solamente dagli addetti ai lavori, al giorno d'oggi corrisponde a una vera e propria realtà sulla quale molte aziende hanno fondato il loro business principale. Con l'avvento del XXI secolo sono stati compiuti dei veri e propri passi da giganti, basti pensare che si è arrivati a creare intere città connesse alla rete [5] o auto che si guidano autonomamente in base ai dati raccolti ininterrottamente dalla moltitudine di sensori di cui esse sono dotate [3]. Gli esempi sopracitati sono solamente alcune delle innumerevoli applicazioni del paradigma IoT. Quest'ultimo deve la sua rapida pervasività principalmente all'enorme diffusione di dispositivi dotati di sensori e alla loro sempre più facile reperibilità da parte dei normali consumatori. Il lavoro di tesi verterà principalmente su un ramo specifico dell'IoT, l'Internet of Things Collaborativo, il quale non si limita al reperimento dei dati da dispositivi dotati di sensori e alla loro analisi ed elaborazione in ambienti chiusi creando una sorta di "Intranet of Things", ma sfrutta la collaborazione di tutte le parti interessate per fornire servizi sempre più accurati e affidabili. In particolare, verranno apportate modifiche e migliorie ad una piattaforma già esistente, denominata SenSquare, la quale, attraverso una moltitudine di dati recuperati da fonti di svariato genere, offre agli utenti la possibilità di creare i propri servizi personalizzati in merito al monitoraggio ambientale. La piattaforma permette quindi ai suoi utenti di creare dei veri e propri modelli personalizzati, detti **template**, che potranno poi essere istanziati per dare vita agli effettivi **servizi** dei quali gli utenti potranno usufruire. Quest'elaborato si articolerà in 6 capitoli. Nel capitolo 1, attraverso una breve panoramica sull'IoT, nello specifico riguardo l'Internet of Things Collaborativo e i Middleware IoT, verrà descritto lo stato dell'arte esponendo vantaggi e problemi riscontrati in precedenza al contributo di questo progetto. In seguito, nel capitolo 2 verrà approfonditamente descritta la piattaforma oggetto di questa tesi, analizzando inizialmente l'architettura alla base di essa, spiegando nel dettaglio il funzionamento dei principali componenti di cui è dotata, per poi illustrare la User Experience mediante appositi screenshot esplicativi. Nel capitolo 3 invece, verranno descritte le scelte di progettazione relative alle migliorie e alle modifiche fornite alla piattaforma. Ci soffermeremo principalmente sui cambiamenti apportati alle tabelle del database, sui nuovi Endpoint aggiunti alle già esistenti API REST e soprattutto sulla rinnovata architettura e sulle nuove potenzialità

offerte dall' Instance Executor, ovvero il modulo incaricato dell'esecuzione di istanze ai fini di creare i servizi finali. Successivamente, nel capitolo 4 si analizzeranno le scelte implementative relative ai 3 macro-contributi forniti da questo lavoro di tesi. Verrà quindi spiegato come si è giunti al potenziamento dell'espressività del linguaggio utilizzato dalla piattaforma per la creazione di servizi personalizzati e saranno descritte le modifiche apportate ai fini di rendere l'esperienza utente più completa e performante. Nel capitolo 5 verranno analizzati i risultati ottenuti, confrontandoli con quelli raggiunti dalla precedente implementazione, per poi mostrarli attraverso specifiche tabelle e grafici creati appositamente. Infine, nel capitolo 6, ultimo capitolo della tesi, sarà lasciato spazio alle conclusioni attraverso un breve riassunto dell'elaborato e saranno proposte alcune delle tante idee realizzabili in futuro per ampliare le tante funzionalità già offerte dalla piattaforma.

Capitolo 1

Stato dell'Arte

1.1 Internet of Things Collaborativo

L'Internet of Things è un paradigma che prevede la connessione alla rete di un qualsiasi sistema di dispositivi fisici in grado di ricevere ed inviare dati con l'ausilio di appositi sensori. Oggi questo paradigma rappresenta una solida realtà dovuta principalmente al notevole incremento che il numero di dispositivi posseduti da ogni individuo ha subito negli ultimi anni.
Secondo uno studio infatti, nel 2020 esistono più di 50 miliardi di dispositivi connessi, che corrispondono in media a 6 dispositivi per ogni abitante del nostro pianeta [7]. Essendo un concetto nato negli ultimi anni, in seguito al suo enorme successo e alla sua rapidissima pervasività, sono state coniate diverse definizioni, ma una tra le più esplicative, in quanto mette in evidenza il rapporto che intercorre fra l'uomo e lo sviluppo tecnologico, è quella data dall'IoT Council [4]:

> "The Internet of Things (IoT) is a vision. It is being built today. The stakeholders are known, the debate has yet to start. In hundreds of years our real needs have not changed. We want to be loved, feel safe, have fun, be relevant in work and friendship, be able to support our families and somehow play a role however small in the larger scheme of things. So what will really happen when things, homes and cities become smart. The result will probably be an tsunami of what at first looks like very small steps, small changes. The purpose of council is to follow and forecast what will happen when smart objects surround us in smart homes,offices, streets, and cities."

Le applicazioni dell'IoT coinvolgono una moltitudine di settori di diversa portata, come ad esempio le nostre case e le città in cui viviamo, fino ad interessare grandi imprese ed enti internazionali coinvolti nella raccolta di una enorme mole di dati. Si sta diffondendo sempre più rapidamente il concetto di Smart Home, la cosiddetta casa smart, nella quale sono presenti svariati oggetti dotati di specifici sensori e connessi alla rete che lavorano in sinergia scambiandosi dati e informazioni con l'obiettivo di facilitare la vita degli occupanti all'interno della propria abitazione, automatizzando azioni eseguite quotidianamente calibrate sulle loro esigenze e preferenze. Uno dei tanti esempi è quello dei "Termostati Smart"[15], i quali possono essere programmati manualmente oppure attraverso sistemi di Machine Learning e di Intelligenza Artificiale, analizzando e memorizzando le abitudini degli inquilini per poi eseguire specifici task. Un altro scenario nel quale l'applicazione e l'utilizzo dell'IoT è in rapida ascesa riguarda le città in cui viviamo, denominate Smart City, in quanto attraverso l'utilizzo di appositi sensori si mira principalmente a rendere la vita del cittadino più sicura, ad esempio attraverso il monitoraggio del traffico per la prevenzione di incidenti[1] e il monitoraggio delle vibrazioni e delle condizioni generali dei ponti [16]. A causa della vastità e diversità di scenari che al giorno d'oggi richiedono l'utilizzo del paradigma IoT, viene prodotta un'enorme mole di dati proveniente da fonti completamente diverse tra loro che genera insiemi eterogenei di informazioni, i quali risultano difficili da analizzare e gestire. Questo, unito all'esigenza di trovare una soluzione ad uno specifico problema il prima possibile piuttosto che impiegare maggior tempo per cercare di trovare una soluzione che un giorno possa essere d'aiuto anche ad altri, porta alla nascita delle cosiddette "IoT Island", delle vere e proprie isole chiuse che rendono impossibile la comunicazione con l'esterno e di conseguenza l'interoperabilità, creando una sorta di "Intranet of Things" [8]. La necessità di avere sistemi in grado di collaborare tra loro, ha portato quindi alla nascita del C-IoT, il cosiddetto "Internet of Things Collaborativo", che può essere visto come caso particolare del paradigma IoT, orientato alla comunicazione e all'interoperabilità tra persone, imprese ed enti governativi. Il C-IoT si serve di "Collective Awareness Paradigms" (CAPs) [9] per la raccolta di informazioni, i quali rivoluzionano il concetto di IoT inteso come isole chiuse, in quanto mirano alla realizzazione di modelli comuni facendo affidamento sulla cooperazione dei partecipanti, che condividono volentieri i loro dati e collaborano nell'esecuzione di task complessi. Ciò si traduce in uno sforzo irrisorio per il singolo individuo portando ad ottenere un enorme vantaggio a livello di collettività e ad un massivo risparmio economico. Due dei paradigmi principali utilizzati da servizi C-IoT per il reperimento dei dati sono il Mobile Crowdsensing (MCS) e l'impiego di Open Data [9]. Questi ultimi, come si evince dal nome,

sono dati accessibili a tutti in maniera molto rapida e facilmente comprensibili dai calcolatori situati in repository pubbliche. Questi possono essere scissi in due tipologie: reliable, provenienti da fonti affidabili e certificate, e non-reliable che comprende tutti i dati ottenuti dagli utenti sotto forma di campagne di crowdsensing o attraverso altre tecniche di raccolta dati [7]. Il MCS è un paradigma attraverso il quale un certo numero di individui, denominati "partecipants", sono incaricati di eseguire dei task coinvolgendo fenomeni di interesse comune, per reperire i dati ad essi relativi tramite i propri dispositivi connessi alla rete e dotati di specifici sensori [2]. Il C-IoT presenta alcuni problemi che nel tradizionale IoT erano solamente di natura tecnologica, ma che in questo caso diventano di natura sociale in quanto gli utenti necessitano di essere istruiti all'approccio verso questa nuova realtà basata sulla cooperazione e successivamente incentivati alla partecipazione.

1.2 Middleware per Servizi IoT

Una tecnologia chiave per la realizzazione di applicativi per l'Internet of Things è l'IoT middleware [10], un software progettato per fungere da intermediario fra i dispositivi IoT e le rispettive applicazioni.
Nel corso degli anni sono stati implementati molteplici middleware e protocolli per dispositivi IoT, tra cui il Bluetooth Low Energy (BLE) e il Constrained Application Protocol (CoAP) [14].
Possiamo suddividere le esistenti architetture di progettazione di middleware IoT in tre tipologie. La prima adotta soluzioni service-based e generalmente utilizza un architettura service oriented (SOA), permettendo ad utenti e sviluppatori di utilizzare un ampio range di dispositivi IoT per la creazione di servizi. La seconda, conosciuta come cloud-based solution, vincola gli utenti riguardo la tipologia e la quantità di dispositivi IoT utilizzabili, ma allo stesso tempo permette loro di connettersi, collezionare e successivamente interpretare i dati con facilità attraverso casi d'uso determinati e implementati a priori dallo sviluppatore. La terza ed ultima si serve dell'actor-based framework, che utilizza l'architettura "open, plug and play" secondo cui una moltitudine di dispositivi IoT riutilizzabili vengono distribuiti all'interno della rete.

L'architettura service-based (figura 1.1) è un middleware ad alte prestazioni ed è generalmente implementata sotto forma di molteplici nodi situati su potenti gateway posti tra i dispositivi e le corrispondenti applicazioni. Questa particolare architettura non è predisposta per funzionare su dispositivi con risorse limitate, come smartphone e smartwatch, e non supporta la comunicazione tra dispositivi.

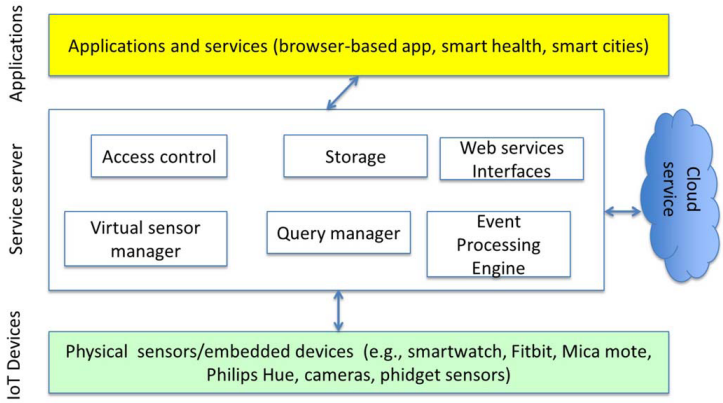


Figura 1.1: Architettura service-based

Come possiamo osservare nella figura 1.2 i componenti funzionali del middleware nell'architettura cloud-based sono limitati solamente a ciò che è disponibile nel cloud. Tipicamente queste funzionalità vengono implementate sotto forma di APIs. Di conseguenza i servizi disponibili nel cloud sono accessibili agli utenti tramite un client dedicato oppure attraverso le sopra citate RESTfull APIs se rese fruibili a terzi.

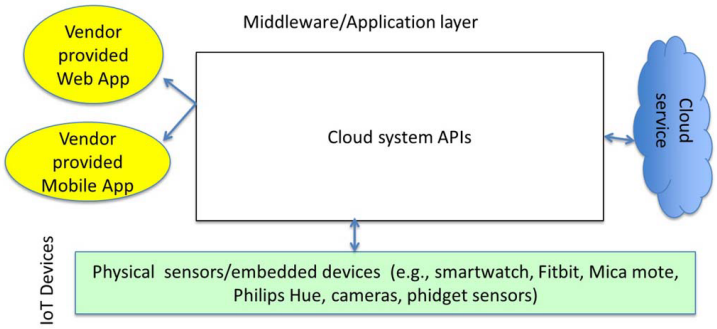


Figura 1.2: Architettura cloud-based

L' architettura actor-based (figura 1.3) è organizzata su tre livelli. Il più basso, denominato "sensor swarm", è costituito dai sensori incaricati di effettuare le misurazioni, il secondo livello è il "cloud services", mentre il terzo ed ultimo è detto "mobile access layer".

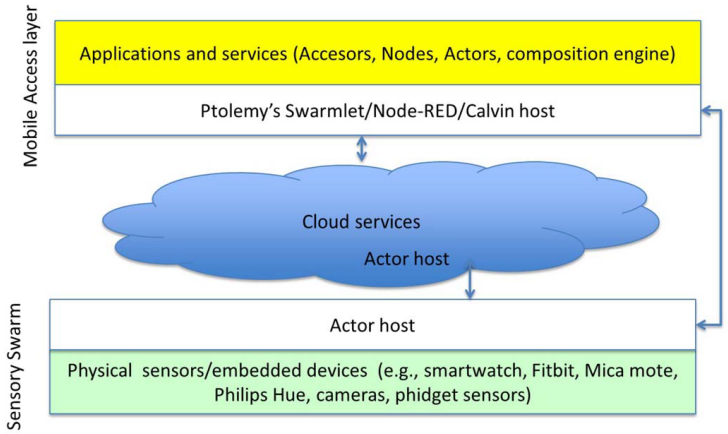


Figura 1.3: Architettura actor-based

La differenza sostanziale tra le tre riguarda l'inclinazione dell'architettura a supportare nuovi dispositivi IoT e al luogo in cui verrà posizionato il middleware. Quello basato su un'architettura service-based è implementato in server o cloud e fornisce agli utenti semplici strumenti sotto forma di Web Application attraverso i quali visualizzare e monitorare i dati raccolti. Spesso però vengono imposte limitazioni all'utente finale, soprattutto nell'interpretazione dei dati e nell'integrazione dell'applicativo con altre applicazioni, inoltre non sono pensate per essere estese e/o customizzate in alcun modo. L'actor-based middleware fornisce la miglior latenza e scalabilità per dispositivi connessi su larga scala, in quanto può essere posto su uno qualsiasi dei tre livelli ed i dispositivi IoT eseguono la loro computazione solamente se necessario. Al contrario dei middleware basati su architetture service-based, l'utente può estendere le sue funzionalità sviluppando un attore aggiuntivo o più semplicemente scaricandone uno da una repository centrale dedicata. Entrambe le tipologie sopra citate non necessitano di un particolare standard, come le RESTful APIs o il BLE per garantire l'interoperabilità tra i dispositivi, in quanto entrambi implementano un livello di astrazione. Il middleware cloud-based, a differenza delle altre due tipologie, garantisce l'interoperabilità tra i dispositivi adottando specifici standard.

Le applicazioni IoT lavorano tipicamente in ambienti dinamici e incerti, in quanto per esempio i dispositivi potrebbero spegnersi in seguito all'esaurimento della batteria, potrebbero perdere la connessione alla rete, ecc. Il middleware deve quindi fornire un "service discovery", che permette di aggiungere nuovi servizi quando disponibili e di rimpiazzare dinamicamente quelli non più funzionanti.

Da quanto detto in precedenza è facile intuire che l'architettura della piattaforma oggetto del lavoro di tesi appartiene alla seconda categoria, la cloud-based architecture, in quanto vi è un nodo centrale, il cloud stesso, in cui possiamo collocare le API REST. Queste permettono agli utenti, mediante il client, di interagire con il server per svolgere operazioni CRUD all'interno della piattaforma o per eseguire la computazione relativa ad uno specifico servizio istanziato.
In seguito, verranno riportati alcuni esempi di applicativi basati su un'architettura di tipo cloud-based.

Google Fit

Google Fit è un ecosistema IoT decisamente vasto e personalizzabile [6].
È considerato a tutti gli effetti un middleware IoT implementato mediante l'utilizzo di un'architettura cloud-based, che offre ai propri utenti la possibilità di tenere traccia dei dati relativi all'attività fisica che praticano oltre a permettere la possibilità di integrare applicazioni di terze parti ai fini di garantire un unico nodo centralizzato nel quale avere a disposizione tutti i dati relativi alla propria forma fisica creando dei veri e propri servizi personalizzati. La piattaforma infatti si serve dei sensori contenuti negli activity tracker o negli smartphone dei propri utenti per registrare le loro attività quotidiane e/o sportive consentendo loro di impostare obiettivi personalizzati e di tener traccia di tutte le attività svolte. L'applicazione quindi aiuta gli utenti a misurare le proprie prestazioni in moltissimi sport e tra la moltitudine di funzionalità che essa offre, permette di calcolare la durata dell'attività sportiva, la distanza percorsa, il numero di passi compiuti, la frequenza cardiaca e il percorso effettuato. Google Fit supporta un vasto numero di dispositivi, in quanto fornisce supporto nativo per tutti i device IoT che comunicano mediante il protocollo Bluetooth Low Energy (BLE). Fornisce inoltre un singolo set di API ai produttori di applicazioni e sensori, in quanto necessarie per archiviare e accedere ai dati delle attività da app, sensori fitness e altri dispositivi. In conclusione, può essere vista come una piattaforma di C-IoT in quanto permette agli utenti di scegliere se ed eventualmente con chi condividere i propri dati per svariati scopi.

Paraimpu

Paraimpu è un social tool ideato per permettere agli utenti di connettere, condividere e far interagire tra loro oggetti, servizi e dispositivi in modo da creare applicazioni personalizzate e che rispondano a tutte le esigenze dei propri utenti [12]. La caratteristica principale di Paraimpu è quella di connettere "smart object" fisici con ambienti virtuali, servizi Web e social network permettendo:

- L'interconnessione in maniere semplificata e assistita di oggetti IoT.
- L'interazione di oggetti con social network suscitando in loro reazioni a stimoli esterni quali sensori ambientali o attività social.
- La condivisione di dati prodotti mediante la pubblicazione di informazioni permettendo il loro riutilizzo da parte di altri utenti.

La piattaforma è inoltre molto aperta e inclusiva in quanto qualsiasi oggetto capace di comunicare attraverso il protocollo HTTP è potenzialmente in grado di interagire con il sistema. Permette quindi ai propri utenti di comporre semplici applicazioni IoT mediante l'utilizzo di Javascript. Tutti i sensori e i servizi di cui è fornita sono rappresentati attraverso API REST per permettere a tutte le parti coinvolte di utilizzare qualsiasi risorsa presente all'interno della piattaforma, in modo da garantire interoperabilità interna.

IFFT

If This Then That (IFTTT) è un servizio web che permette la creazione di semplici "catene di condizioni" denominate applet. Un'applet può essere utilizzata da altri servizi e in base all'azione eseguita all'interno di essi in fase di attivazione scatena una determinata azione che è una diretta conseguenza della precedente. Ad esempio, si potrebbe creare un applet che invia una e-mail quando l'utente utilizza un hashtag in un tweet e che invia invece ad un cloud una copia di una foto estrapolata dal proprio account Facebook quando l'utente viene citato(taggato) in essa. Possiamo quindi definire IFTTT come un Web-Service che aiuta altri Web-Service a interconnettersi e comunicare tra loro. Utilizzando una metafora, potremmo definirlo come il collante che tiene uniti tra loro vari servizi web [11].

I principali attori all'interno del paradigma IFTTT sono:

- **Services**: sono i blocchi alla base di IFTTT. Ad esempio, descrivono dati che possono riguardare uno specifico servizio web o anche azioni controllate da apposite APIs.
- **Triggers**: corrispondono alla parte "this" del paradigma e sono gli elementi che avviano un'azione.
- **Actions**: corrispondono alla parte "that" del paradigma e sono il risultato di quanto avviato dai trigger.
- **Applets**: costituiscono l'insieme formato da trigger e action.
- **Ingredients**: sono i singoli dati resi disponibili da un trigger.

Dalle loro descrizioni possiamo notare che tutte condividono la stessa architettura della piattaforma oggetto del lavoro di tesi, ma che allo stesso tempo si differenziano, anche se in modi diversi da essa. La prima infatti, permette di condividere i dati tra i propri utenti ma al contrario di essa non permette che questi ultimi si servano in maniera esplicita e a proprio piacimento dei dati appartenenti agli altri utilizzatori della piattaforma. La seconda invece, anche se ha un differente campo di applicazione, è molto più simile alla piattaforma oggetto di questo progetto, in quanto anch'essa permette ai propri utenti di utilizzare servizi creati da altri in modo da garantire una migliore e più performante esperienza utente. Infine, anche la terza è molto simile alla piattaforma sopracitata, in quanto, allo stesso modo, dà la possibilità ai propri utenti di creare servizi personalizzati congeniali alle proprie esigenze.

1.3 Motivazioni e Contributi

Il contributo di questo lavoro di tesi è incentrato sul potenziamento e miglioramento di SenSquare, una piattaforma C-IoT per il Community Based Monitoring e l'Envirormental Monitoring, che integra fonti di dati eterogenee e le mette a disposizione dell'utente finale grazie all'impiego di un client facilmente accessibile, senza il bisogno di possedere specifici sensori compatibili con l'architettura o di dover consumare delle APIs [8].
Attraverso la piattaforma l'utente può creare servizi personalizzati a lui utili mediante la tecnica del Visual Programming, servendosi dell'editor visivo Blockly, uno strumento molto semplice e intuitivo utilizzabile anche da individui che non hanno particolari competenze di programmazione.
Per comprendere a pieno il contributo di questa tesi, è necessario fare una breve panoramica sul processo di creazione e gestione di un servizio personalizzato da parte dell'utente, di conseguenza faremo riferimento ad alcuni elementi che compongono l'architettura di SenSquare, come i CustomServiceTemplate e i CustomService, che verranno poi analizzati dettagliatamente nell'apposita sezione.
Per realizzare il proprio servizio personalizzato, l'utente deve inizialmente creare un Custom Service Template servendosi dell'editor visivo sopracitato. Senza entrare troppo in dettaglio, possiamo considerarlo a tutti gli effetti come lo scheletro del servizio finale, un vero è proprio modello predefinito che successivamente potrà essere istanziato in una determinata area geografica per dare vita al Custom Service, ovvero all'effettivo servizio.
La gran parte del lavoro di tesi è incentrata sul potenziamento dell'espressività del linguaggio per la creazione di servizi, in quanto, in seguito al contributo apportato, l'utente avrà la possibilità di riutilizzare template già esistenti per creare il proprio "template composto", importandoli mediante due modalità, sotto forma di Whitebox o Blackbox, che saranno dettagliatamente esposte nella sezione a loro dedicata. Di conseguenza il servizio non viene più concepito come una singola istanza di un template, ma come un'istanza madre che al suo interno contiene delle istanze figlie che dovranno essere eseguite singolarmente e successivamente sarà necessario aggregare i risultati ottenuti per ottenere il valore finale da restituire all'utente.
Questa soluzione trova una perfetta collocazione all'interno della categoria contenente le applicazioni basate su architettura cloud-based. Nel paragrafo precedente infatti, possiamo notare una certa corrispondenza con il secondo esempio fornito, in quanto anche SenSquare, proprio come paraimpu, permette ai propri utenti di utilizzare servizi creati da altri per garantire una migliore esperienza utente.
Successivamente, si è deciso di rendere la piattaforma ancor più "user-centered",

inserendo la possibilità di modificare e rimuovere Template o Servizi già esistenti. Da ciò, è scaturita la necessità di introdurre i concetti di "Broken Template" e "Broken Instance", dando di conseguenza la possibilità all'utente proprietario di ripararli così da renderli perfettamente funzionanti. Infine, dato che potenzialmente potrebbero essere presenti servizi molto articolati, dopo un' attenta disamina sui dati raccolti riguardo il tempo di esecuzione degli stessi, è stata rivoluzionata la loro modalità di esecuzione in quanto è stata implementata una procedura che viene eseguita in parallelo sfruttando i molteplici thread offerti dall'architettura che ospita la piattaforma per minimizzarne il tempo di esecuzione.

Capitolo 2

SenSquare

Nel corso degli ultimi anni è stato appurato che la diffusione dell'Internet of Things ha permesso di incrementare le possibilità di realizzare servizi incentrati sulle persone e sull'ambiente nel quale queste vivono. A tal proposito, in un precedente progetto[8], è stata implementata una piattaforma basata sui principi del Collaborative Internet of Things (C-IoT) denominata SenSquare, che si pone l'obiettivo di stimolare l'utente finale a utilizzare, per scopi nobili, i dati eterogenei presenti in cloud non pienamente utilizzati per la creazione di servizi finalizzati a migliorare principalmente la vita delle persone e la salvaguardia ambientale. Si pone quindi come un'efficiente soluzione di Internet of Things Collaborativo in cui gli utenti collaborano nella raccolta di dati e nella creazione di servizi condivisi. La piattaforma infatti, offre agli utenti la possibilità di utilizzare i dati presenti nel proprio database, fornendo un mezzo con cui definire modelli predefiniti per la creazione e la successiva istanziazione di servizi personalizzati. Gli elementi principali coinvolti nel processo di creazione e successivamente di istanziazione ed esecuzione di un servizio sono i CustomServiceTemplate, i CustomService e i Datastream. I template possono essere considerati come il codice sorgente del servizio che si vuole istanziare e i datastream invece sono gli effettivi argomenti passati al programma al momento dell'esecuzione. Quindi un template non è altro che un'entità astratta in quanto non dispone né di dati concreti né di un'area geografica di operatività. Di conseguenza, nel momento in cui se ne istanzia uno, gli si dà concretezza scegliendo i datastream da utilizzare e l'area geografica di operatività. SenSquare prevede che ogni utente possa definire in maniera semplice i propri template, e allo stesso tempo, che i più esperti possano sbizzarrirsi nella creazione di programmi complessi. Questo è reso possibile anche mediante l'utilizzo di apposite strutture, denominate Blackbox e Whitebox, introdotte grazie al contributo di questa tesi per importare e quindi inglobare template già esistenti nella piattaforma al fine di dare vita

a template composti.

2.1 Architettura della piattaforma

In questa sezione analizzeremo i componenti principali dell'architettura della piattaforma mediante un breve excursus per poi soffermarci maggiormente sugli elementi aggiunti durante lo svolgimento di questo progetto.
Come si può osservare nella figura 2.1 , l'architettura di SenSquare è formata da 3 macro-aree:

- il client, il quale, mediante l'utilizzo di svariati dispositivi collegati alla rete e tramite una Web Application, conferisce agli utenti la possibilità di visualizzare i dati e di interagire con la piattaforma.
- il server, nodo centrale della piattaforma, incaricato della gestione della comunicazione tra tutte le risorse che garantiscono il corretto funzionamento dell'applicativo, tra cui il modulo per la conversione e l'interpretazione dei blocchi per l'editor visuale e i moduli incaricati del reperimento e della classificazione dei dati.
- il database, incaricato di conservare i dati essenziali per il corretto funzionamento del programma.

Il client costituisce la parte relativa al front-end, mentre sia il server che il database formano il back-end della piattaforma.

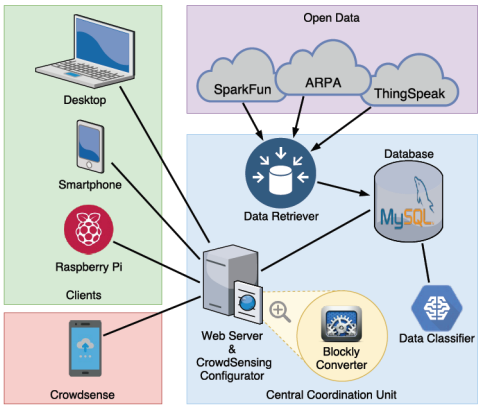


Figura 2.1: Architettura SenSquare

2.1.1 Back-end

Il server, come già detto, rappresenta il cuore della piattaforma che partecipa attivamente alla maggior parte dei processi coinvolti durante tutto il ciclo di vita del programma, ricoprendo quindi una moltitudine di ruoli e offrendo svariati servizi. Come si nota in figura 2.1, il server funge da intermediario fra il client e i diversi moduli che compongono il back-end.
Tra essi, è doveroso spiegare dettagliatamente il funzionamento del Data Retriever e del Data Classifier[7], moduli incaricati rispettivamente del reperimento e della classificazione dei dati.
Il primo viene utilizzato per recuperare periodicamente informazioni da fonti "statiche", le quali sono costituite da enti governativi che rendono disponibili le informazioni in repository pubbliche e da applicativi Open Data che mettono a disposizione di tutti enormi moli di dati (in figura, nel riquadro viola in alto, possiamo osservare esempi di alcune delle piattaforme Open Data dalle quali SenSquare reperisce i dati). Oltre alla collezione dei dati, poiché questi non sempre vengono reperiti da fonti affidabili e di solito non sono presentati attraverso apposite strutture che permettono di identificarli a priori, il server ha il compito di classificarli secondo delle regole prestabilite. Questo è reso possibile dal secondo modulo sopracitato, il Data Classifier, il quale, attraverso un approccio basato su Natural Language Processing (NLP), è in grado di identificare e classificare i dati eterogenei reperiti dal Data Retriever, assegnando ad ognuno di essi un tipo, detto Data Type (pressione, temperatura, PM10, CO2, ecc) e un'unità di misura.
Tra i tanti ruoli ricoperti vi è sicuramente quello più classico e comune di WebServer, dato che è l'unico punto di accesso per i client attraverso protocollo HTTP mediante REST API e in quanto è incaricato di fornire i dati necessari alla Web Application per permettere l'interazione con gli utenti. Possiede inoltre il fondamentale compito di ospitare l'Instance Executor, il modulo che si occupa della computazione relativa alle varie istanze, grazie al quale si ottengono i risultati restituiti a seguito dell'esecuzione di un servizio che saranno poi passati al client per permettere all'utente di visualizzarli (La descrizione di questo particolare componente verrà approfondita nella sezione dedicata in quanto sono state apportate delle sostanziali modifiche ai fini del lavoro di tesi). Infine, il server è l'unico componente in grado di comunicare e interagire rispettivamente con il database e il modulo denominato Blockly Converter il quale ha il compito di convertire il codice generato dal framework Blockly in uno script in python che sarà poi eseguito dall'Instance Executor.
Per comprendere in maniera più approfondita il funzionamento di quest'ultimo modulo, verranno illustrati in seguito i componenti principali che com-

pongono un servizio: **CustomServiceTemplate**(CST) e **CustomServiceInstance**(CSI).

CustomServiceTemplate

Prima del contributo di questa tesi, avremmo potuto definire i custom service template come una combinazione di dati primitivi ottenuta mediante l'utilizzo di semplici espressioni matematiche, operatori logici e blocchi condizionali (if-then-else). Attraverso la BNF sottostante, possiamo osservare la rappresentazione della sintassi del linguaggio utilizzato per la composizione di CSTs.

$$E := IFTE(C, E', E') \mid E'$$

$$E' := c \mid DC \mid (E' + E') \mid (E' - E') \mid (E' * E') \mid (E'/E')$$

$$C := b \mid C \wedge C \mid C \vee C \mid \neg C \mid E > E \mid E \geq E \mid E < E \mid E \leq E \mid E = E \mid E \neq E$$

Dove c è una costante, b un booleano, E identifica l'output generato dal CST, E' l'espressione matematica, DC un datastream e IFTE rappresenta il blocco condizionale if-then-else. Con le modifiche apportate durante il lavoro di tesi, la definizione di CST assume un'accezione decisamente più complessa rispetto alla precedente, in quanto, i template possono inglobare CSTs già esistenti all'interno della piattaforma non limitandosi solamente a contenere combinazioni di dati primitivi ottenuti attraverso semplici operazioni matematiche. È stato inoltre aggiunto un nuovo blocco funzione, il quale ha il compito di contenere template importati sotto forma di blackbox (nella BNF saranno identificate dalla sigla BBX) o blocchi condizionali per far si che questi possano essere restituiti in output. Questo è stato necessario in quanto precedentemente potavano essere restituite solo espressioni semplici o booleani. In seguito, possiamo osservare la BNF relativa alla nuova concezione di CST:

$$E := c \mid DC \mid (E + E) \mid (E - E) \mid (E * E) \mid (E/E) \mid IFTE(C, E, E) \mid BBX$$

$$C := b \mid C \wedge C \mid C \vee C \mid \neg C \mid E > E \mid E \geq E \mid E < E \mid E \leq E \mid E = E \mid E \neq E$$

CustomServiceInstance

Possiamo concepire un CSI come il servizio vero e proprio che la piattaforma fornisce ai propri utenti. Questo è ottenuto mediante la concretizzazione di un CST, passandogli dei datastream come parametro e attribuendogli un'opportuna area di operatività identificata dalle coordinate ad essa relative. Va ricordato infatti, che quest'ultimo non è altro che lo scheletro del servizio che inizialmente risulta essere un'entità astratta. È opportuno specificare inoltre che qualsiasi utente può utilizzare ogni CST pubblico presente sulla piattaforma, anche se questo appartiene ad altri, per la creazione del proprio servizio personalizzato. Una volta creato, al momento della sua esecuzione, il CSI viene passato come parametro all'Instance Executor, il quale, come vedremo nella sezione ad esso relativa, si occuperà della computazione e di restituirne il risultato. All'interno della piattaforma possono essere creati una moltitudine di servizi, alcuni molto semplici come ad esempio un servizio che calcola la temperatura minima in una determinata zona o il tasso di PM10 presente nell'aria, ma che, grazie anche alle modifiche apportate con questo lavoro di tesi, possono raggiungere livelli di complessità non banali.

2.1.2 Front-end

La porzione relativa al front-end del progetto, attraverso una Web Application, garantisce agli utenti di interfacciarsi con la piattaforma per visualizzare i dati e/o creare ed eseguire servizi personalizzati. Di seguito, per comprendere appieno le modalità di utilizzo di SenSquare e per presentare le nuove funzionalità della piattaforma introdotte ai fini di questa tesi, verrà brevemente illustrata la user experience (UX).

1. Se l'utente accede per la prima volta alla piattaforma dovrà registrarsi attraverso l'apposito form (figura 2.2), altrimenti dovrà semplicemente effettuare il login.

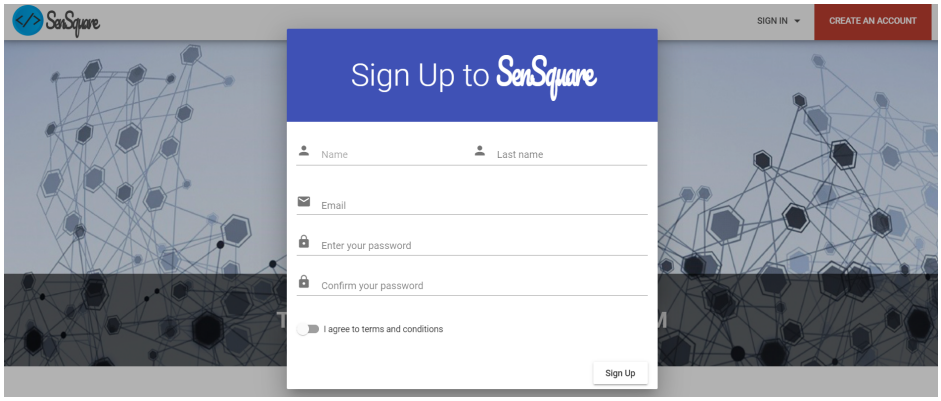


Figura 2.2: Schermata di registrazione

2. In seguito l'utente si ritroverà nella schermata contenente tutti i servizi relativi ad un'opportuna area geografica (figura 2.3), reperita attraverso i servizi di localizzazione in seguito alla richiesta da parte del client di ottenere la posizione dell'utente o ad un luogo inserito manualmente. Qui inoltre, l'utente potrà effettuare un click sulla card relativa ad un servizio istanziato per accedere alla pagina ad esso dedicata.

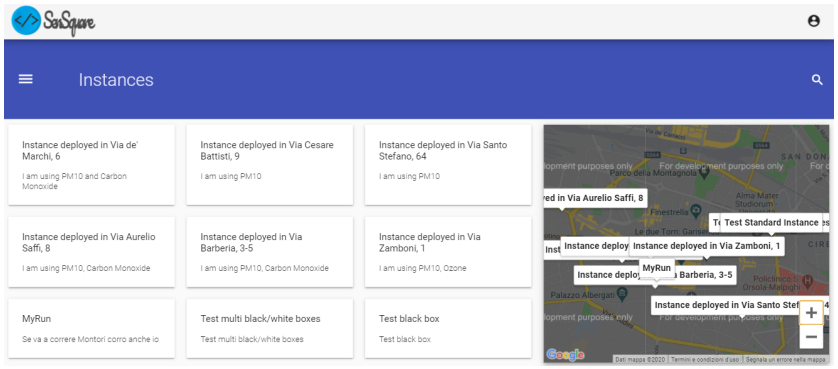


Figura 2.3: Schermata relativa ai servizi presenti nella piattaforma

3. All'interno della pagina di dettaglio di uno specifico servizio (figura 2.4), l'utente potrà visualizzarne i dettagli attraverso un grafico e la lista specifica degli ultimi risultati calcolati per ogni datastream, oltre a poter individuare le precise coordinate di operatività dell'istanza attraverso un'apposita mappa e i valori risultanti dalla sua esecuzione. Inoltre sarà eventualmente in grado di modificare o eliminare definitivamente l'istanza del servizio in questione.

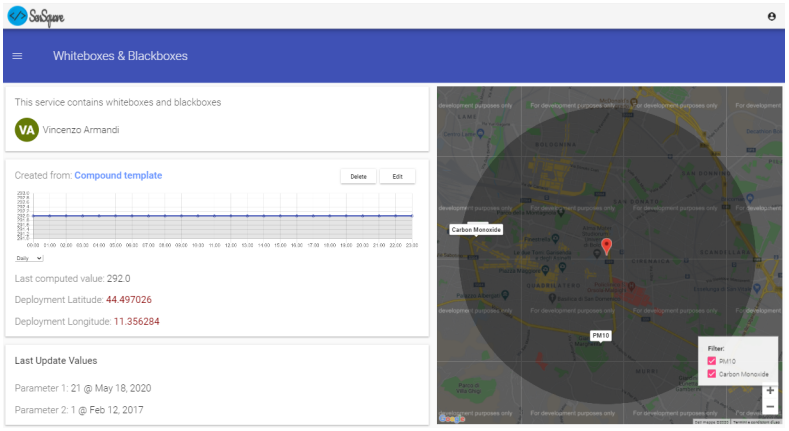


Figura 2.4: Schermata relativa ad uno specifico servizio

4. Cliccando sul pulsante di modifica si potranno cambiare le coordinate relative all'area geografica di operatività dell'istanza e modificare i valori dei datastream selezionandone dei nuovi tra quelli disponibili (figura 2.5).

Figura 2.5: Schermata relativa alla modifica di uno specifico servizio

5. L'utente successivamente potrebbe decidere di spostarsi nella sezione template (figura 2.6) mediante la sidebar di cui la Web App è dotata, nella quale, oltre a poter visualizzare l'elenco di tutti i template presenti all'interno della piattaforma, potrà eseguire due azioni: cliccare sulla card relativa ad un template per accedere alla pagina di dettaglio di quest'ultimo o cliccando sull'apposito pulsante potrà accedere all'editor visivo attraverso cui creare il proprio template personalizzato.

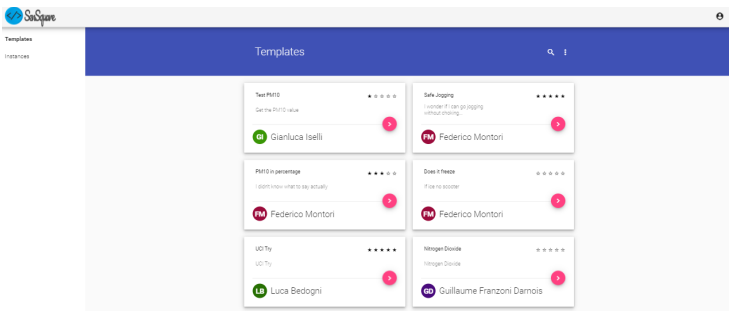


Figura 2.6: Schermata relativa ai template presenti nella piattaforma

6. Nella pagina relativa allo specifico template (figura 2.7) l'utente potrà visualizzarne la sua rappresentazione attraverso il framework di Google Blockly, creare un servizio a partire da esso istanziandolo o eventualmente potrà procedere alla modifica o alla definitiva eliminazione dello stesso.

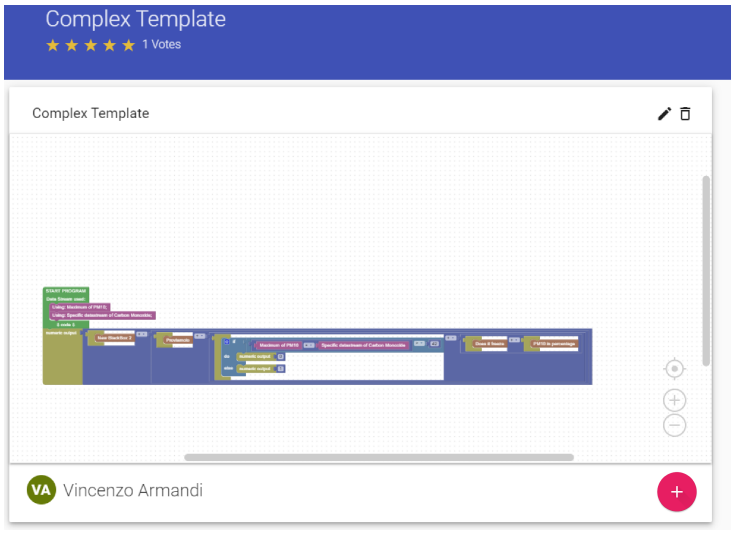


Figura 2.7: Schermata relativa ad uno specifico template

7. Nella schermata di creazione del template (figura 2.8) invece, l'utente potrà creare il suo template personale attraverso l'editor visuale sopracitato e, premendo sull'apposito pulsante, importare altri template già presenti all'interno della piattaforma attraverso due modalità di cui discuteremo in seguito.

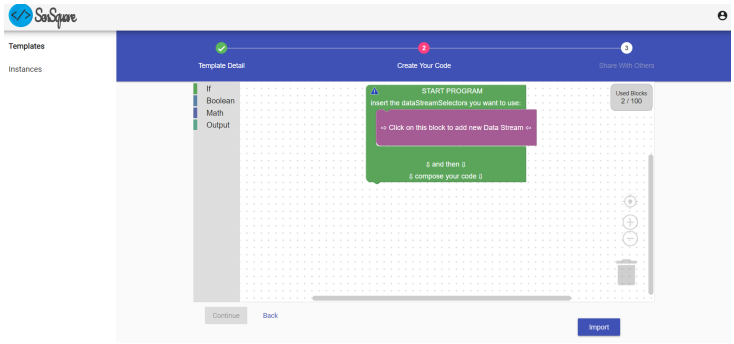


Figura 2.8: Schermata relativa alla creazione di un template

8. Infine, l'utente potrà creare un servizio a partire da uno dei template (figura 2.9) presenti nella piattaforme specificandone un nome e una breve descrizione e definendo le coordinate dell'area geografica di operatività, oltre a selezionare tra quelli disponibili i datastream necessari a garantire la corretta esecuzione del servizio da parte dell'Instance Executor.

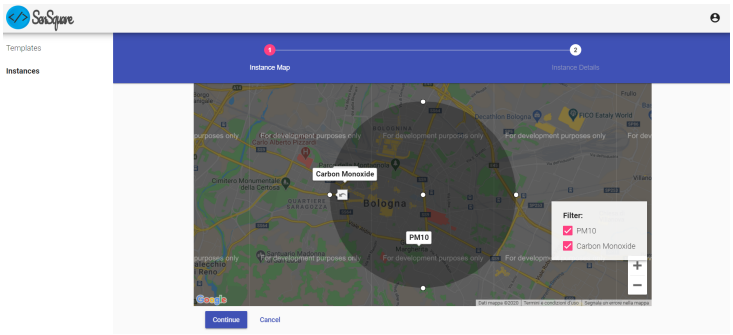


Figura 2.9: Schermata relativa alla creazione di un servizio

Capitolo 3

Progettazione

In questa sezione non verranno trattati aspetti riguardanti l'implementazione già esistente della piattaforma, ma verranno analizzate solamente le principali scelte di progettazione in merito alle nuove funzionalità introdotte in questa tesi. Precisamente analizzeremo le modifiche apportate alle strutture dati presenti nel database, gli endopint delle APIs aggiunti per garantire il corretto funzionamento delle nuove features e le modifiche apportate al concetto di Istanza e di conseguenza alla rinnovata architettura dell'Instance Executor.

3.1 Database

Il database è uno dei componenti principali dell'architettura di SenSquare. La sua presenza infatti è fondamentale per il corretto funzionamento del programma, in quanto contiene molteplici tabelle che fanno riferimento ad una moltitudine di dati imprescindibili ai fini dell'utilizzo della piattaforma. Si è scelto di mantenere il database usato sin dalla nascita dell'applicazione, ovvero MySQL, e di modificare solamente le strutture dati coinvolte nel lavoro di tesi. MySQL è un Relational Database Management System (RDBMS), ovvero un database basato sul modello relazionale e che quindi presenta i dati sotto forma di relazioni mediante l'utilizzo di tabelle costituite a loro volta da righe e colonne.

In seguito verranno quindi elencate e analizzate le strutture dati che hanno ricevuto delle modifiche durante lo svolgimento del progetto.

- **Customservice**: è la tabella del database (figura 3.1) che contiene informazioni relative all'istanziazione dei CustomServiceTemplate, che non sono altro che i veri e propri servizi dei quali gli utenti possono usufruire utilizzando la piattaforma. Come in tutte le tabelle del database è presente un campo univoco che contiene l'ID del servizio istanziato e che svolge il compito di chiave primaria ai fini dell'identificazione dello stesso. Successivamente vi è un campo contenente l'ID associato all'utente che ha istanziato il servizio, che è l'identificatore univoco di quest'ultimo nella tabella che contiene informazioni relative a tutti gli utenti registrati alla piattaforma. In seguito possiamo osservare tre campi che identificano l'area geografica di operatività, ovvero deployment_center_lat, deployment_center_lon e deployment_center_radius che indicano rispettivamente la latitudine, la longitude e il centro del raggio dell'area geografica in cui il servizio è attivo. Possiamo inoltre notare la presenza di alcune informazioni relative al CustomServiceTemplate istanziato quali il suo ID presente nella riga denominata template_ID. Nel campo json_args sono presenti informazioni aggiuntive in formato JSON relative al tipo dei datastream utilizzati dal servizio e alla modalità di misurazione relativa ad ognuno di essi, in base al fatto che l'utente abbia richiesto il minimo, il massimo, la media o una specifica misurazione. In questo lavoro di tesi è stato aggiunto il campo sub_instance_args che contiene il tipo e lo specifico id dei datastream che richiedono misurazioni specifiche, contenuti in tutte le istanze figlie dell'istanza madre (spiegheremo approfonditamente questa nuova concezione di istanza nella sezione dedicata all'Instance Executor). A completare la tabella, troviamo due campi contenenti le informazioni primarie relative al servizio, quali il titolo e una breve descrizione.

CustomService	
ID	**int(11)**
partecipant_ID	**int(11)**
deployment_center_lat	double
deployment_center_lon	double
deployment_radius	double
template_ID	**int(11)**
title	varchar(64)
json_args	longtext
description	varchar(128)
sub_instance_args	longtext

Figura 3.1: Tabella relativa ai Custom Services (le chiavi primarie sono evidenziate in grassetto e i campi aggiunti sono sottolineati)

- **CustomServiceTemplate**: è la tabella del database (figura 3.2) che contiene informazioni relative a uno specifico template creato da uno specifico utente. Di conseguenza, proprio come per la tabella customservice sopra descritta, sono presenti due campi identificativi, uno che identifica il template attribuendogli un ID univoco e l'altro che allo stesso modo identifica l'utente che lo ha creato. In seguito possiamo notare la presenza di alcuni campi relativi alla struttura e al codice prodotto dal template, in quanto il campo xml_template contiene la rappresentazione attraverso il metalinguaggio XML del template generato mediante l'editor visivo fornito dal framework Blockly e il campo python_code che ne contiene la corrispondente rappresentazione tradotta in codice python, che sarà poi passato all'Instance Executor ai fini dell'esecuzione del servizio. Vi è inoltre il campo share_with_others che contiene un booleano che indica se il template in questione è condiviso con gli altri utenti oppure se questo è stato reso privato dal suo creatore. Anche la tabella customservicetemplate proprio come la precedente contiene informazioni basilari come il titolo e una breve descrizione del template, oltre ad avere anch'essa la colonna json_args nella quale sono elencati i datastreams e la loro modalità di misurazione utilizzati dal template.

Infine è stata aggiunta una colonna denominata used_templates necessaria ai fini di identificare tutti i template già esistenti importati sotto forma di Blackboxes in fase di creazione.

CustomServiceTemplate	
ID	**int(11)**
partecipant_ID	**int(11)**
output_type	varchar(16)
expression	varchar(128)
title	varchar(50)
xml_template	longtext
json_args	longtext
python_code	longtext
description	varchar(128)
share_with_others	tinyint(1)
used_templates	longtext

Figura 3.2: Tabella relativa ai Custom Service Templates (le chiavi primarie sono evidenziate in grassetto e i campi aggiunti sono sottolineati)

3.2 REST API

Le API REST sono una scelta implementativa molto importante intrapresa dall'autore della prima implementazione di SenSquare in quanto permettono al back-end di non preoccuparsi di dati superflui e di computazioni costose che inficerebbero solamente sulle performance della piattaforma, poiché non sono altro che un insieme di URI invocabili dai client mediante appositi metodi del protocollo HTTP.
Dato che uno dei punti sui cui si è incentrato il lavoro di tesi è stato quello di rendere la piattaforma ancor più "user-centered", introducendo la possibilità di eliminare o modificare i template e i servizi istanzianti presenti sulla piattaforma, è stato necessario implementare dei metodi appartenenti al back-end attraverso i quali modificare e eliminare template e/o servizi.
Di seguito verranno illustrati solamente gli URI e quindi gli End Point introdotti in questa tesi.

TEMPLATES

1. **Update Template:**
   PATCH: api/v1/user/templates/<template_id>/<data> recupera il template da modificare attraverso il template_id passato come parametro e successivamente verranno rimpiazzati nel database solamente i parametri ottenuti dal campo data, anch'esso passato come parametro.

2. **Delete template:**
   DELETE: api/v1/user/templates/<template_id>/<participant_id> recupera il template da eliminare definitivamente attraverso il template_id passato come parametro e successivamente controlla se questo appartiene all'utente che ha effettuato la richiesta confrontando l'id dell'utente che ha create il template reperito dal database con il participant_id passato come parametro

INSTANCES

1. **Get Instances by Instantiated Template:**
   GET: api/v1/instances/<template_id>
   recupera tutte i servizi che sono stati istanziati a partire dal template indicato dal parametro template_id

2. **Update Instance:**
   PATCH: api/v1/instances/<instance_id>/<args>/<template_id>
   recupera l'istanza da modificare attraverso l'instance_id passato come parametro e successivamente verranno rimpiazzati nel database solamente i parametri contenuti nel campo args, anch'esso passato come parametro

3. **Delete Instance:**
   DELETE: api/v1/instances/<instance_id>/<participant_id> recupera il servizio da eliminare definitivamente attraverso l'instance_id passato come parametro e successivamente controlla se questo è stato creato dall'utente che ha effettuato la richiesta confrontando l'id dell'utente che ha istanziato il template reperito dal database con il participant_id passato come parametro

3.3 Instance Executor

L' Instance Executor è il modulo incaricato di eseguire la computazione associata ad una specifica istanza di un servizio, in modo da poter mostrare all'utente il risultato da esso restituito. La computazione viene eseguita compilando in bytecode il codice python per poi interpretarlo al fine di ottenere l'effettivo valore restituito dal servizio in questione. Questo processo, prima del contributo apportato da questa tesi, consisteva nel calcolare operazioni elementari su diversi dati reperiti dal server attraverso apposite query al database.

Per ogni datastream presente all'interno del template istanziato, potrebbero essere necessarie fino a 4 tipologie differenti di misurazioni: massimo, minimo, media aritmetica e valore corrispondente ad un datastream specifico. Possiamo quindi raggruppare le differenti tipologie di misurazioni sopra elencate in 2 macro-categorie. Vi è il caso in cui venga richiesto uno specifico valore di un datastream, oppure quello in cui è necessario calcolare un valore all'interno di un range di misurazione che può corrispondere appunto al massimo, al minimo o alla media matematica dei valori presenti nell'intervallo di tempo considerato. Di conseguenza, nei precedenti lavori di tesi relativi a questo progetto, sono state implementate due classi all'interno del server per gestire rispettivamente le due situazioni: SpecificDataStream e MaxMinAvgOfDataStream. Le due classi sono molto simili tra loro, in quanto si differenziano solamente per la diversa modalità che utilizzano per il reperimento dei dati dal database. La prima effettua una query al database filtrando la tabella delle misurazioni con l'id dello specifico datastream considerato, mentre la seconda colleziona i dati filtrando le misurazioni in base alle coordinate geografiche ad esse relative e al tipo di sensore che le ha effettuate. In realtà, vi è anche una terza classe che si occupa di effettuare la computazione per tutti i giorni specificati nell'intervallo di tempo e successivamente di aggregare i valori ottenuti attraverso opportune operazioni aritmetiche ai fini di ottenere il risultato finale da passare al server, in modo che questo possa, a sua volta, inviarlo al client che avrà il compito di mostrarlo all'utente.

Come già detto nelle precedenti sezioni dell'elaborato, dopo un'attenta analisi ed un accurato studio, è stato rivoluzionato il concetto di istanza, in quanto in precedenza non era possibile aggregare template già esistenti sulla piattaforma in fase di creazione di un nuovo template personalizzato. Pertanto durante il lavoro di tesi da me svolto, è stata aggiunta la possibilità di importare template già esistenti sotto forma di **Whitebox** o **Blackbox**. In questa frangente ci soffermeremo sulla seconda modalità per poi descrivere la prima e le differenze tra di esse nella sezione relativa alle scelte implementative e all'effettiva implementazione dei vari moduli realizzata durante lo svolgimento del progetto.
Quando l'utente decide di importare un template sotto forma di BlackBox, questo risulta essere una vera e propria "scatola nera" agli occhi della piattaforma. Infatti, nel momento dell'istanziazione di un template, prima di effettuare la computazione relativa ad essa, il sistema controlla se il template da istanziare contiene blackbox, esaminando il codice python da esso prodotto, e in base al risultato di questo controllo agisce di conseguenza.
Se il template non contiene blackbox, l'istanza verrà trattata come un normale servizio e di conseguenza verrà eseguita la computazione mediante il procedimento sopradescritto. Altrimenti, se il template contiene una o più blackbox, queste verranno trattate come servizi indipendenti che si è scelto di denominare sotto-istanze o meglio istanze-figlie, che avranno in comune tra loro e con l'istanza madre solamente l'area geografica di operatività. Di conseguenza l'esecuzione dell'istanza viene scissa in N computazioni indipendenti (con N che indica il numero di sotto-istanze presenti all'interno del servizio), le quali verranno eseguite parallelamente in multithreading e successivamente verranno aggregati i risultati ottenuti da ognuna di esse per comporre quello che sarà il valore effettivo calcolato dal servizio che verrà mostrato mediante la Web Application all'utente finale.

Nello snippet sottostante possiamo osservare la presenza di blackbox all'interno dello script in python corrispondente al template da istanziare, dalla parola chiave "template" affiancata dall'id numerico corrispondente alla blackbox importata. Di conseguenza per ogni sotto-istanza, il sistema reperirà dal database le informazioni necessarie tra cui il codice python ad essa relativa ed eseguirà ricorsivamente il procedimento sopra esplicato. Al termine verrà calcolato il valore restituito dall'istanza madre, che verrà poi aggregato con tutti i risultati ottenuti dalle istanze-figlie, così da ottenere il valore che sarà poi restituito al client.

```
import sys

var_PM10_MAX = float(sys.argv[1])
var_COPR_DSID = float(sys.argv[2])

if (var_PM10_MAX) * (var_COPR_DSID) > 42:
  print(0)
else:
  print((((template23) + ((template24) + (template12))) + ((
    template13) + (template16))))
```

Codice 3.1: Codice python relativo ad un template contenente blackbox

3.4 Introduzione ai concetti di Whitebox e Blackbox

In questo paragrafo, come accennato in precedenza, verranno presentati brevemente i concetti di whitebox e blackbox (che saranno poi ampiamente descritti all'interno del capitolo relativo all'implementazione) introdotti all'interno della piattaforma durante questo lavoro di tesi e in seguito verranno analizzate le problematiche che sono scaturite dal loro impiego. Come già detto, durante lo svolgimento di questo progetto, tra le tante modifiche e migliorie apportate alla piattaforma, è stata introdotta una funzionalità che, in fase di creazione di un nuovo template, permette all'utente di riutilizzare template già esistenti, creati in precedenza da lui o da altri utenti, semplicemente importandoli all'interno dell'editor di creazione, scegliendo tra le due modalità disponibili. L'utente infatti può decidere di importare un template già esistente sotto forma di Whitebox o Blackbox. Questi due concetti, nonostante abbiano lo stesso scopo, ovvero quello di permettere all'utente di riutilizzare un CST presente nella piattaforma, inserendolo all'interno del template in fase di creazione, sono estremamente differenti l'uno dall'altro.

La prima modalità permette all'utente di acquisire una vera e propria copia del template importato all'interno del proprio editor di creazione di un nuovo template. Qui, in seguito, oltre a poterlo utilizzare nella sua interezza, sarà possibile modificare e/o estrapolare solamente una parte di quest'ultimo prima di porlo nella posizione desiderata.
Quindi, ricapitolando, una whitebox non è altro che una copia del template originale che, una volta inserita all'interno dell'editor, si comporta esattamente come se fosse interna al template stesso e non come un componente esterno ad esso.
Un template importato sotto forma di blackbox è totalmente differente dal precedente. Infatti, come si evince dal nome, il template si presenta come una vera e propria "scatola nera", della quale l'utente ne conosce la struttura, poiché ha avuto modo di osservarla nella finestra in cui scegliere il template da importare, ma che non può in nessun modo modificare in quanto all'interno dell'editor di creazione si presenta come un unico blocco denominato con il titolo del template stesso. Le blackbox, al contrario delle whitebox, vengono trattate come componenti esterni al template in quanto in fase di esecuzione di un servizio istanziato a partire da un template che le contiene, come già spiegato nella sezione dedicata all'Instance Executor, vengono considerate come istanze indipendenti da quella principale. Queste possono essere considerate solamente dei marker all'interno del template contenitore, in quanto indicano all'Instance Executor i template importati e la loro posizione all'interno di essi, dato che, come spiegato nella sezione dedicata, quest'ultimo sarà incaricato di reperire tutte le informazioni relative ad ogni blackbox per poi eseguirne la computazione. Di conseguenza possiamo osservare come le whitebox non portano a nessun tipo di problema in quanto una volta importate, vengono considerate a tutti gli effetti parte integrante del template che le contiene.
Le blackbox invece, essendo considerate istanze indipendenti, hanno portato alla modifica del concetto di istanza con la conseguente modifica del modulo incaricato di eseguire la computazione ad esse relative, ovvero l'Instance Executor.
In seguito all'introduzione di questi due nuovi concetti, come già visto nel capitolo 2 , è stato necessario introdurre una nuova BNF relativa al linguaggio utilizzato dalla piattaforma per la creazione e l'esecuzione di servizi personalizzati. Adesso infatti, un Custom Service Template risulta avere una struttura più complessa rispetto alla precedente, in quanto i template possono inglobare più CSTs al proprio interno.

In basso possiamo osservare la rinnovata BNF:

$$E := c \mid DC \mid (E + E) \mid (E - E) \mid (E * E) \mid (E/E) \mid IFTE(C, E, E) \mid BBX$$

$$C := b \mid C \wedge C \mid C \vee C \mid \neg C \mid E > E \mid E \geq E \mid E < E \mid E \leq E \mid E = E \mid E \neq E$$

dove E rappresenta l'espressione relativa al template, C una condizione logica, DC i datastream ad esso assegnati,BBX una blackbox,"c" e "b" infine rappresentano rispettivamente costanti e booleani.

Capitolo 4

Implementazione

4.1 Potenziamento dell'espressività del linguaggio per la creazione di servizi

Uno degli obbiettivi principali del lavoro svolto ai fini di questa tesi riguarda il potenziamento dell'espressività del linguaggio adoperato dalla piattaforma per la creazione di servizi nell'ambito del monitoraggio ambientale. Prima di esporre e spiegare nel dettaglio i miglioramenti apportati, analizzeremo mediante un breve excursus le possibilità relative alla creazione di un template che la piattaforma offriva nella versione precedente al mio contributo.
In quest'ultima infatti, la creazione di un template avveniva solamente mediante l'aggregazione attraverso semplici operazioni logiche e/o aritmetiche di dati grezzi relativi alle misurazioni presenti nel database. Da ciò si evince che potevano essere creati template più o meno complessi (come quello in figura 4.1) che però si limitavano all'utilizzo dei dati primitivi presenti nel database, non potendo in nessun modo usufruire della moltitudine di template creati da altri utenti ai fini di creare servizi composti partendo da quelli esistenti.

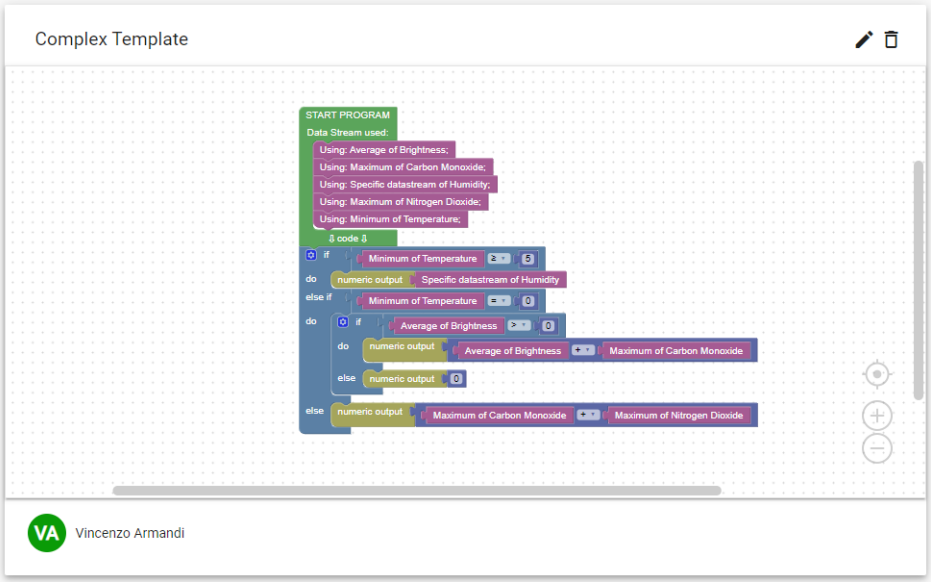


Figura 4.1: Template generico implementazione precedente

Un caso d'uso che mette fortemente in risalto questo problema, o meglio questa limitazione, si verificava quando un utente voleva creare un template che calcolasse la medesima cosa di uno già esistente aggiungendovi un determinato valore. Per esempio infatti, il template in figura 4.2 restituisce il massimo valore di PM10 presente nell'aria in una determinata zona. Dalla descrizione della precedente implementazione possiamo ben osservare che se l'utente ad esempio, avesse voluto creare un template che ritornasse la quantità ottenuta dalla somma del massimo valore di PM10 e il massimo valore di CO2 presente in una determinata zona, avrebbe dovuto creare da zero il suo template non potendo in nessuna maniera utilizzare quello già esistente.

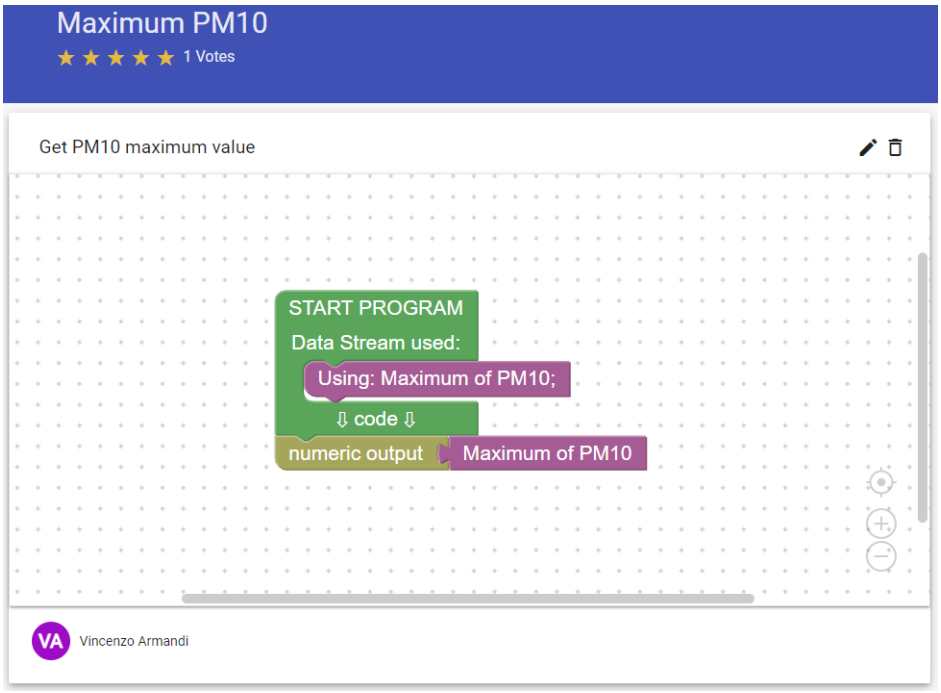


Figura 4.2: Template che restituisce il massimo valore di PM10 in una determinata zona

In seguito al contributo di questa tesi e grazie all'ampliamento delle possibilità offerte dal linguaggio utilizzato da SenSquare, si è ovviato a questa limitazione offrendo all'utente la possibilità di importare template già esistenti nella piattaforma. Infatti, premendo sul pulsante "importa" situato nell'editor visivo che permette la creazione di un nuovo template, si apre una nuova finestra contenente tutti i template presenti all'interno della piattaforma. Qui, l'utente ne può visualizzare un'anteprima premendo sulla card contenente il titolo relativo ad esso (figura 4.3).

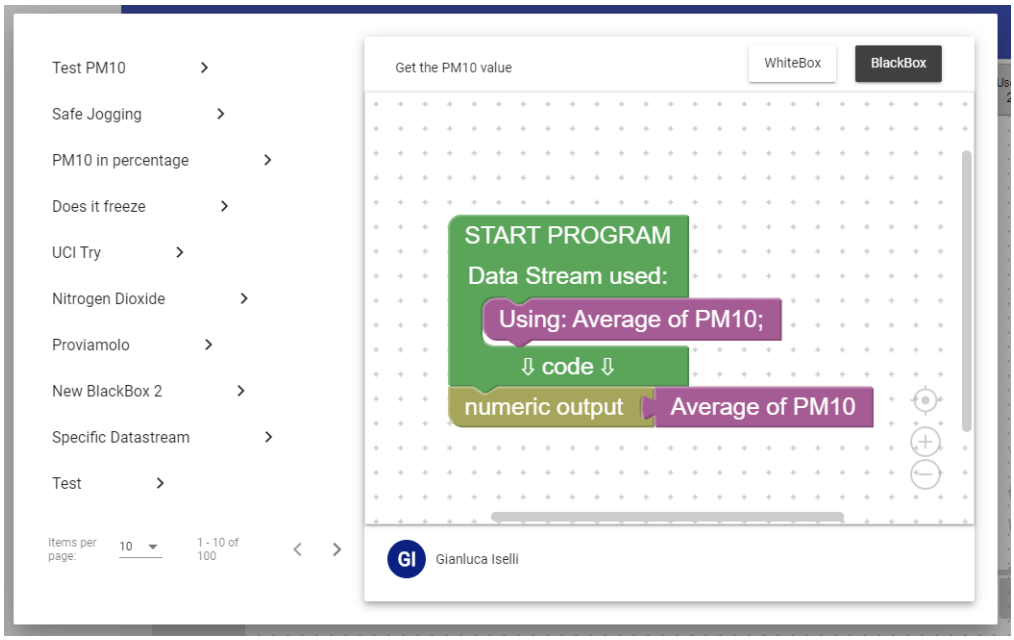
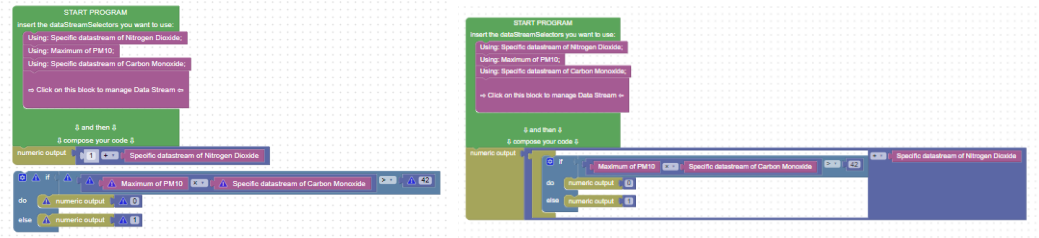


Figura 4.3: Antemprima template da importare

In figura 4.3, in alto a sinistra, di fianco al titolo, possiamo notare la presenza dei due pulsanti che permettono di importare il template all'interno dell'editor attraverso le due omonime modalità. L'utente infatti potrà scegliere di utilizzare il template da lui selezionato sotto forma di Whitebox o di Blackbox. In seguito faremo una panoramica su entrambe le modalità per poi metterle a confronto, evidenziandone le differenze e le relative implicazioni sul concetto di istanza.

4.1.1 Whitebox

La prima modalità di importazione di un template nel processo di creazione di un "Template Composto" è la cosiddetta **Whitebox Import**. Una whitebox non è altro che una vera e propria copia del template selezionato, la cui rappresentazione in blocchi (ottenuta dal framework Blockly) viene inserita all'interno dell'editor di creazione (figura 4.4(a)), dando all'utente la possibilità di posizionare il template in qualunque parte del proprio progetto e persino di modificare la whitebox appena importata se necessario.
Per esempio, possiamo osservare in figura come il template venga posto all' interno di una somma algebrica (figura 4.4(b)) attraverso un nuovo componente introdotto nella toolbox dell'editor,denominato blocco funzione, che funge come una sorta di adattatore per blocchi logici e come vedremo in seguito anche per inglobare una blackbox.



(a) (b)

Figura 4.4: Whitebox import

Oltre alla sua rappresentazione visiva, analizzeremo anche il meccanismo di creazione e di interpretazione del codice python di un template contenente una o più whitebox. Come si evince dallo snippet sottostante (codice 4.1), per ogni whitebox importata e posta nell'apposito blocco sopracitato, nello script verrà creato un metodo identificato dalla dicitura "def" propria del linguaggio di programmazione python, che ha il compito di definire la funzione e dal nome di questa che inizierà con la keyword "getOutput" seguita da un id univoco generato automaticamente dal sistema. Nel caso in cui la whitebox non necessiti del blocco ausiliario, verrà trattata esattamente come se fossero dei blocchi aggiunti dall'utente normalmente all'interno dell'editor e quindi esattamente come accadeva nella precedente implementazione. In seguito, nel codice verranno dichiarate le variabili contenute al suo interno, insieme a quelle eventualmente già presenti nel template e infine verrà invocato il metodo sopracitato nel punto in cui l'utente posizionerà la whitebox importata.

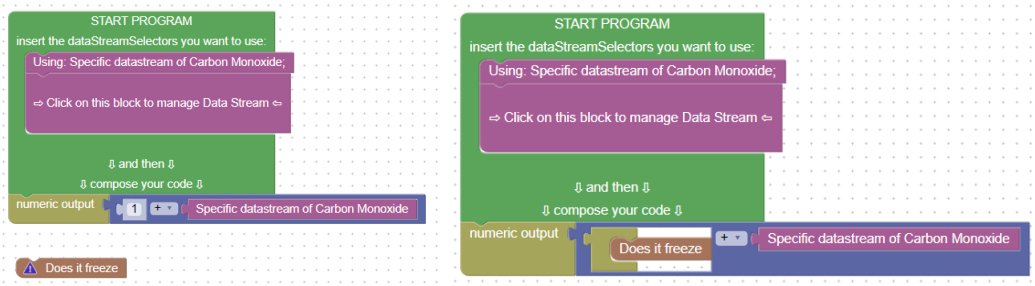
```
1  import sys
2
3  def getOutputZqWizPtI():
4    if (var_PM10_MAX) * (var_COPR_DSID) > 42:
5      return(0)
6    else:
7      return(1)
8
9  var_NO2A_DSID = float(sys.argv[1])
10 var_PM10_MAX = float(sys.argv[2])
11 var_COPR_DSID = float(sys.argv[3])
12
13 print(((getOutputZqWizPtI()) + (var_NO2A_DSID)))
```

Codice 4.1: Script python relativo ad un template contentente whitebox

Per quanto riguarda l'impatto che le whitebox hanno sul concetto di istanza, possiamo affermare che, al contrario delle blackbox, come vedremo in seguito, non vanno ad intaccare la normale esecuzione di un servizio, in quanto nel momento in cui un template contenente una o più whitebox viene istanziato, questo è considerato come un unico servizio, con l'unica differenza che durante il processo di esecuzione da parte dell'Instance Executor verranno eseguiti anche tutti i metodi generati dall'importazione di queste ultime.

4.1.2 Blackbox

La seconda modalità di importazione di un template nel processo di creazione di un "Template Composto" è la cosiddetta **Blackbox Import**. Una blackbox, come suggerito dal nome stesso, può essere considerata una vera e propria "scatola nera", ovvero un blocco che contiene tutte le informazioni relative al template importato ma che, al contrario delle whitebox, non permette all'utente di modificare in alcun modo la sua struttura originale. Nel momento in cui viene cliccato l'omonimo pulsante nella finestra in cui viene visualizzata l'anteprima del template da importare, verrà inserito nell'editor di creazione un nuovo blocco (figura 4.5(a)), introdotto nella piattaforma in seguito al contributo da me apportato, che svolge il compito di identificare il template importato sotto forma di blackbox e permette all'utente di disporlo a suo piacimento all'interno del programma attraverso il blocco funzione presentato nel paragrafo relativo alle whitebox (figura 4.5(b)).



(a) (b)

Figura 4.5: Blacbox import

Come già fatto per le whitebox, analizzeremo il meccanismo di creazione e interpretazione del codice python anche per quanto riguarda template composti contenenti blackbox. Possiamo osservare come nello snippet sottostante (codice 4.2) per ogni blackbox importata verrà creata una nuova variabile il cui nome inizia con la keyword "template" seguita dal suo id univoco reperito direttamente dal database, che sarà posta nel punto corrispondente alla posizione in cui l'utente ha inserito il template in fase di creazione all'interno dell'editor visivo.

```
1  import sys
2
3
4  var_COPR_DSID = float(sys.argv[1])
5  var_NO2A_MAX = float(sys.argv[2])
6
7  if (var_NO2A_MAX) >= 5 and (var_NO2A_MAX) <= 10:
8    print(((template23) + (var_COPR_DSID)))
9  else:
10   print(((var_NO2A_MAX) + (template13)))
```

Codice 4.2: Script python relativo ad un template contentente blackbox

A differenza di ciò che accadeva con template composti contenenti whitebox, in questo caso non verranno dichiarate all'interno dello script le variabili utilizzate dalle blackbox, in quanto queste, come già spiegato nella sezione dedicata, verranno considerate come istanze-figlie e di conseguenza saranno a tutti gli effetti istanze indipendenti le cui variabili verranno reperite al momento dell'esecuzione dall'Instance Executor dal database mediante l'utilizzo di opportune query.
Pertanto come ampiamente discusso nella sezione relativa alla modalità di esecuzione dei servizi, utilizzando blackbox, si va a stravolgere il concetto di istanza così com'era inteso precedentemente al lavoro di tesi.

4.1.3 Template composti ottenuti mediante aggregazione di blackbox e whitebox

Come abbiamo avuto modo di osservare nelle sezioni precedenti, le due modalità descritte si basano su concetti estremamente diversi e che a loro volta hanno un impatto differente sul concetto di istanza. Nonostante la loro diversità, questi possono essere utilizzati contemporaneamente e più di una volta nella finestra di creazione di un nuovo template con il fine di permettere all'utente una maggiore personalizzazione, conferendo al linguaggio una più potente espressività. Possiamo osservare in figura 4.6 un esempio di template composto con un grado di complessità notevole data la presenza di diversi template importati sotto forma di whitebox, e soprattutto considerando il notevole numero di blackbox inglobate. Inoltre, nello snippet in basso (codice 4.3) possiamo notare come il codice python relativo al template composto sia ottenuto dalla combinazione delle due tecniche di generazione descritte nelle sezioni riguardanti le due modalità di importazione.

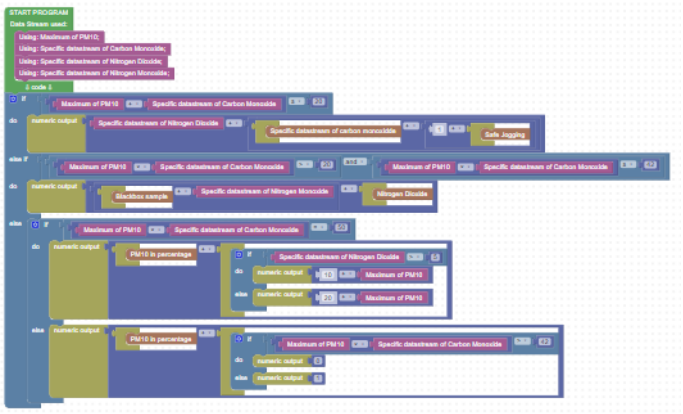


Figura 4.6: Esempio template complesso contenente svariate whitebox e blackbox

```
import sys

def getOutputjwcebYNDqogp():
  if (var_NO2A_DSID) > 5:
    return((10 + (var_PM10_MAX)))
  else:
    return((20 + (var_PM10_MAX)))

def getOutputZqWizPtI():
  if (var_PM10_MAX) * (var_COPR_DSID) > 42:
    return(0)
  else:
    return(1)

var_PM10_MAX = float(sys.argv[1])
var_COPR_DSID = float(sys.argv[2])
var_NO2A_DSID = float(sys.argv[3])
var_NOXO_DSID = float(sys.argv[4])

if (var_PM10_MAX) + (var_COPR_DSID) <= 20:
  print(((var_NO2A_DSID) + ((template153) + (1 + (template13)
   ))))
elif (var_PM10_MAX) * (var_COPR_DSID) > 20 and (var_PM10_MAX)
    * (var_COPR_DSID) <= 42:
  print((((template184) + (var_NOXO_DSID)) + (template25)))
else:
  if (var_PM10_MAX) * (var_COPR_DSID) == 50:
    print(((template16) + (getOutputjwcebYNDqogp())))
  else:
    print(((template16) + (getOutputZqWizPtI())))
```

Codice 4.3: Script python relativo ad un template complesso

4.2 Miglioramenti apportati ai concetti di Template e Istanza

Come anticipato precedentemente, un ulteriore contributo apportato durante il lavoro di tesi consiste nel rendere la piattaforma ancor più "user-centered". Di conseguenza sono state introdotte alcune funzionalità aggiuntive con il fine di garantire agli utenti un ruolo ancora più centrale nel processo di creazione ed esecuzione di un servizio.

4.2.1 Template

Innanzitutto, è stata introdotta la possibilità di eliminare un template creato in precedenza da parte dell'utente proprietario premendo il pulsante identificato dall'icona di un cestino come in figura 4.7. Da questo si evince che verrà effettuato un controllo incrociato tra l'utente che ha effettuato la richiesta e il proprietario del template e, se questi corrispondono, si procederà alla definitiva eliminazione di quest'ultimo dalla piattaforma. In seguito si è scelto di conferire all'utente la possibilità di modificare un template da lui precedentemente creato. Proprio come per l'eliminazione, per procedere alla modifica, basterà premere il corrispondente pulsante identificato dall'icona di una matita (figura 4.7).

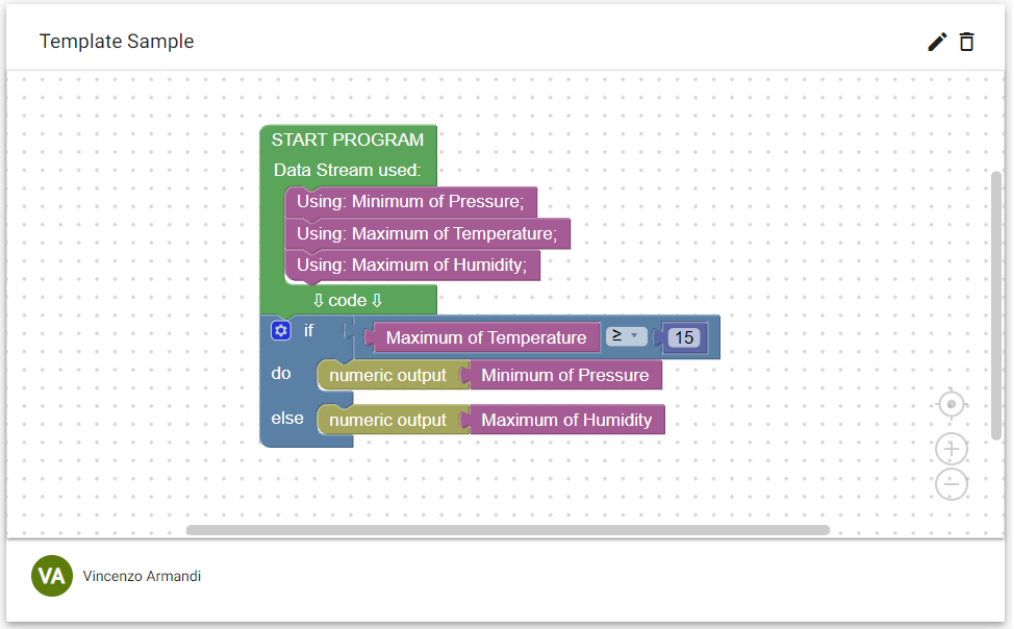
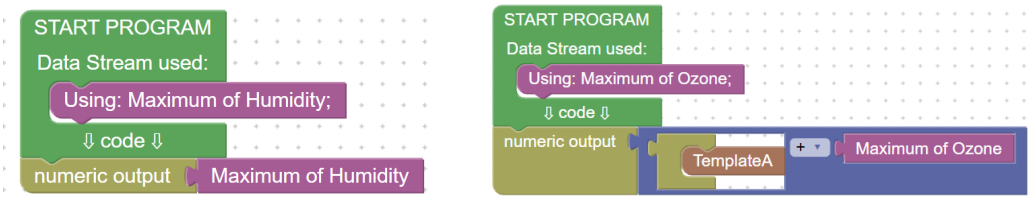


Figura 4.7: Schermata dettaglio Template

Inoltre, è stato implementato un metodo booleano attraverso il quale, prima di rendere effettive le modifiche apportate dall'utente viene controllato ricorsivamente per tutte le blackbox importante, se queste o quelle inglobate all'interno di esse (considerando tutti i livelli di annidamento) utilizzano il template che si sta modificando. Praticamente, si crea un grafo contenente tutti i template coinvolti e, attraverso una visita in ampiezza, si controlla se al suo interno è presente almeno un riferimento a quello che l'utente sta cercando di modificare. Se il metodo in questione, al termine della sua esecuzione, ritorna "False", si procederà a salvare le modifiche all'interno del database. Viceversa, se il valore di ritorno corrisponde a "True", l'utente verrà notificato dal sistema potendo così effettuare le dovute modifiche per rendere legittimi i cambiamenti ad esso apportati.

Nella figura 4.8 possiamo vedere due semplici template che per comodità, ai fini dell'esempio, denomineremo rispettivamente TemplateA e TemplateB. Il primo (figura 4.8(a)) calcola semplicemente il massimo valore di umidità presente nell'aria in una determinata zona. Il secondo invece (figura 4.8(b)), utilizza il templateA sotto forma di blackbox in modo da ritornare la somma del valore calcolato da questo e della massima misurazione di ozono in una specifica località.



(a) (b)

Figura 4.8: Templates di esempio

Per mostrare l'effettivo funzionamento del metodo sopradescritto, considereremo il caso in cui un utente stia modificando il templateA importando la blackbox corrispondente al templateB come in figura 4.9. Nel momento in cui l'utente andrà a confermare le modifiche, verrà notificato un errore dovuto alla presenza di un template che causa un ciclo infinito. Di conseguenza potrà scegliere di tornare indietro ed effettuare ulteriori modifiche per risolvere il problema o di uscire dall'editor e tornare alla pagina di dettaglio del proprio template.

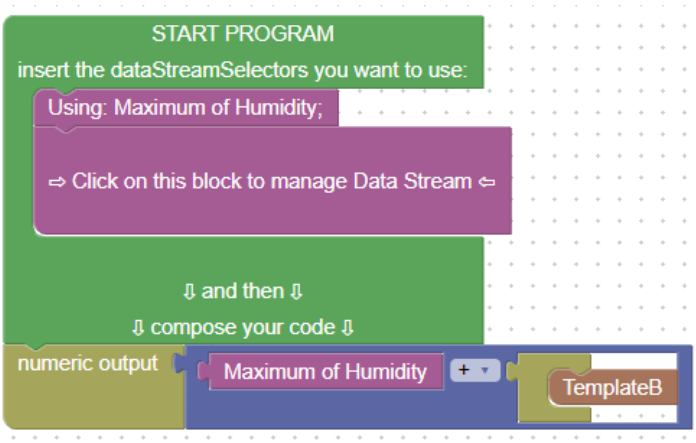


Figura 4.9: Esempio errata modifica

4.2.2 Instance

Così come per i template, si è deciso di dare la possibilità agli utenti finali di eliminare definitivamente e/o modificare anche servizi istanziati in precedenza. Pertanto è stata introdotta la possibilità di eliminare un servizio da parte dell'utente che lo ha istanziato.
Quindi, con la stessa modalità utilizzata per l'eliminazione di un template, verrà effettuato un controllo incrociato e se positivo si potrà procedere alla sua definitiva eliminazione attraverso l'apposito pulsante (figura 4.10).
Data la possibilità di modificare i template, è sorta l'esigenza di permettere all'utente di aggiornare anche un servizio già istanziato, in quanto potrebbero essere state effettuate modifiche rilevanti al template che ne influenzerebbero il corretto funzionamento.
Per esempio, l'utente potrebbe decidere di modificare il template aggiungendo una misurazione relativa ad uno specifico datastream, la quale porterebbe alla "rottura" del servizio, in quanto la scelta dei datastream specifici avviene in fase di istanziazione. Di conseguenza, attraverso le migliorie apportate, l'utente sarà in grado, premendo sul pulsante di modifica (figura 4.10), di includere i datastream mancanti oltre a poter cambiare l'area geografica di operatività della propria istanza.

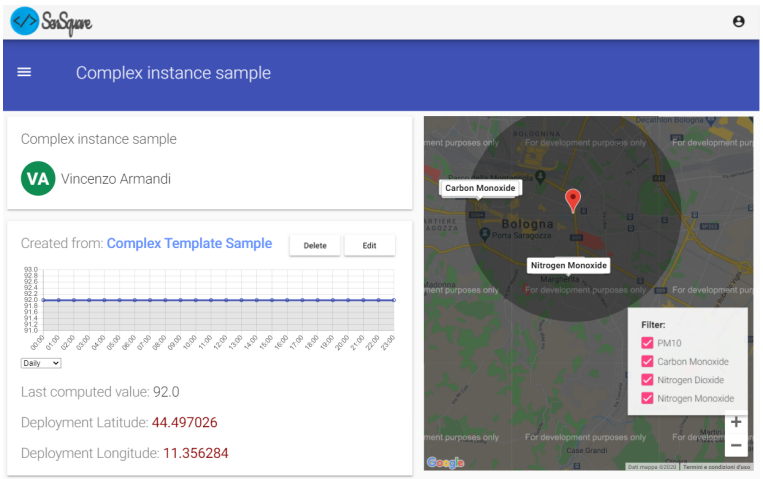


Figura 4.10: Schermata Istanza Generica

4.2.3 Broken Template e Broken Instance

Come brevemente accennato in precedenza, data l'introduzione delle funzionalità di modifica ed eliminazione di template e istanze, si è presentata l'esigenza di introdurre il concetto di template e istanze "rotte", denominati rispettivamente **"Broken Template"** e **"Broken Instance"**, poiché la definitiva eliminazione di template e/o le modifiche apportate ad alcuni di essi avrebbero potuto causare malfunzionamenti di vario genere a tutta la piattaforma.
Partiamo con il dare una definizione di "Broken Template": un template si trova nello stato "broken" se al suo interno contiene blackbox relative a template non più presenti sulla piattaforma. Naturalmente, proprio come avviene per il controllo che precede la modifica di un template, anche in questo caso vengono considerati tutti i livelli di annidamento, non limitandosi quindi alle blackbox contenute dal template in questione, ma effettuando l'ispezione in maniera ricorsiva su di esse.

Quando un template si trova in questo stato è caratterizzato dalla presenza di un triangolo rosso all'interno della schermata ad esso relativa. L'utente potrà quindi posizionarvici sopra il puntatore del mouse per far si che appaia un messaggio contenente l'errore riscontrato e l'opportuna procedura per correggerlo. Oltre ad esso, apparirà anche un pulsante con la dicitura "ripara", attraverso il quale si verrà reindirizzati all'editor di modifica in cui attuare i cambiamenti illustrati nel messaggio precedente (figura 4.11).

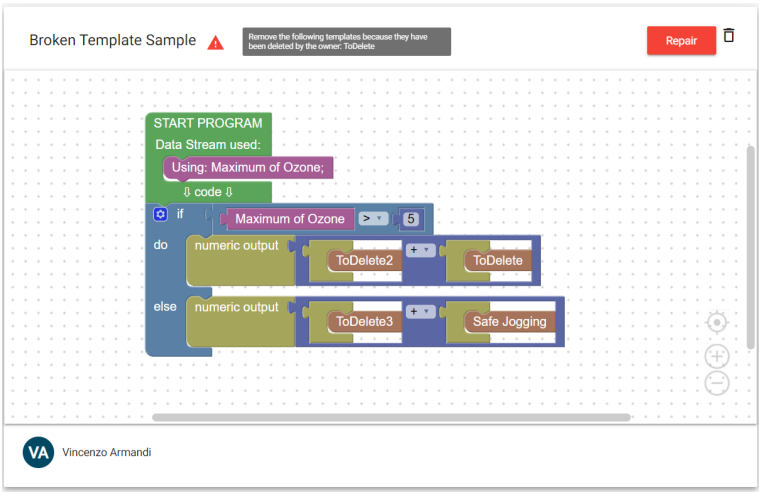


Figura 4.11: Broken Template

Così come per i template, anche i servizi potranno trovarsi nello stato "broken". Un'istanza infatti si trova in questo stato se contiene uno o più template ai quali sono stati aggiunti datastream specifici, diversi da quelli selezionati dall'utente che ha inizializzato il servizio in fase di istanziazione. Di conseguenza, nella schermata relativa al servizio, l'utente visualizzerà l'ormai noto triangolo rosso, attraverso il quale riuscirà a visualizzare il messaggio di errore e la procedura da attuare per risolverlo (figura 4.12).

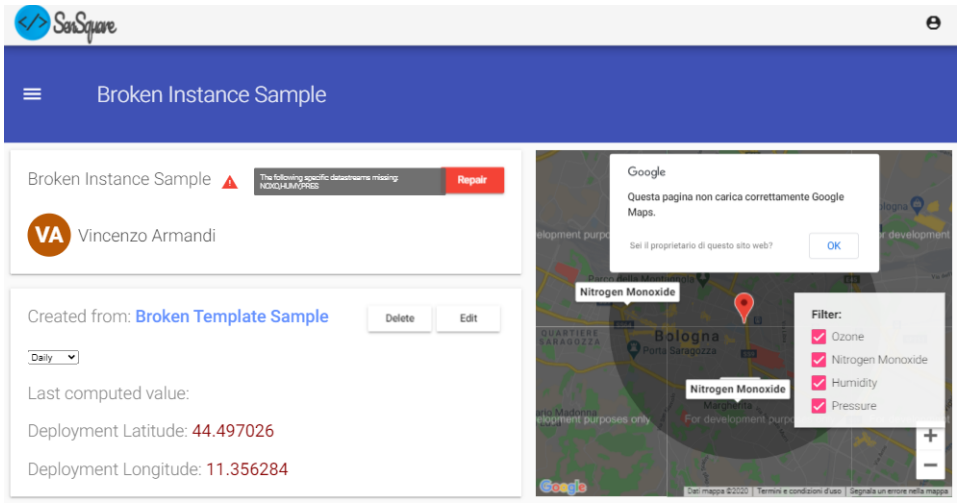


Figura 4.12: Broken Instance

Premendo sul pulsante "ripara", si aprirà un dialog (figura 4.13) contenente tutte le rispettive misurazioni corrispondenti ai datastream mancanti. Qui l'utente potrà selezionare quelli da aggiungere al servizio e una volta confermato, se ha soddisfatto tutte le richieste contenute nel messaggio di errore, visualizzerà l'istanza aggiornata e perfettamente funzionante.

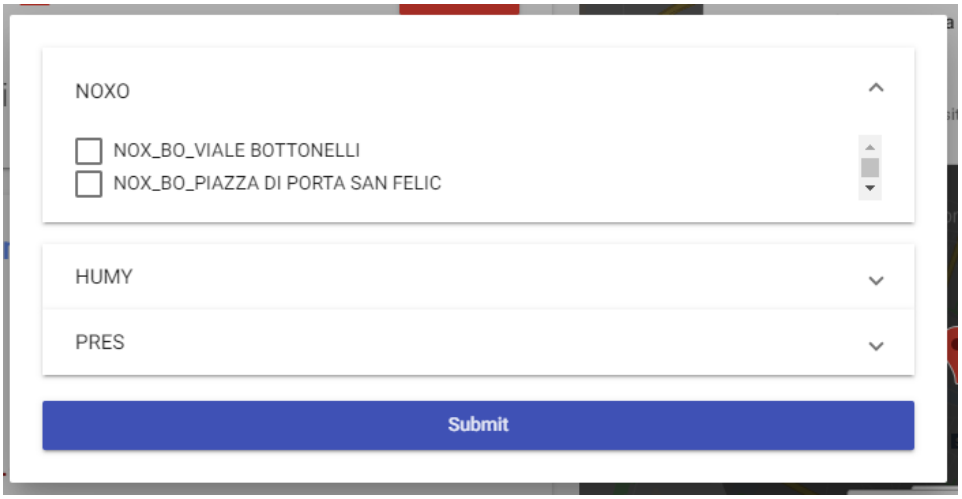


Figura 4.13: Dialog contenente datastream da aggiungere

4.2.4 Migliorie apportate alla struttura a blocchi dei template

Per il corretto funzionamento delle migliorie apportate e discusse nelle sezioni precedenti, è stato necessario intervenire anche sull'implementazione e sulla struttura a blocchi utilizzata all'interno della piattaforma. Innanzitutto, dato il cambiamento apportato alla struttura di un generico template, avendo aggiunto la possibilità di importare whitebox e blackbox, è sorta l'esigenza di modificare il parser che ha il compito di convertire il codice XML corrispondente ad un template nella relativa rappresentazione in blocchi e viceversa, adattandolo alla nuova implementazione. Successivamente, per poter utilizzare i nuovi template importati e per offrire una maggiore espressività al linguaggio, è stato introdotto un blocco funzione che non è altro che un contenitore di blocchi condizionali (if-then-else) e di blackbox. Questo ha il compito di inglobare i blocchi sopra citati e di immagazzinare tutte le informazioni relative ad essi in modo da passarle al chiamante. Infine, data la presenza di blackbox, è stato implementato un apposito blocco per la loro rappresentazione visiva che eredita tutte le informazioni del template importato, ma che non permette in nessun modo all'utente di modificarne il contenuto. In figura 4.14 possiamo vedere un esempio di come i nuovi blocchi si presentano all'interno dell'editor visivo per la creazione di nuovi template.

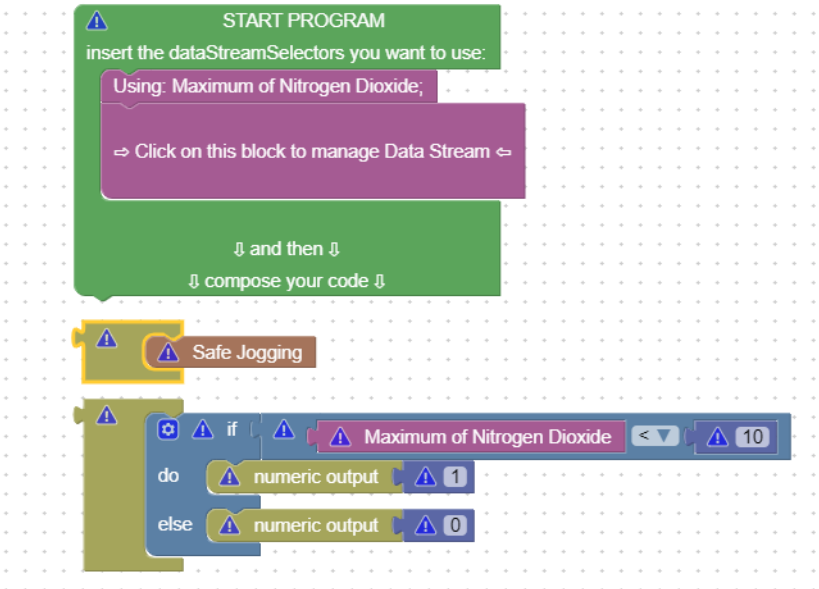


Figura 4.14: Blocchi aggiunti

4.3 Esecuzione parallela di servizi

In seguito alle migliorie apportate e all'introduzione di nuove funzionalità all'interno della piattaforma, gli utenti possono creare template articolati con strutture che potrebbero raggiungere una complessità notevole. Di conseguenza, esaminando i dati raccolti relativi al tempo di esecuzione di istanze più o meno complesse, è nata l'esigenza di rivoluzionare la modalità con cui la computazione relativa ai servizi veniva eseguita per ottimizzare al massimo la velocità di esecuzione.
Infatti, come si può osservare nella tabella sottostante (figura 4.15(a)) con l'implementazione in sequenziale, sulla base di 100 misurazioni effettuate, il tempo medio di esecuzione di un servizio istanziato a partire da un template contenente solamente una blackbox, corrisponde a circa 17 secondi. Confrontando la tabella in figura 4.15(b) relativa ad un servizio istanziato a partire da un template contenente 4 blackbox con quella precedente, possiamo notare come il tempo di esecuzione di un servizio è direttamente proporzionale al numero di blackbox che il template di partenza ingloba al suo interno.

Sequential	
Sum	1740.9884426594
Average	17.409884426594
Median	17.240359544754
Largest	20.910627841949463
Smallest	17.04383611679077
Range	3.8667917251587

(a)

Sequential	
Sum	6352.6791820526
Average	63.526791820526
Median	63.247979283333
Largest	74.52005815505981
Smallest	62.71752905845642
Range	11.802529096603

(b)

Figura 4.15: Dati relativi al tempo di esecuzione di 100 istanze di test con implementazione sequenziale

Di conseguenza, come già detto, è stata implementato un nuovo metodo all'interno dell'Instance Executor che si occupa dell'esecuzione di istanze figlie di un servizio in parallelo. Nell'implementazione precedente le sotto-istanze venivano eseguite su un unico thread. Invece nella nuova è stato utilizzato il modulo "concurrent.futures" del linguaggio python per permettere l'esecuzione di ogni istanza-figlia su uno specifico thread mediante uno specifico ThreadPoolExecutor, in modo da abbattere drasticamente i tempi di esecuzione.
Infatti nella figure 4.16(a) e 4.16(b) possiamo notare come, in seguito al compimento delle modifiche sopradescritte, il tempo di esecuzione di un servizio il cui template di partenza conteneva una blackbox sia passato da una media di circa 17 secondi ad una di circa 2,5 secondi. Osservando la figura relativa al tempo di esecuzione del servizio istanziato a partire dal template contenente 4 blackbox, possiamo inoltre notare che il miglioramento in percentuale ottenuto cresce all'aumentare del numero di blackbox importate all'interno del template. Il reale miglioramento ottenuto, verrà analizzato nello specifico nella successiva sezione dedicata ai risultati ottenuti in seguito al contributo di questa tesi.

Multithreading	
Sum	254.82872128487
Average	2.5482872128487
Median	2.5073611736298
Largest	3.0872929096221924
Smallest	2.3825225830078125
Range	0.70477032661438

(a)

Multithreading	
Sum	320.96676659584
Average	3.2096676659584
Median	3.2207018136978
Largest	3.4324402809143066
Smallest	3.065392255783081
Range	0.36704802513123

(b)

Figura 4.16: Dati relativi al tempo di esecuzione di 100 istanze di test con implementazione sequenziale multithread

Capitolo 5

Risultati

In questo paragrafo verranno analizzati i risultati ottenuti, relativi ai tre punti principali introdotti nella sezione 1.3 , in seguito alle migliorie apportate dal lavoro di tesi.
Il potenziamento dell'espressività del linguaggio utilizzato dalla piattaforma per la creazione di servizi ha portato diverse migliorie all'esperienza utente. Questa risulta essere molto più fluida e meno macchinosa, grazie alla possibilità di riutilizzare template già esistenti.
L'utente infatti non è costretto a creare ogni volta il proprio template da zero, ma può usare questi ultimi come base di partenza o semplicemente per aggiungere funzionalità al proprio servizio in maniera rapida e sicura. Inoltre, la decisione di aver introdotto due modalità tra cui scegliere per importare i template già esistenti porta ad un notevole vantaggio in merito alla stabilità e alla correttezza del servizio. Infatti, attraverso l'utilizzo delle blackbox, non si va a compromettere in alcun modo il suo corretto funzionamento, poiché per definizione, questa struttura non permette la modifica del template importato.
Di conseguenza possiamo notare come un utente meno esperto, soprattutto se non ancora pratico della piattaforma, potrebbe scegliere, per lo meno all'inizio, di utilizzare solamente questo metodo di importazione per evitare di incorrere in errori dovuti alla sua inesperienza e per avere allo stesso tempo un servizio perfettamente funzionante. Inoltre, per garantire una corretta gestione della piattaforma, è stata introdotta la possibilità di eliminare definitivamente template e servizi che non servono più.
Un altro notevole risultato è stato ottenuto in seguito all'introduzione dei concetti di "broken instance" e "broken template", in quanto questi evitano che la piattaforma subisca dei crash quando rileva anomalie e contemporaneamente permettono la segnalazione agli utenti di eventuali errori per far si che vengano corretti.

Infine, con la scelta implementativa di eseguire in parallelo e in multithread le computazioni relative a istanze-figlie, durante l'esecuzione di servizi ottenuti mediante l'istanziazione di template composti, ha portato ad abbattere notevolmente il tempo di esecuzione degli stessi, velocizzando in maniera sostanziale la piattaforma. Nel grafico in figura 5.1, creato mediante uno script python scritto appositamente per questa tesi, possiamo notare come il tempo di esecuzione, oltre ad essere notevolmente diminuito, sia inversamente proporzionale al numero di blackbox, e quindi al numero di sotto-istanze eseguite dal servizio considerato. Il risultato ottenuto dalla nuova implementazione, fa si che la piattaforma sia sempre fluida e performante e che le sue performance migliorino all'aumentare della complessità d'esecuzione di una computazione. Nel grafico sopracitato, è stato considerato lo stesso template per tutte le prove, contenente però un diverso numero di blackbox corrispondenti allo stesso servizio. In seguito, sono state effettuate 100 misurazioni per ognuno di essi con entrambe le modalità di esecuzione (sequenziale e multithread). Possiamo osservare che nel caso del template contenente solamente una istanza figlia è stato ottenuto un decremento percentuale relativo al tempo di esecuzione pari al 85,36%. Il notevole risultato ottenuto si evince, come già detto, dal fatto che all'aumentare del numero di blackbox inglobate da un servizio, il decremento percentuale cresca. Infatti, considerando le misurazioni relative al servizio contenente 4 istanze figlie, possiamo osservare come il decremento percentuale corrisponda al 94,95%.

Figura 5.1: Confronto tra implementazione sequenziale e multithreading dell'Instance Executor

Capitolo 6

Conclusioni

In questo elaborato, è stato inizialmente introdotto il paradigma Internet of Things mediante l'ausilio di alcune delle tante definizioni ad esso attribuibili. Inoltre, sono stati esplorati alcuni campi di applicazione dell'IoT relativi a Smart Home, Smart Citiy e al monitoraggio ambientale. In seguito, sono state presentate le principali caratteristiche di una particolare applicazione dell'IoT, il cosiddetto Internet of Things Collaborativo (C-IoT), che è il concetto alla base di SenSquare, la piattaforma su cui si incentra questo lavoro di tesi. È stata quindi descritta la piattaforma, facendo una breve panoramica sulla sua architettura e definendo le funzionalità che offre agli utenti. Dopo una prima parte introduttiva si entra nel vivo del lavoro svolto, in quanto vengono illustrate le scelte di progettazione e di implementazione delle modifiche e migliorie apportate. A tal proposito, inizialmente, prima di procedere al potenziamento dell'espressività del linguaggio utilizzato all'interno della piattaforma per la creazione di servizi, è stato necessario riadattare la struttura con la quale erano concepiti i template alla base dei servizi e di conseguenza modificare il parser incaricato della traduzione e dell'interpretazione di questi ultimi. Una volta portata a termine questa prima fase, sono state introdotte le due strutture fondamentali alla base della nuova concezione di template, ovvero le Blackbox e le Whitebox. In seguito, è stata studiata e implementata una soluzione al secondo grande problema presente in precedenza all'interno della piattaforma, cioè l'impossibilità da parte dell'utente di modificare o eliminare un template/servizio. Per finire, in seguito ai miglioramenti apportati, è nata l'esigenza di rendere il processo di esecuzione dei servizi più rapido e quindi è stato implementato un meccanismo che sfrutta il parallelismo e la moltitudine di thread offerti dall'architettura che ospita la piattaforma, in modo da abbattere enormemente il tempo necessario alla computazione. Infine, dato il rapido sviluppo del paradigma IoT, più nello specifico di quello C-IoT, è facile notare come ci siano innumerevoli scenari

da approfondire e svariate migliori da apportare alla piattaforma. Ad esempio, in futuro si potrebbero sfruttare algoritmi di intelligenza artificiale e di machine learning per suggerire agli utenti quali template creare in modo da garantire la creazione di servizi realmente utili ai fini di migliorare il loro stile di vita. In conclusione possiamo affermare che il lavoro di tesi ha contribuito ad accrescere le potenzialità, già ingenti, di SenSquare, concentrandosi principalmente sulle esigenze degli utenti finali per migliorare la loro esperienza di utilizzo della piattaforma.

Bibliografia

[1] Antonio Celesti, Antonino Galletta, Lorenzo Carnevale, Maria Fazio, Aime Ĺay-Ekuakille, and Massimo Villari. An iot cloud system for traffic monitoring and vehicular accidents prevention based on mobile sensor data processing. *IEEE Sensors Journal*, 18(12):4795–4802, 2017.

[2] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11):32–39, 2011.

[3] Shantanu Ingle and Madhuri Phute. Tesla autopilot: semi autonomous driving, an uptick for future autonomy. *International Research Journal of Engineering and Technology*, 3(9), 2016.

[4] IoT Council. In need of independent advice and unique context on internet of things?, 2016.

[5] Klaus R Kunzmann. Smart cities: a new paradigm of urban development. *Crios*, 4(1):9–20, 2014.

[6] Paolo Menaspà. Effortless activity tracking with google fit. *British journal of sports medicine*, 49, 09 2015.

[7] F. Montori, L. Bedogni, and L. Bononi. A collaborative internet of things architecture for smart cities and environmental monitoring. *IEEE Internet of Things Journal*, 5(2):592–605, 2018.

[8] F. Montori, L. Bedogni, G. Iselli, and L. Bononi. Delivering iot smart services through collective awareness, mobile crowdsensing and open data. In *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 1–6, 2020.

[9] Federico Montori, Prem Prakash Jayaraman, Ali Yavari, Alireza Hassani, and Dimitrios Georgakopoulos. The curse of sensing: Survey of techniques and challenges to cope with sparse and dense data in mobile

crowd sensing for internet of things. *Pervasive and Mobile Computing*, 49, 07 2018.

[10] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng. Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, 4(1):1–20, 2017.

[11] Steven Ovadia. Automate the internet with "if this then that" (ifttt). *Behavioral & Social Sciences Librarian*, 33(4):208–211, 2014.

[12] Antonio Pintus, Davide Carboni, and Andrea Piras. Paraimpu: A platform for a social web of things. *WWW'12 - Proceedings of the 21st Annual Conference on World Wide Web Companion*, 04 2012.

[13] J. Romkey. Toast of the iot: The 1990 interop internet toaster. *IEEE Consumer Electronics Magazine*, 6(1):116–119, 2017.

[14] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.

[15] Xiangyu Zhang, Rajendra Adhikari, Manisa Pipattanasomporn, Murat Kuzlu, and Saifur Rahman. Deploying iot devices to make buildings smart: Performance evaluation and deployment experience. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 530–535. IEEE, 2016.

[16] A. Zrelli, H. Khlaifi, and T. Ezzedine. Application of damage detection for bridge health monitoring. In *2017 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC)*, pages 42–46, 2017.

Ringraziamenti

Mi sento in dovere di dedicare questa sezione del presente elaborato alle persone che mi hanno supportato durante tutto il mio percorso di laurea.

In primis, un ringraziamento speciale al Prof. Montori per la sua disponibilità e per l'aiuto che mi ha fornito sin dal primo giorno.

Non posso non menzionare i miei genitori, mio fratello e tutti i miei famigliari che da sempre mi sostengono nella realizzazione dei miei progetti. Non smetterò mai di ringraziarvi per avermi permesso di arrivare fin qui.

Ringrazio la mia fidanzata per avermi supportato durante tutto il percorso.

Grazie ai miei colleghi e amici Silvano, Giulio, Antonio e Gian Marco per tutte le giornate passate insieme a studiare in allegria.

Infine vorrei ringraziare tutti i miei amici che nonostante la lontananza mi sono sempre stati vicino.