Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

SVILUPPO DI UN TOOL PER LA CONVERSIONE BIDIREZIONALE DI WEB THINGS IN SERVIZI ARROWHEAD

Relatore:
Dott.
FEDERICO MONTORI

Presentata da:
LUCA COTUGNO

Correlatore:
Dott.
LORENZO GIGLI

II SESSIONE
2021/2022

A chi c'è stato,
A chi c'è,
A chi ci sarà ...

Abstract

Negli ultimi anni una delle tecnologie che ha preso più piede all'interno della vita umana è l'Internet of Things, dispositivi che ti permettono in modo sempre più semplice di migliorare la nostra vita, semplificare azioni che si compiono giornalmente e automatizzare aspetti della vita che ci circondano. Uno degli ambiti che l'Internet of Things sta rivoluzionando è quello lavorativo, ogni azienda con l'avvento dell'Internet of Things si è trovata a dover gestire grandi quantità di sensori e permettere di metterli in comunicazione gli uni con gli altri in modo autonomo, questo ha portato alla nascita di piattaforme Internet of Things ognuna con i propri standard, una conseguenza della proliferazione di queste piattaforme è la nascita di problemi di interoperabilità tra piattaforme diverse che non permette la comunicazione automatizzata tra i dispositivi che ne fanno parte.
Il W3C con la creazione del Web of Things vuole risolvere proprio questi problemi definendo uno standard condiviso da tutti che permetta quindi la comunicazione tra dispositivi in modo facile e automatizzato anche di piattaforme diverse e che quindi vada a diminuire la frammentazione dovuta alla presenza di sistemi autonomi in grado di non comunicare in modo automatico con altri sistemi, questo comporta una limitazione nei possibili usi dell'Internet of Things che utilizzato nel modo corretto può portare a netti vantaggi in qualsiasi settore della vita umana.
In questo documento viene proposta una possibile integrazione del mondo Web of Things all'interno del progetto Arrowhead. Il progetto Arrowhead è un progetto europeo il cui obiettivo è ottimizzare alcuni settori lavorativi attraverso la connessione di cloud locale di dispositivi Internet of Things basati sui principi di una Architettura Orientata ai Servizi (SOA).
La novità di questa proposta è la possibilità di creare una integrazione bidirezionale quindi in grado di trasformare elementi Arrowhead in elementi WoT e anche elementi WoT in elementi Arrowhead ed è stata utilizzata come contributo per la realizzazione del paper "Two-way Integration of Service-Oriented Systems-of-Systems with the Web of Things" [28]. Nella proposta viene definito anche un componente in grado di

interagire in modo semplice con entrambi i tipi di componenti e quindi permettere una interoperabilità tra i due mondi.

Contents

List of Figures

Chapter 1

Introduzione

Il termine Internet of Things è stato coniato per la prima volta nel 1999 dall'ingegnere inglese Kevin Ashton [24] ed indicava un sistema in grado di mappare il mondo fisico attraverso dei sensori e connetterlo alla rete Internet. Negli anni il termine Internet of Things è diventato sempre più utilizzato fino a diventare di uso comune e questo ha portato nella vita di tutti e in qualsiasi ambito quei sensori di cui Ashton parlava nel 1999.
Questi sensori che con il tempo sono evoluti e migliorati oggi possono essere trovati praticamente in qualsiasi oggetto utilizzato in casa o al lavoro rendendo l'Internet of Things uno dei campi dell'informatica più di moda degli ultimi anni.
Per molto tempo l'Internet of Things si è sviluppato senza avere un vero e proprio standard questo ha portato alla creazione di standard proprietari per la creazione di questi sensori e di insiemi di sensori, la creazione di standard diversi ha portato naturalmente a problematiche relative alla comunicazione tra piattaforme Internet of Things diverse e una frammentazione all'interno del mondo IoT.
Il World Wide Web Consortium (W3C) ha provato a risolvere questo problema creando il Web of Things (WoT), uno standard per la creazione di Thing lato software che permette di poter esporre, interagire e trovare Thing in modo facile e veloce ma soprattutto permettere la comunicazione tra Thing di sistemi diversi in modo automatico essendo queste definite seguendo lo stesso standard e permette di risolvere questa frammentazione andando a permettere l'interoperabilità tra varie piattaforme.

Per poter risolvere questa frammentazione c'è bisogno di creare metodi che permettono l'integrazione all'interno di queste piattaforme riuscendo a tradurre i loro standard nei requisiti Web of Things. In questo documento andiamo a proporre una possibile integrazione del mondo Wot all'interno del mondo Arrowhead e viceversa.
Il progetto Arrowhead è un progetto europeo il cui principio è quello di permettere l'interoperabilità di sensori attraverso l'utilizzo di cloud locali basati sull'Architettura Orientata ai Sistemi (SOA) per migliorare cinque settori lavorativi quali la produzione (processo e fabbricazione), costruzioni e infrastrutture smart, mobilità elettrica, produzione di energia e mercato virtuale dell'energia.
In questo documento viene proposta una possibile integrazione bidirezionale del mondo Web of Things con il mondo Arrowhead . La bidirezionalità ci permette di poter integrare Thing all'interno del mondo Arrowhead e integrare Service Arrowhead nel mondo Web of Things. Questa conversione viene permessa da un componente presente all'interno della nostra architettura chiamato WAE che si comporta come un middleware tra i due mondi e ne permette la comunicazione. All'interno di questa proposta oltre la possibilità di convertire elementi di un mondo all'altro c'è la possibilità di interagire con questi elementi attraverso gli standard Web of Things utilizzando un componente chiamato Configurator che permette all'utente tramite input di poter recuperare i valori dei sensori ed eventualmente permette anche la possibilità di modificarli.

Il documento è suddiviso in questo modo:

Nel capitolo 2 vengono descritte in modo più specifico il mondo Internet of Things, il mondo Web of Things ed infine il progetto Arrowhead.
Nel capitolo 3 viene descritta l'architettura del sistema che mette in relazione in due mondi.
Nel capitolo 4 vengono descritte nel dettaglio le 3 fasi che hanno portato all'implementazione di tutti i componenti del sistema e il lavoro che compiono durante l'esecuzione del sistema.
Nel capitolo 5 vengono mostrati i risultati ottenuti dai test che ho eseguito riguardo

la conversione da mondo Web of Things a mondo Arrowhead e viceversa.
Nel capitolo 6 espongo le mie conclusioni riguardo questa proposta ed alcune limitazioni per l'utilizzo del tool nella realtà e quindi anche possibili sviluppi futuri per poter risolvere queste limitazioni.

Chapter 2

Stato dell'arte

2.1 Internet of Things

Prima di poter descrivere il Web of Things c'è bisogno di introdurre l' Internet of Things (IoT).

Con Internet of Things facciamo riferimento a tutti quegli oggetti formati da sensori in grado di raccogliere dati dall'esterno, poterli elaborare e poterli inviare attraverso Internet ad altre apparecchiature che a loro volta possono elaborare e gestire i dati raccolti in base all'utilizzo per cui sono state sviluppate, in altre parole indichiamo tutti quei device smart che negli ultimi anni sono diventati parte integrante della vita dell'uomo e che sono entrati prepotentemente all'interno di qualsiasi settore lavorativo. Le ampie possibilità di utilizzo di questi device ne hanno portato ad una larga espansione nel mondo e secondo le statistiche nel 2020 c'erano oltre 13 miliardi di dispositivi connessi e se ne prevede un aumento tale che nel 2025 si supererà la quota di 30 miliardi[19]. I campi di applicabilità dei dispositivi IoT sono molteplici possiamo utilizzarli nel campo della domotica che negli ultimi anni è diventato uno dei settori principali in cui grandi colossi quali Amazon o Google hanno investito grandi quantità di denaro andando a creare apparecchiature in grado di rendere qualsiasi casa una cosiddetta smart home e quindi migliorare la qualità della vita e semplificare l'utilizzo della tecnologia che sono alcuni degli obiettivi principali

della domotica. Altri settori in cui IoT è diventato parte integrante e lo sarà sempre di più nei prossimi anni sono il settore medico in cui possiamo monitorare costantemente i parametri vitali di un essere umano grazie a device indossabili quali orologi, anelli o collane, il settore automobilistico che grazie a sensori possiamo individuare ed andare a risolvere in anticipo qualsiasi anomalia che potrebbe occorrere ad uno dei tanti componenti che formano un'automobile ed anche tanti altri quali agricoltura, sorveglianza, zootecnia, monitoraggio ecc.
Purtroppo la grande diffusione di questi dispositivi ha portato alla luce anche due grandi problemi relativi alla privacy e alla sicurezza. La diffusione di questo gran numero dispositivi permette alle grandi multinazionali come le già citate Amazon o Google di ricevere grandi quantità di dati relative alla vita di ogni persona che utilizza questi smart device, e come sollevato da alcune associazioni nazionali e mondiali non essendo ben definito come questi dati possano essere utilizzati potrebbero essere utilizzati da grandi aziende o da governi per ottenere ancora più vantaggio economico e di controllo sulle persone andandone a limitare sempre più la loro privacy senza neanche che queste se ne rendano conto.[13]. Altro problema è relativo alla sicurezza poiché la grande diffusione di questi dispositivi non ha portato con se un occhio di riguardo dal punto di vista della sicurezza di tali dispositivi e quindi ci sono stati spesso attacchi hacker che hanno portato al furto di quantità di dati privati da parte di malintenzionati che potrebbero usarli in modo sicuramente non legale[15]. Questi due sono i principali problemi che il mondo dei dispositivi IoT però c'è un grande impegno per la ricerca di possibili soluzioni e vedendo i grandi passi avanti che sono stati fatti negli ultimi anni si prospettano grandi novità che porteranno ad un ulteriore sviluppo e diffusione del mondo IoT all'interno di qualsiasi ambito della vita umana lavorativa e non.

2.2 Web of Things

Con Web of Things (WoT) [21] indichiamo lo standard introdotto dal World Wide Web Consortium (W3C) [23] il cui obiettivo è quello di permettere l'interoperabilità e l'usabilità tra piattaforme Internet of Things andando a preservare gli standard e

le soluzioni IoT già esistenti.
L'architettura Wot [22] si basata su alcuni requisiti che ne permettono l'utilizzo in vari possibili domini d'uso ed è formata da blocchi con ognuno un suo compito specifico che lavorando insieme permettono il giusto funzionamento della piattaforma e questi blocchi sono:

- WoT Thing Description
- WoT Binding Templates
- WoT Scripting API
- WoT Security Privacy and Guidelines

2.2.1 WoT Thing Descrption

La specifica del WoT Thing Description (TD)[18] definisce un modello di informazione basato su un vocabolario e su una semantica basata sul formato JSON che permette di avere un modo per descrivere una Thing che sia Human-Readable quindi interpretabile da un umano ma anche machine-understandable quindi riconoscibile da una macchina, in più l'utilizzo del JSON ne permette l'integrazione con i Linked Data attraverso l'utilizzo del JSON-LD per permettere elaborazione semantica dei metadata. Una Thing Description (TD) permette di descrivere una istanza di una Thing attraverso metadata generici quali nome, id ed una descrizione in più possono esserci relazioni con altre Thing o documenti. Altri metadata importanti all'interno di un TD sono quelli che definiscono il modello di interazione con la Thing, quelli che ne definiscono i possibili modi per comunicare con la Thing e infine i metadata che ne definiscono la sicurezza. Solitamente la TD viene creata e hostata dalla thing stessa e quindi ottenuta al momento della scoperta oppure può essere hostata all'esterno per esempio all'interno di directory apposite.
I due benefici principali del WoT Thing Description sono la possibilità di comunicare direttamente tra due dispositivi all'interno del Web of Thing e soprattutto la presenza di uno standard unificato che permette la descrizione di una Thing e i possibili modi con cui possiamo ottenere i suoi dati.

2.2.2 WoT Binding Templates

Il mondo IoT utilizza un'ampia varietà di protocolli per la comunicazione e l'accesso ai dispositivi non essendoci un protocollo appropriato per qualsiasi contesto. WoT Binding Templates [16] definisce una raccolta di modelli di metadata che forniscono informazioni su come interagire con le varie piattaforme IoT. Un dispositivo che vuole interagire con il dispositivo dovrà implementare quindi uno dei modelli contenuti e in seguito poter comunicare con il dispositivo cercato. I metadata riguardano 5 dimensioni

- IoT Platform

  Le piattaforme IoT spesso richiedono modifiche proprietarie e quindi l'aggiunta per esempio di header HTTP specifici o opzioni COAP

- Media Type

  Le piattaforme IoT differiscono spesso per il formato di rappresentazione usato per scambiare dati

- Transfer Protocol

  Lo schema URI della destinazione contiene informazioni per identificare il protocollo di trasferimento utilizzato, ad esempio HTTP attraverso http: o CoAPS attraverso coaps:

- Subprotocol

  Alcuni protocolli di trasferimento possono avere estensioni che non possono essere riconosciuti dallo schema URI e quindi devono essere dichiarati esplicitamente

- Security

  I meccanismi di sicurezza possono essere utilizzati in vari livelli della comunicazione e il loro uso permette di rendere la comunicazione sicura

2.2.3 WoT Scripting API

Il Wot Scripting API [27] è un blocco opzionale il quale contiene script simili alle API dei browser Web i quali definiscono il comportamento delle Thing, dei Consumers e degli Inermediaries. L'utilizzo di questo blocco permette il riutilizzo di script che vengono eseguiti in un sistema a runtime con il beneficio di migliorare la produttività e ridurre i costi di integrazione. La specifica del WoT SCripting API definisce strutture ed algoritmi che permettono di scoprire, recuperare, consumare, produrre ed esporre Thing Description.

2.2.4 WoT Security Privacy and Guidelines

La sicurezza all'interno del mondo WoT è uno degli aspetti che più deve essere discusso e migliorato. Questo blocco fornisce indicazioni sulla sicurezza e sulla privacy riguardo tutti i livelli[14].

2.3 Piattaforme IoT

Con lo sviluppo e la diffusione dell'Internet of Things molte aziende si sono trovate a dover gestire e collegare tante Thing diverse con endpoint diversi per poter ottimizzare il proprio lavoro e le loro risorse. Questo bisogno ha portato allo sviluppo di piattaforme IoT in cui sono presenti collezioni di Thing spesso in cloud che forniscono funzionalità di un infrastruttura web per supportare soluzioni IoT di base o avanzate. Questa diffusione di piattaforme ha portato anche però problemi per l'interoperabilità tra di esse poiché ogni piattaforma potrebbe avere uno standard diverso che potrebbe essere incompatibile con altre piattaforme. Uno degli obiettivi dello standard WoT è proprio unificare queste piattaforme e renderle tutte in grado di comunicare l'une con le altre. Un esempio di queste piattaforme è proprio quella creata attraverso il progetto Arrowhead, la quale attraverso questa tesi voglio definire un approccio da utilizzare per renderla compatibile con il mondo WoT.

2.4 Arrowhead Framework

Il progetto Arrowhead [12] nasce con l'intento di migliorare l'efficienza e la flessibilità di cinque settori quali produzione,edifici e infrastrutture smart, mobilità elettrica e mercato virtuale dell'energia attraverso l'utilizzo di "system of system" automatizzati. L'obiettivo è quello di consentire l'automazione attraverso dispositivi in rete riuscendo a consentire l'interoperabilità e l'integrabilità dei servizi forniti da questi dispositivi. Secondo i fondatori del progetto l'automazione cooperativa resa possibile attraverso l'Internet of Things e le architetture Service Oriented sono la chiave per affrontare le sfide sia energetiche sia competitive che stiamo affrontando nel mondo del lavoro e quindi abbiamo bisogno di un interazione tra produttori e consumatori di energia, tra macchine e sistemi,tra persone e sistemi ecc.

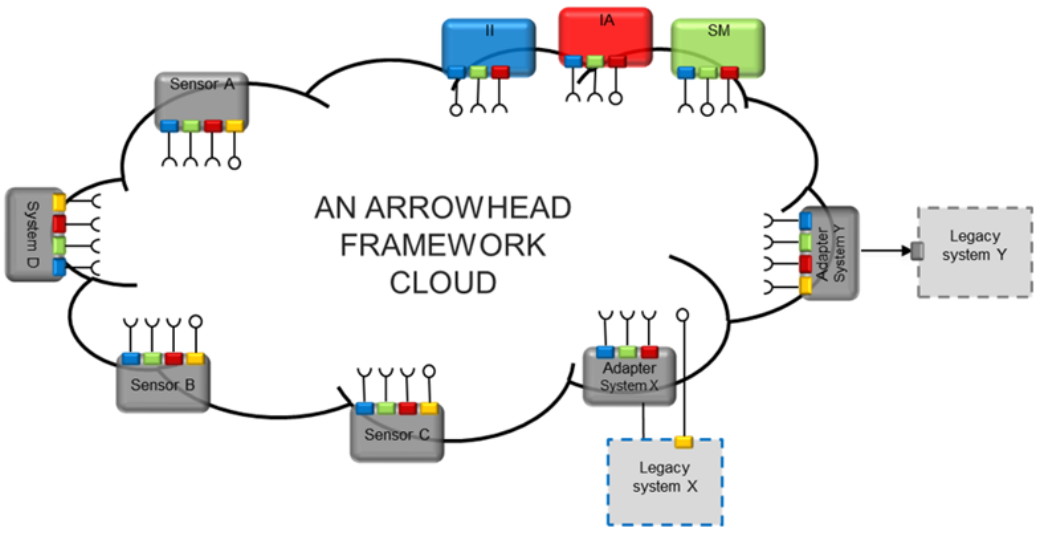

Figure 2.1: Esempio di un local cloud

Il framework Arrowhead [11] quindi affronta il problema dell'automazione basata su IoT, l'approccio si basa sull'idea di cloud locali automatizzati basati sul paradigma SOA ovvero sull'architettura orientata ai servizi andando a soddisfare 3 proprietà importanti:

- Loose Coupling

  In italiano può essere tradotto come accoppiamento lasco ed indica che uno scambio di servizi non è supervisionato, che i servizi sono distribuiti su più dispositivi e che un sistema è responsabile per le informazioni che contiene e decide lui con chi condividerle

- Late Binding

  Questa proprietà indica la possibilità di ottenere le informazioni in qualsiasi momento andando a collegarsi con la risorsa cercata

- Lookup

  Questa proprietà indica che i servizi devono registrare e pubblicare gli endpoint quindi come raggiungerli e devono essere in grado di cercare gli endpoint dei servizi che hanno bisogno

Altre proprietà importanti da fornire per l'automazione e sono relative al cloud, alla scalabilità, alla sicurezza ecc.

2.4.1 Architetture Arrowhead

Come già detto in precedenza il Framework Arrowhead si basa sul concetto di cloud locali i quali devono soddisfare alcuni requisiti che permettono l'ottenimento dei dati in tempo reale, la sicurezza e l'affidabilità dei dati e l'automazione delle funzionalità.

Per avere un cloud Arrowhead locale sono richiesti tre sistemi centrali obbligatori come raffigurato in figura 2.2 e questi sono:

- Service Registry

  Permette di pubblicare le istanze dei Service al fornitore e consente ad un consumatore di trovare l'istanza del Service di cui ha bisogno

- Orchestrator

  Consente al fornitore di determinare quali consumatori di Service accettare

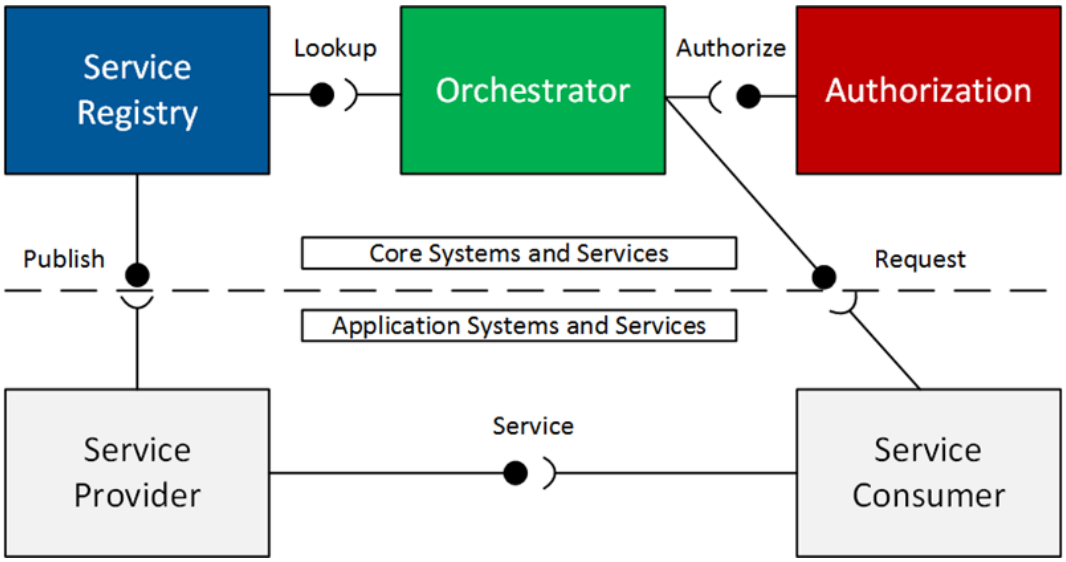

Figure 2.2: Architettura Arrowhead

- Authorization

  Permette il controllo remoto di quale istanza di un Service un consumatore può consumare

Ci sono già alcune proposte per l'integrazione del Framework Arrowhead all'interno del mondo WoT come proposto all'interno del paper "Discovering Web Things as Services within the Arrowhead Framework" [17] in cui viene presentata un'architettura in cui un sistema WoT autonomo è integrato all'interno di cloud locale Arrowhead. La proposta iniziale presenta un'architettura a tre livelli un livello fisico, un livello WoT ed un livello Arrowhead. Un nuovo componente importante è WAE presente all'interno del livello WoT il quale si comporta come intermediario e comunica direttamente con il livello Arrowhead e quindi con il Service Registry del cloud locale. WAE esegue un loop alla ricerca di nuove Thing all'interno del sistema WoT e le registra come service all'interno del Service Registry. In questo modo le Thing del sistema service possono essere consumate sia da consumer Arrowhead sia da consumer del sistema WoT. Questa tesi partendo proprio dal paper ne presenta una nuova proposta in cui non solo un sistema WoT è integrato all'interno del cloud

locale ma anche i service che sono contenuti all'interno del cloud vengono resi thing che sono quindi utilizzabili all'interno del mondo WoT.

2.5 Motivazioni

Le motivazioni che mi hanno spinto a presentare questa tesi sono dovute alla voglia di migliorare la proposta del paper [17] già citato in precedenza quindi migliorare l'integrazione tra il mondo WoT e il progetto Arrowhead e quindi avere un tool non più monodirezionale come descritto nel paper che porta all'integrazione di un sistema WoT all'interno di un cloud locale Arrowhead ma un tool bidirezionale il quale oltre a rendere una Thing IoT un Service Arrowhead rende un Service Arrowhead in una Thing IoT. Nei prossimi capitoli verrà presentata l'architettura della mia proposta che riprende l'architettura del paper [17] ma leggermente modificata e verranno presentate anche dei test con cui sono andato a misurare il tempo necessario alla conversione da Thing a Service e viceversa di gruppi di 10, 50, 100, 200 Service e Thing.

Chapter 3

Progettazione

L'architettura utilizzata per lo sviluppo di questo tool si basa su quella utilizzata e descritta all'interno del paper [17] da cui è partita questa proposta con naturalmente gli adattamenti tali per poter permettere di lavorare in maniera corretta in base ai requisiti della mia proposta. L'architettura del sistema viene illustrata all'interno della figura 3.1 che rappresenta il diagramma di tutta l'architettura.
L'implementazione e il funzionamento nel dettaglio dei vari componenti verrà spiegato nel prossimo capitolo, in questo capitolo andiamo a descrivere a grandi linee l'intera architettura del sistema andandoci a concentrare su quei componenti principali per il funzionamento dell'intero sistema.

Il sistema deve essere in grado di collegare il mondo WoT con il mondo Arrowhead quindi abbiamo bisogno di un componente che rappresenti il mondo WoT, un componente che rappresenti il mondo Arrowhead e un componente che li unisca e ne permetta la comunicazione. Per quanto riguarda il mondo WoT abbiamo la piattaforma Modron, un framework software innvoativo e versatile per l'acquisizione e la gestione dei dati da sensori a cloud. Nello specifico, il framework si occupa dell'acquisizione dei dati dei sensori dal livello di monitoraggio, la gestione dei sensori, l'archiviazione dei dati distribuiti, la visualizzazione e l'analisi dei dati[10]. Nel nostro caso useremo Modron come una Thing Description Directory (TDD) ovvero come un servizio di directory con un'API definita che consente la registrazione, la

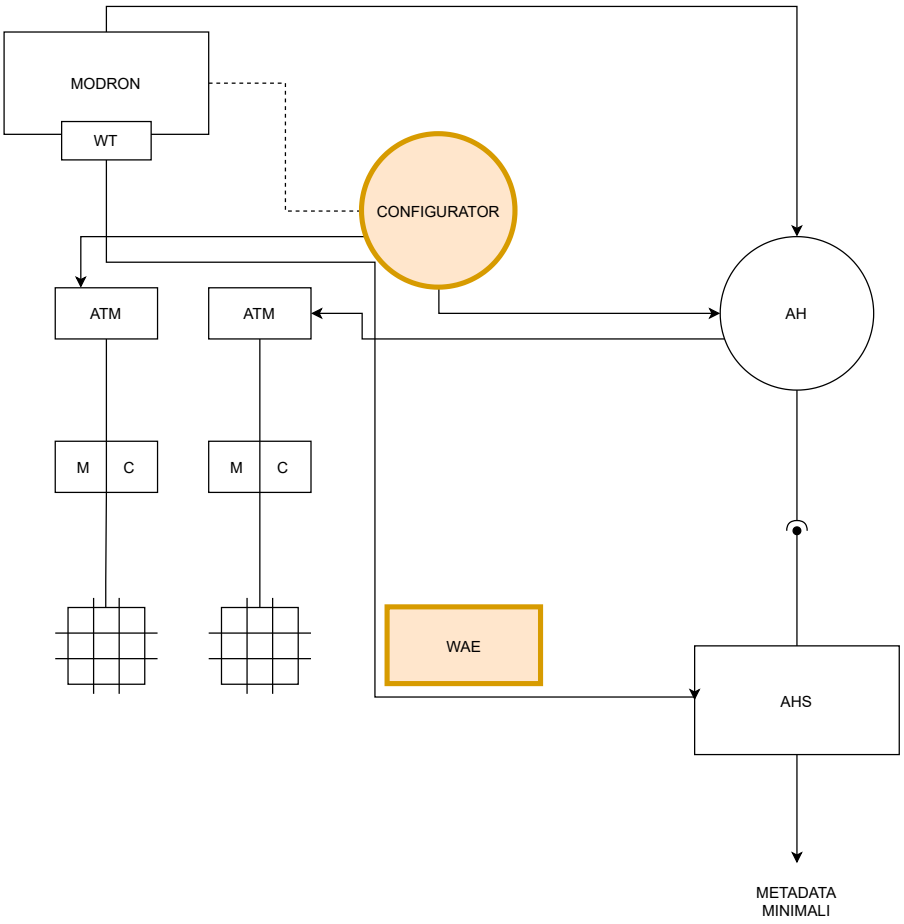

Figure 3.1: Diagramma architettura

gestione e la ricerca in un database di Thing Description [20] e quindi contiene al suo interno delle Thing rappresentate da una Thing Description come specificato dai requisiti del mondo WoT. Per il mondo Arrowhead abbiamo un cloud locale Arrowhead nella figura 3.1 raffigurato con la sigla "AH" e dal suo Service Registry "AHS" l'unico dei tre core system richiesti da Arrowhead con cui andiamo ad interagire direttamente all'interno del sistema. Poi abbiamo probabilmente il componente più importante chiamato WAE evidenziato in arancione nella figura 3.1 che si comporta da ponte tra i due mondi permettendo di registrare in modo automatico e temporiz-

zato le Thing presenti su Modron e viceversa permette anche la conversione di un Service riuscendo a creare una proxy Thing funzionante, esporla e caricarla su Modron, nella mia proposta non vogliamo trasformare tutti i Service in Thing per evitare sprechi computazionali quindi attraverso l'utilizzo dei metadata e di parole chiavi ho pensato ad un modo per distinguere i Service da convertire e differenziarli dagli altri Service che sono sempre caricati all'interno del Service Registry, lo stratagemma per il filtraggio dei Service viene spiegato nel dettaglio nel prossimo capitolo in cui viene spiegato l'implementazione in tutti i dettagli dell'intero sistema.
Nelle figure 3.2 e 3.3 sono raffigurati attraverso dei diagrammi di sequenza le interazioni tra i tre componenti per il corretto funzionamento del sistema.

Nella figura 3.2 vediamo come il WAE registra le Thing presenti su Modron all'interno del Service Registry in loop dopo aver eseguito il login su Modron. Una volta ottenute le Thing viene eseguito un ciclo in cui tutte le Thing attraverso un meccanismo di caching vengono controllate ed eventualmente caricate se sono state aggiunte per la prima volta oppure in base al controllo vengono aggiornate, eliminando il service ormai obsoleto e aggiungendolo di nuovo oppure non si fa nulla e la computazione continua.

Nella figura 3.3 invece vengono mostrate le interazioni che portano alla conversione di un Service Arrowhead in una Thing. Inizialmente il WAE esegue una query per ottenere i Service presenti all'interno del Service Registry, una volta ottenuti e filtrati in base alle nostre esigenze questi vengono trasformati in una Thing e anche qui dopo un controllo attraverso il meccanismo di caching se c'è bisogno vengono aggiunte su Modron andando ad eliminare la Thing oramai obsoleta se la Thing è stata modificata dall'ultima interazione.

Ultimo componente all'interno del sistema è il Configurator anch'esso evidenziato in arancione nella figura 3.1. Il compito del Configurator è quello di recuperare le Thing registrate all'interno del Service indicate attraverso un metadata settato in modo particolare, e permettere all'utente di interagirci attraverso la lettura dei valori delle Properties della Thing ed eventualmente modificarli attraverso input dell'utente.

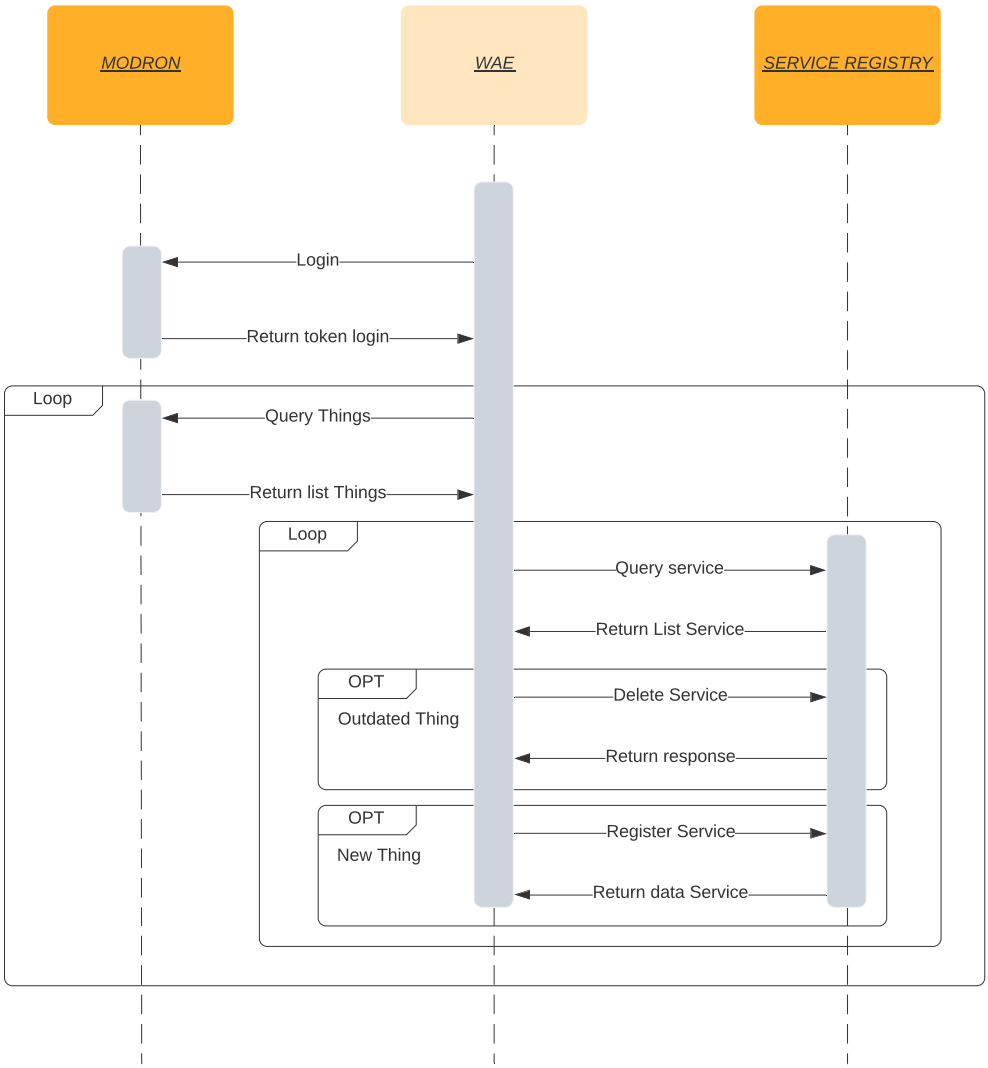

Figure 3.2: Diagramma sequenza Thing

Nella figura 3.4 viene mostrato le interazione del Configurator il quale una volta ottenuta la lista dei Service presenti nel Service Registry e filtrati permette di interagire con le Thing attraverso input da parte dell'utente il quale sarà in grado di leggere i valori delle Properties delle Thing attraverso la ReadProperty ed eventualmente modificarli attraverso l'utilizzo della WriteProperty.

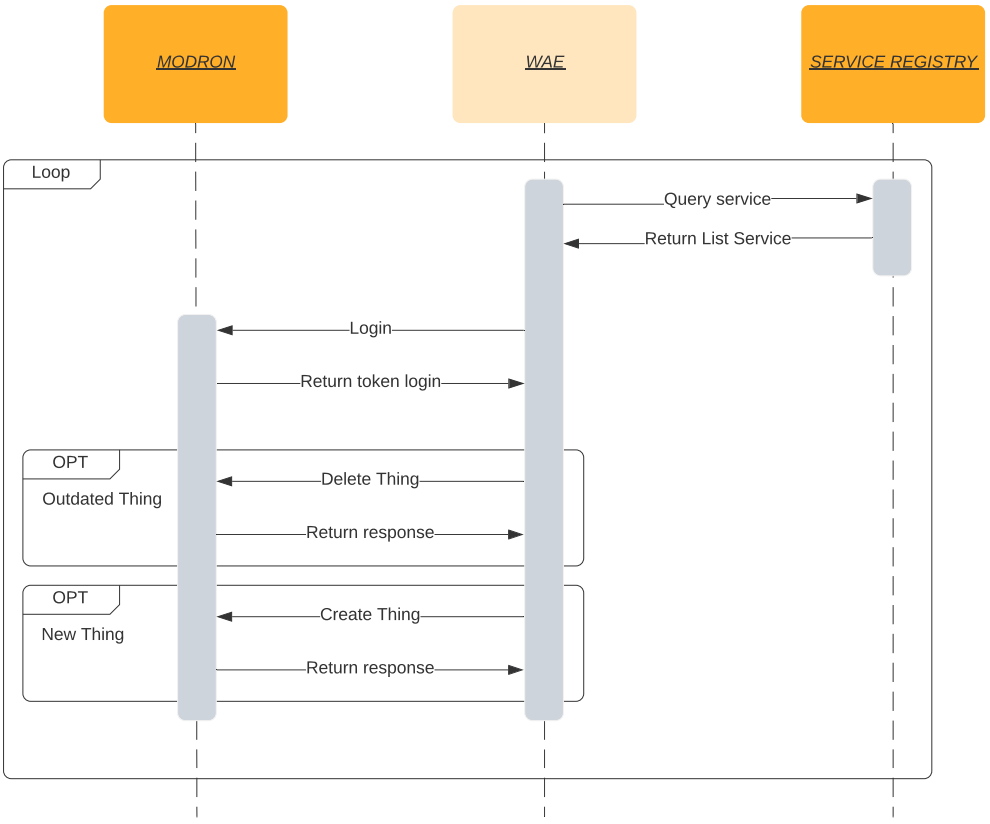

Figure 3.3: Diagramma sequenza Service

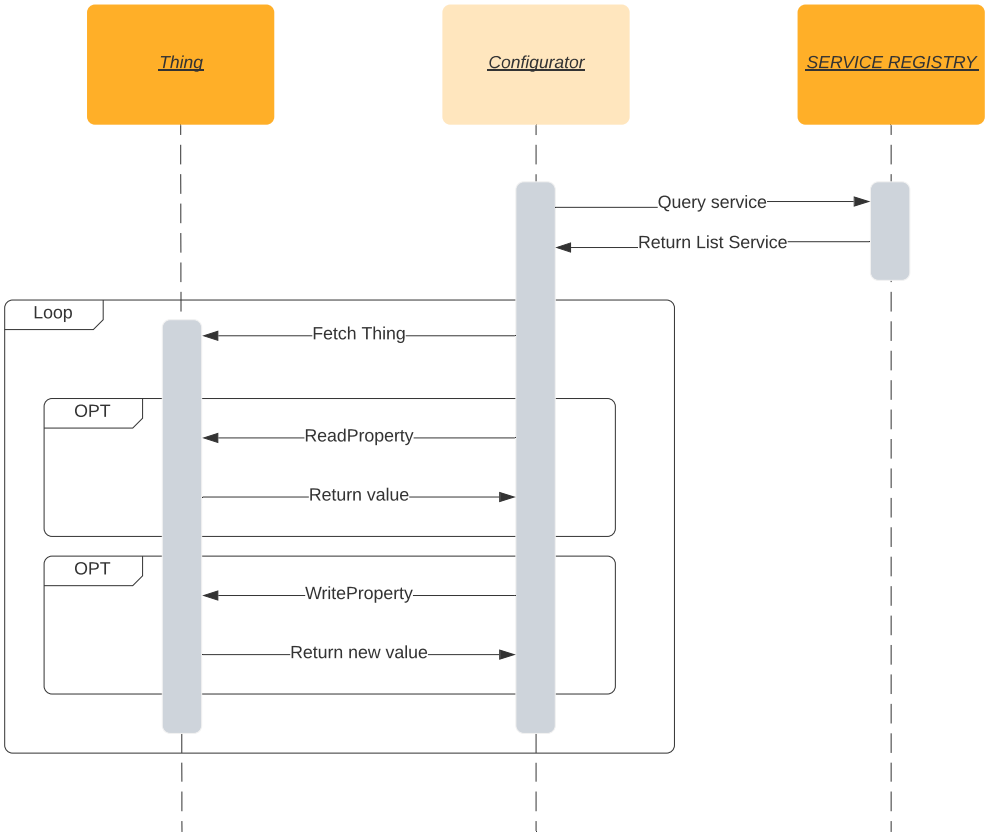

Figure 3.4: Diagramma sequenza Configurator

Chapter 4

Implementazione

Come già spiegato in precedenza l'obiettivo della tesi è quello di creare un tool bidirezionale tra il mondo WoT e il mondo Arrowhead e quindi permettere che una Thing possa essere riconosciuta come un Service all'interno del mondo Arrowhead ed un Service possa essere riconosciuto come una Thing all'interno del mondo WoT. L'implementazione del sistema può essere divisa in tre fasi:

- Nella prima fase ho sviluppato la parte del WAE in grado di registrare le Thing all'interno del mondo Arrowhead.
- Nella seconda fase ho sviluppato il tool chiamato Configurator in grado di interagire con le Thing e quindi poter leggere e/o modificare i valori delle Thing.
- Nella terza fase ho integrato nel WAE la possibilità di poter convertire un Service nel mondo Arrowhead con particolari metadata in una Thing funzionante e in grado di interagirci come nel mondo WoT.

Tutti gli strumenti sviluppati sono script scritti utilizzando linguaggio Javascript eseguiti attraverso node.js che permette di eseguire codice Javascript lato server. Ognuno degli script oltre i normali strumenti forniti di base dal linguaggio Javascript utilizza particolari librerie ottenute attraverso npm (Node Package Manager) [6] un gestore di pacchetti per il linguaggio di programmazione Javascript che utilizza un

database online formato da pacchetti pubblici e privati. L'utilizzo di npm ha semplificato la ricerca, l'integrazione e l'utilizzo delle librerie aggiuntive che mi hanno permesso di non dover implementare alcune funzioni perché già sviluppate e quindi presenti in tali librerie che soddisfavano i requisiti di cui avevo bisogno. Vediamo cosa implementano le librerie utilizzate per poi vedere in seguito in modo approfondito in che modo sono state utilizzate nelle tre fasi.

4.1 Librerie aggiuntive

4.1.1 thingweb.node-wot [9]

Thingweb è un implementazione del modello WoT definito dal W3C e fornisce tutti gli strumenti necessari per poter sviluppare applicazioni in grado di creare e sperimentare con applicazioni WoT e in quanto tale è la libreria principale su cui si basano tutti gli strumenti sviluppati per questa tesi. La libreria implementa il blocco WoT Scripting Api e si basa sul concetto delle Thing esposte e consumate come definito dall'architettura Web of Things, quindi per consumare una Thing, un client crea un risorsa locale eseguita a runtime che può accedere alle Properties, Actions e Events relative ad una Thing esposta solitamente su un server esterno.
Per esporre una Thing abbiamo bisogno:

- La definizione di una Thing Description (TD)
- L'implementazione dell'interfaccia WoT definita nella Thing Description che quindi permetta di soddisfare le richieste di accesso alle Properties, Actions e Events della Thing esposta
- Eventualmente pubblicare la Thing Description per esempio in una Thing Directory in modo da rendere semplice la ricerca della Thing

Possiamo dividere i casi di utilizzo della libreria in tre parti: Consume, Expose e Discovery.

Consume

Consumare una Thing Description ci permette di:

- Leggere un valore di una Property o di un insieme di Properties
- Impostare un valore di una Property o di un insieme di Properties
- Attendere il cambiamento di valore di una Property
- Invocare una Action
- Attendere WoT Events emessi dalla Thing
- Esaminare la Thing Description

Expose

Con esporre una Thing intendiamo la creazione di una Thing locale basandoci da una Thing Description e l'implementazione dei protocolli di connessione che permettono l'accesso a funzionalità di livello inferiore.
Prima dell'esposizione della Thing possiamo modificare la Thing Description andando a:

- Aggiungere o rimuovere una definizione di una Property della Thing
- Aggiungere o rimuovere una definizione di una Action della Thing
- Aggiungere o rimuovere una definizione di un Event della Thing

Una volta esposta possiamo emettere un evento e quindi notificare le Thing in attesa, e registrare dei service handler per richieste esterne quali:

- Recuperare un valore di una Property
- Aggiornare un valore di una Property
- Attendere una modifica del valore di una Property
- Non attendere più la modifica del valore di una Property

- Invocare una Action prendendo i parametri dalla richiesta,invocare l'Action e ritornare il risultato
- Attendere un Event
- Non attendere più un Event

Discovery

Questo caso definisce la ricerca di una Thing e possiamo ricercarla in vari modi, per esempio:

- Cercarla in una rete attraverso una richiesta Broadcast
- Cercarla all'interno del WoT Runtime locale
- Cercarla nelle vicinanze attraverso connessioni NFC o Bluetooth
- Cercarla all'interno di una Thing Directory
- Cercarla filtrando attraverso la Thing Description
- Cercarla filtrando attraverso query semantiche

In più possiamo:

- Fermare un processo di ricerca
- Definire opzionalmente un timeout per il processo di ricerca

4.1.2 isomorphic-fetch [4]

Questa libreria ci permette di sopperire alla mancanza del supporto del "Api fetch()" da parte di alcuni browser quindi lato web ma soprattutto lato server su Node.js che era il mio bisogno primario. L'Api fetch() ci permette di gestire di gestire chiamate asincrone basate sulle promise, specificando l'URL su cui effettuare la richiesta HTTP possiamo gestire l'eventuale risposta positiva e spostarci nel ramo then() altrimenti spostarci nel ramo catch() in seguito a problemi di connessione

Internet o problemi di comunicazione con il server. Naturalmente nella richiesta possono essere dichiarati tutti i metodi gestiti da una normale chiamata HTTP quali GET, POST, HEAD, PUT, DELETE ecc.

4.1.3 js-base64 [5]

La libreria js-base64 ci permette la codifica di un qualsiasi dato binario e quindi ottenere una stringa ASCII di 64 caratteri ottenuta utilizzando l'algoritmo di codifica Base64 ed anche il viceversa quindi decodificare una stringa ASCII ed ottenere l'eventuale valore binario di cui è la codifica.

4.1.4 object-hash [7]

La libreria object-hash è molto simile alla libreria js-base64 differendo però nella possibilità di scelta dell'algoritmo di codifica e quindi non solo Base64 come la precedente e soprattutto permette solo la codifica dell'elemento ma non la decodifica.

4.1.5 fast-xml-parser [2]

La libreria fast-xml-parser permette di lavorare in modo efficiente e semplice con documenti XML. Con XML (eXtensible Markup Language) [26] secondo la definizione di Wikipedia si intende "un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo." Rispetto all'HTML, l'XML ha uno scopo ben diverso: il secondo è un metalinguaggio utilizzato per creare nuovi linguaggi, atti a descrivere documenti strutturati. Mentre l'HTML ha un insieme ben definito e ristretto di tag, con l'XML è invece possibile definirne di propri a seconda delle esigenze.

Esempio di un file XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<utenti>
    <utente anni="20">
```

```
        <nome>Ema</nome>
        <cognome>Princi</cognome>
        <indirizzo>Torino</indirizzo>
    </utente>
    <utente anni="54">
        <nome>Mario</nome>
        <cognome>Rossi</cognome>
        <indirizzo>Roma</indirizzo>
    </utente>
</utenti>
```

Le funzionalità principali della libreria sono quindi:

- La possibilità di validare un dato XML sintatticamente
- Ottenere un oggetto Javascript a partire da un file XML
- Ottenere un file XML a partire da un oggetto Javascript

E ci permette di fare tutto ciò lavorando sia attraverso Node.js, su Browser o CLI e potendo modificare opzioni in modo da ottenere ciò di cui abbiamo bisogno in base alla nostre specifiche.

4.1.6 express [1]

Express è un framework Node.js molto utilizzato per poter sviluppare applicazioni web e mobile in modo semplice e leggero, questo fornisce funzioni per sviluppare API di routing ed impostare middleware per rispondere a richieste HTTP. Express è uno dei quattro componenti dello stack MEAN utilizzato in ampia scala per sviluppare applicazioni web.

4.1.7 prompt-sync [8]

Prompt-sync permette di simulare in ambiente Node un prompt che indica all'utente che il sistema attende un comando per poter continuare o un input dall'esterno senza

aver bisogno di librerie C++ o bash. Le opzioni della libreria permettono di rendere visibile l'input che digitiamo, di poterlo trasformare per esempio con un '*' oppure di nasconderlo del tutto per esempio quando dobbiamo digitare password o codici che vogliamo nascondere.

4.2 Prima Fase

Durante la prima fase è stato sviluppato il componente che permette di registrare all'interno del cloud Arrowhead le Thing che sono all'interno della Thing Directory Modron. Come prima azione l'utente dovrà fare il login all'interno di Modron quindi viene chiesto all'utente attraverso un prompt utilizzando la libreria 'prompt-sync' di inserire l'username e la password dell'account di Modron. Il server Modron è basato sul linguaggio GraphQL [3] cioè un linguaggio di interrogazione e manipolazione dei dati open-source per API e un runtime per soddisfare query con dati esistenti, GraphQL consente agli sviluppatori di ottenere richieste contenenti dati provenienti da più sorgenti, in una singola chiamata API. GraphQL gestisce due tipi di operazioni le query e le mutations, se volessimo confrontarlo con il modello CRUD possiamo dire che le query equivalgono alle read mentre le create,update e delete sono gestite dalle mutations che quindi ci permettono la creazione, modificata ed eliminazione dei dati mentre le quey ci permettono di ottenerli. GraphQL permette di definire i modelli di dati che si vuole ricevere in input e che si vuole restituire in output evitando così problemi vari. Detto questo possiamo dire che una volta ottenuti questi dati viene eseguita una richiesta HTTP utilizzando la libreria 'isomorphic-fetch' e quindi l'API fetch(), questa richiesta sarà composta dal methodo POST e come body avrà i dati inseriti dall'utente schematizzati come una mutation come richiesto dal linguaggio GraphQL. Se i dati sono corretti ci sarà restituito un token che ci autentifica e ci permette di eseguire qualsiasi operazione all'interno del server altrimenti restituiremo un messaggio di errore attraverso il catch() della funzione fetch().
Il componente che ho sviluppato dovrà controllare ciclicamente se ci sono nuove Thing all'interno di Modron. Per questo una volta completato il login viene chiesto all'utente di inserire un valore numerico che equivale all'intervallo di tempo che passa

ogni volta che Modron verrà controllato, quindi avremo che ogni x minuti scelti dall'utente verrà effettuato il controllo sul server Modron.
Una volta ottenuto l'accesso e impostato il timer ogni volta verrà eseguita una query su Modron che restituirà un array contenente le Thing che sono salvate su Modron, ottenendo oltre la Thing Description otteniamo anche altri metadata che però non andremo a usare. Una volta ottenuto l'array per ogni elemento dell'array e quindi per ogni Thing andiamo come prima cosa ad ottenere i dati che ci servono per registrare il Service su Arrowhead, un esempio del dato che serve per registrare il Service è questo qui:

```
{
        "endOfValidity": " 2022-01-01 23:59:59",
        "interfaces": [
          "HTTPS-SECURE-JSON"
        ],
        "metadata": {
          "additionalProp1": "'+ td + '",
          "additionalProp2": "'+ hashTd + '"
        },
        "providerSystem": {
          "address": "'+ address + '",
          "port": '+ port + ',
          "systemName": "modron"
        },
        "secure": "NOT_SECURE",
        "serviceDefinition": "'+ title + '",
        "serviceUri": "'+ serviceUri + '",
        "version": 0
    }
```

Per ottenere questi dati andiamo a trovarli nella TD, i dati più complicati da recuperare sono address, port e serviceUri poichè nella TD avremo un indirizzo completo per esempio del tipo "http://100.100.100.100:8080/counter", quindi andando

a lavorare sulle stringhe e sulla posizione dei caratteri quali "/" e ":" riusciamo a dividere la stringa ed ottenere in questo caso come address "http://100.100.100.100", come port invece "8080" e infine come serviceUri avremo "counter". Oltre questi dati avremo bisogno del title che però riusciamo ad ottenerlo facilmente perchè andremo a prendere il campo title che è già presente all'interno della Thing Description e i due campi all'interno del campo metadata; è stato scelto di aggiungere come primo metadata l'intera Thing Description della Thing che andiamo a registrare come Service mentre come secondo metadata aggiungiamo l'hash dell'intera Thing Description utilizzando la libreria 'object-hash' che ci permette di ottenere l'hash attraverso codifica in Base64 dell'intera TD e quindi ottenere una stringa di 64 caratteri che identifica quella Thing Description, la scelta dell'aggiunta di questa stringa è data dal requisito di avere una sorta di meccanismo di caching.
Il bisogno di avere un meccanismo di caching è dato dalla possibilità che la Thing salvata su Modron venga modificata in seguito alla sua prima registrazione come Service nel mondo Arrowhead ma vogliamo evitare anche che la stessa Thing venga aggiornata anche se non è cambiato nulla questo per evitare sprechi dal punto di vista della memoria computazionale e quindi eseguire solo le operazioni che sono necessarie. Per attuare questo meccanismo di caching una volta ottenuti tutti i dati che abbiamo bisogno della Thing da registrare andiamo ad eseguire una nuova richiesta Fetch() questa volta al Service Registry del cloud locale Arrowhead e tale richiesta sara una query dei Service registrati andandoli a filtrare in base al title della Thing che dobbiamo registrare poichè non possono esserci Service con lo stesso titolo sappiamo che se quella Thing è già stata registrata allora sarà unica e ci verrà restituita dalla richiesta. Una volta ottenuta la risposta andiamo a controllare se l'array di Service restituito è vuoto o contiene il Service cercato, se contiene il Service cercato dobbiamo controllare se è da aggiornare o non è cambiato nulla e qui entra in gioco l'hash della Thing Description andando a confrontarli, se gli hash differiscono vuol dire che la Thing è stata modificata e quindi il Service è da aggiornare allora andiamo ad inviare una richiesta DELETE per eliminare il Service oramai obsoleto e inviamo una nuova richiesta per registrare il nuovo Service aggiornato, altrimenti se gli hash sono uguali il Service è aggiornato e allora possiamo proseguire senza far

nulla. Invece se l'array ritornato dalla richiesta è vuoto vuol dire che la Thing è stata appena aggiunta su Modron quindi sicuramente non è ancora stata registrata sul cloud Arrowhead allora proseguiamo registrandola con i dati che avevamo ottenuto in precedenza e come metadata la TD e il suo hash come stabilito secondo il nostro standard.

4.3 Seconda Fase

Nella seconda fase mi sono occupato dello sviluppo di un componente chiamato Configurator in grado di poter recuperare le Thing registrate come Service all'interno del Service Registry del cloud locale Arrowhead e poterci interagire andandone a recuperare il valore di una Property e eventualmente modificarla.
Il Configurator come prima operazione dovrà recuperare le Thing registrate sul Service Registry di Arrowhead quindi eseguirà una query tramite una fetch() che gli restituirà un array contenente tutti i Service registrati, una volta ottenuti abbiamo bisogno di filtrarli ed ottenere solo quelli che sono stati registrati in seguito alla loro presenza sul server Modron e quindi che abbiano come metadata la Thing Description che poi dovremmo utilizzare. Per filtrarli andiamo a controllare il systemName del Service, durante la fase uno il systemName viene impostato di default come "modron" in questo modo riusciamo solo controllando il systemName ad ottenere i Service di cui abbiamo bisogno. Una volta ottenuti possiamo stampare a video la lista delle Thing definite da un numero e dal loro title, l'utente potrà selezionare la Thing con cui vuole interagire attraverso l'inserimento del numero della Thing tramite un prompt quindi utilizzando la libreria "prompt-sync" che si ripete fin quando non viene digitato un valore numerico corretto in base al numero delle Thing presenti, una volta quindi inserito un valore corretto e scelta la Thing abbiamo bisogno di ottenere l'indirizzo e abbiamo due possibili modi per ottenerlo

1. Recuperarlo attraverso i campi address, port e uri all'interno dei campi del service registrato sul Service Registry

2. Recuperarlo attraverso il campo tdURL presente all'interno della Thing Description

In questo caso è stato scelto di utilizzare il primo metodo ma entrambi i metodi sono validi e non ci sono grosse differenze nel scegliere uno o l'altro.
Una volta ottenuto l'indirizzo dobbiamo interagire con la Thing e per farlo dobbiamo istanziare un servient attraverso la libreria "thingweb.node-wot" che non è altro che un oggetto a runtime che permette l'integrazione con il mondo WoT e permette di eseguire le tre tipologie di operazioni del mondo WoT expose, consume e discovery. Una volta istanziato il servient possiamo attraverso la fetch() interagire con la Thing. Una volta ottenuta la Thing Description possiamo ottenere le Property con cui l'utente vorrebbe interagire. Quindi vengono stampate a video la lista delle Property della Thing con un numero assegnato e attraverso un prompt che anche qui si ripete fin quando il valore inserito non è corretto l'utente seleziona la Property che desidera attraverso il numero assegnato alla Property, una volta quindi scelta la Property il comportamento del Configurator varia in base al valore dei campi ReadOnly e WriteOnly, se la Property è ReadOnly non può essere modificata quindi viene stampato il valore attuale della Property attraverso la funzione readProperty() della libreria "thingweb.node-wot" e viene comunicato all'utente che la Property essendo ReadOnly non può essere modificata dopo di che viene chiesto all'utente se vuole continuare attraverso un prompt e se la risposta è positiva il Configurator visualizza di nuovo a video la lista delle Thing disponibili e l'utente può continuare ad interagire con le Thing che sceglierà. Se invece la Property scelta è WriteOnly viene comunicato all'utente e gli verrà chiesto se vuole modificarla o meno, se si vuole modificarla verrà visualizzato a video il tipo della Property scelta e l'utente potrà inserire attraverso un prompt il valore nuovo della Property che verrà aggiornato attraverso la funzione writeProperty() sempre della libreria "thingweb.node-wot", una volta che viene modificato il valore viene stampato il nuovo valore aggiornato e poi come nel caso precedente viene chiesto all'utente se vuole continuare o meno. Ultimo caso possibile che possiamo avere è quando la Property non è ReadOnly e neanche WriteOnly, in questo caso verrà normalmente stampato a video il valore della Property e in seguito chiesto all'utente se desidera modificarlo, anche in questo caso una volta letto il valore ed eventualmente modificato viene chiesto all'utente se vuole continuare e quindi reiniziare il ciclo oppure concludere l'esecuzione del Con-

figurator. Viene tenuto conto anche del caso in cui all'interno del Service Registry non siano registrati Service con systemName Modron e quindi non avendo Thing con cui interagire verrà comunicato semplicemente all'utente che non ci sono Thing con cui interagire.

4.4 Terza Fase

La terza fase è stata probabilmente la più complicata e mi sono occupato di riuscire a convertire un Service registrato sul Service Registry di Arrowhead in una Thing che sia funzionante e permetta la comunicazione con il Service Arrowhead. Primo problema che ho dovuto affrontare è stato trovare un modo per descrivere il comportamento di un Service che poteva essere un qualsiasi dispositivo IoT che quindi avrebbe potuto non rispettare i requisiti definiti dal W3C con il mondo WoT. Per poter risolvere questo problema la scelta che mi è sembrata la migliore è stato il Web Service Description Language (WSDL) [25].
Con Web Service Description Language indichiamo un un linguaggio formale in formato XML utilizzato per la creazione di documenti per la descrizione di Web Service. Attraverso l'utilizzo di WSDL può essere descritta l'interfaccia pubblica di un Web Service attraverso una descrizione basata su XML che indica come interagire con tale Web Service. Questo documento contiene informazioni riguardo:

1. Cosa può essere utilizzato quindi le operazioni che mette a disposizione il web service

2. Come poter utilizzare tale web service quindi specificando i protocolli di comunicazione ed il formato dei dati in input e in output

3. Dove si trova il web service quindi il suo endpoint solitamente in formato URI dove accedere al web service

Si possono notare facilmente le analogie con il modello definito per il mondo WoT, l'elemento in posizioni 1 nello standard WoT equivale alle Properties e le Actions che una Thing possiede e può eseguire, l'elemento in posizioni 2 equivale ai Bindings che

in una Thing definiscono i protocolli che permettono l'interazione con la Thing e anche i media type per i dati che vengono scambiati, infine l'ultimo elemento equivale al campo tdURL presente all'interno della Thing che indica l'indirizzo dove è presente la Thing.
Una volta trovato il linguaggio da utilizzare ho avuto il bisogno di trovare un modo per indicare che tale Service registrato sul cloud Arrowhead avesse i requisiti per poter essere trasformato in una Thing, per far ciò ho scelto di utilizzare anche in questo caso i campi metadata che compongono il dato salvato sul cloud per ogni Service. Ho deciso di aggiungere un campo metadata chiamato "WSDL" che al suo interno contenesse proprio il WSDL che descrive il comportamento di tale Service e quindi avere un modo per distinguere i Service che possono essere convertiti in una Thing.
Una volta definito la standard per indicare i Service scelti e poterne descrivere il comportamento ho potuto sviluppare il componente in grado di convertirli in Thing funzionanti. Anche in questo caso avevo bisogno di meccanismo di caching per evitare di convertire nuovamente Service già convertiti che non hanno subito modifiche. In questo caso ho deciso di utilizzare un array, ma per spiegarne il funzionamento bisogna prima specificare che quando un Service viene registrato sul Service Registry gli viene assegnato un id numerico univoco che non verrà modificato a meno che il Service non venga deregistrato e registrato nuovamente, quindi ho deciso di utilizzare questo id numerico come indice dove salvare i dati di cui abbiamo bisogno relativi a quel Service. Tali dati sono l'hash del WSDL del Service in questione e visto che anche Modron una volta che aggiungiamo una Thing associa ad essa un id univoco andiamo a salvare tale id all'interno dell'array. Quindi per ricapitolare avremo che preso per esempio il Service con id 1 che viene registrato in Modron con id 2, avremo che nell'array in posizione 1 verrà salvata la coppia formata dall'hash del WSDL di tale Service e dal'id 2 in questo caso. In questo modo riesco a controllare velocemente se il WSDL è stato modificato dall'ultima volta che ho controllato quel Service e quindi bisogna aggiornare la sua Thing e eventualmente riesco a eliminarlo velocemente da Modron avendo l'id della Thing da dover aggiornare.
Una volta definite le scelte implementative possiamo andare a spiegare il comporta-

mento dell'algoritmo. Come prima cosa andiamo ad ottenere attraverso una query tutti i Service registrati sul Service Registry del local cloud di Arrowhead. Una volta ottenuto l'array di Service andiamo a filtrarli tramite la presenza o meno del campo "WSDL" all'interno dei metadata come spiegato in precedenza, fatto ciò abbiamo tutti i Service che soddisfano i nostri requisiti e che possiamo trasformare in Thing, quindi per ogni Service presente all'interno dell'array ottenuto andiamo come prima cosa a controllare se all'interno dell'array usato come meccanismo di caching nella posizione del suo id, se l'array in quella posizione è vuota vuol dire che il Service è nuovo e mai stato visitato prima quindi possiamo trasformarlo in Thing senza altri controlli altrimenti se in quella posizione c'è qualcosa ci sarà sicuramente l'hash del WSDL relativo al Service l'ultima volta che è stato incontrato che quindi possiamo confrontare con il nuovo WSDL se questi coincidono vuol dire che non ci sono modifiche da effettuare e possiamo andare avanti altrimenti se diversi bisogna aggiornare la Thing quindi prima cancellare la Thing da Modron attaverso l'id che avevamo salvato all'interno dell'array di caching e poi creare la nuova Thing e aggiungerla nuovamente su Modron. Una volta che sono stati controllati i Service da aggiornare bisogna come prima cosa trasformare il WSDL del Service in una Thing Description e poi esporre la Thing per poterla caricare su Modron.
Vediamo come poter passare da un documento WSDL ad una Thing Description. Qui di seguito possiamo vedere un esempio di WSDL di un web service che è stato usato per testare l'implementazione che non è altro che un counter formato da un unico valore intero e da due operazioni possibili una "get" che ci permette di ottenere il valore attuale del counter e un "increment" che permette di ottenere di nuovo il valore del counter ma dopo essere stato incrementato. Essendo una proposta la mia è stato scelto di utilizzare un esempio di WSDL non troppo complicato ma sufficiente per poter dimostrare la fattibilità della mia proposta.

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
             xmlns:tns="http://www.example.com/wsdl20sample"
             xmlns:whttp="http://www.w3.org/ns/wsdl/http"
             xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
```

```
                targetNamespace="http://www.example.com/wsdl20sample">
</description>

<!-- Abstract types -->
   <types>
      <xs:schema xmlns="http://www.example.com/wsdl20sample"
                 xmlns:xs="http://www.w3.org/2001/XMLSchema"
                 targetNamespace="http://www.example.com/wsdl20sample"/
        <xs:element name="response" type="xs:int"/>
   </types>

   <interface  name = "Interface" >

    <operation name="get"
             pattern="http://www.w3.org/ns/wsdl/in-out"
             style="http://www.w3.org/ns/wsdl/style/iri"
             wsdlx:safe = "true">
        <output    messageLabel="Out" element="tns:response" />
    </operation>

    <operation name="increment"
             pattern="http://www.w3.org/ns/wsdl/in-out"
             style="http://www.w3.org/ns/wsdl/style/iri"
             wsdlx:safe = "true">
        <output    messageLabel="Out" element="tns:response" />
    </operation>

</interface>

<binding name="RESTfulInterfaceHttpBinding" interface="RESTfulInterfac
             type="http://www.w3.org/ns/wsdl/http">
```

```
    <operation ref="get" whttp:method="GET"/>
    <operation ref="increment" whttp:method="GET"/>
  </binding>

  <service
    name ="Service"
    interface="Interface">

  <endpoint name ="Endpoint"
          binding ="RESTfulInterfaceHttpBinding"
          address ="http://localhost:8080/api"/>

</service>‘
```

Prima di iniziare la creazione della Thing Description andiamo ad ottenere la WSDL attraverso la decodifica del campo "WSDL" del Service in cui era salvato l'hash del WSDL del service. Una volta ottenuto il WSDL attraverso la libreria "fast-xml-parser" trasformo l'oggetto WSDL in un oggetto Javascript che ci permette di lavorarci più facilmente e permette di accedere velocemente ai campi che ci interessano del WSDL. Adesso possiamo iniziare a costruire la Thing Description. Come prima cosa andiamo ad ottenere il title della Thing e possiamo andare ad utilizzare il nome definito all'interno del WSDL all'interno del campo <service>

```
<service
    name ="Service"
    interface="Interface">
</service>‘
```

Quindi in questo caso il titolo della Thing sara "Service"

Una volta ottenuto il title andiamo a definire le Properties della Thing, che sono i valori che il Service può ricevere in input e restituire in output che sono definiti nel campo <types>

```
  <types>
```

```
        <xs:schema xmlns="http://www.example.com/wsdl20sample"
        xmlns:xs="http://www.w3.org/2001/XMLSchema
        "targetNamespace="http://www.example.com/wsdl20sample"/>

        <xs:element name="response" type="xs:int"/>
    </types>
```

Quindi possiamo andare a prendere tutti i campi element ottenerne il nome e il type e definirli come Property all'interno della Thing Description tutte in modalita Read-Only. In questo capo avremo un'unica Property di nome "response" che sarà di tipo "integer".

Una volta definite le Property possiamo andare a definire le Actions della Thing, per farlo abbiamo bisogno di controllare i campi <operation>,<binding>e <endpoint>. Per esempio nel WSDL scelto nel nostro caso:

```
    <interface  name = "Interface" >

    <operation name="get"
            pattern="http://www.w3.org/ns/wsdl/in-out"
            style="http://www.w3.org/ns/wsdl/style/iri"
            wsdlx:safe = "true">
        <output    messageLabel="Out" element="tns:response" />
    </operation>

    <operation name="increment"
            pattern="http://www.w3.org/ns/wsdl/in-out"
            style="http://www.w3.org/ns/wsdl/style/iri"
            wsdlx:safe = "true">
        <output    messageLabel="Out" element="tns:response" />
    </operation>

</interface>
```

```
<binding name="RESTfulInterfaceHttpBinding" interface="RESTfulInterface"
            type="http://www.w3.org/ns/wsdl/http">
      <operation ref="get" whttp:method="GET"/>
      <operation ref="increment" whttp:method="GET"/>
   </binding>

<endpoint name ="Endpoint"
    binding ="RESTfulInterfaceHttpBinding"
    address ="http://localhost:8080/api"/>
```

abbiamo due operazioni get e increment che non richiedono input ma solo un output che è "response" e vediamo attraverso il campo <binding>che entrambe utilizzano protocollo HTTP con metodo GET, quindi andiamo a definire le Actions della Thing andando a salvare nella description della Actions una stringa formata da il metodo HTTP da utilizzare seguito da ":" e l'indirizzo dove eseguire la chiamata HTTP per esempio nel caso della Actions "get" avremo che le description sarà formata dalla stringa "GET:http://localhost:8080/api/get", in più se la Actions dovesse richiedere input aggiungiamo campi UriVariables in cui andiamo a salvare il nome del dato richiesto in input.
Dopo aver definito anche le Actions la Thing Descripton è completa e possiamo andare a creare ed esporre la Thing. Come sempre per poter esporre la Thing istanziamo un servient e procediamo poi all'esposizione della Thing. Prima di esporre però la Thing dobbiamo definire i comportamenti della Thing attraverso le Actions. Come prima cosa dobbiamo definire un array contenente tante posizioni quante le Properties della Thing e utilizziamo come indice i nomi della Properties in modo da renderle facilmente recuperabili. Dopo aver definito le Properties bisogna definire il comportamento della Thing al momento della chiamata di una Actions, in questo caso il Service che ho utilizzato per testare il tutto utilizza il protocollo HTTP quindi possiamo ottenere l'URL e il metodo della chiamata HTTP da utilizzare dalla description della Actions dove abbiamo salvato la stringa con i dati che ci interessano. Una volta ottenuto questi dati possiamo fare che la Thing al momento della chia-

mata di quella Actions eseguirà una chiamata HTTP utilizzando il metodo richiesto e l'URL della Actions che verrà modificato in base se l'Actions richiede dati in input. Una volta che viene eseguita la chiamata se l'Actions ha anche un output avremo che i dati ritornati dalla chiamata andranno ad aggiornare la Property associata a quella Actions. In questo modo avremo che la Thing ogni volta che eseguirà una Action aggiornerà i valori delle Property in base ad i valori del Service originale che stiamo convertendo in modo corretto.
Una volta che abbiamo definito il comportamento della Thing questa è pronta per essere esposta e pronta per essere aggiunta su Modron e una volta caricata ed aggiornato l'array che utilizzavamo come meccanismo di caching abbiamo completato l'operazione di conversione da service a Thing.

4.5 Riepilogo e considerazioni

Nei paragrafi precedenti sono stati spiegati nel dettaglio il funzionamento dei componenti che utilizzati in contemporanea permettono quindi l'integrazione del mondo WoT all'interno del mondo Arrowhead e viceversa andando quindi a creare un tool bidirezionale che permetta la conversione da Thing in Service e da Service in Thing. In questo modo possiamo far vedere che quando abbiamo una Thing registrata su Modron questa verrà registrata anche sul Service Registry Arrowhead e potrà essere utilizzata attraverso il Configurator direttamente dal cloud locale Arrowhead, mentre quando abbiamo un Service registrato sul Service Registry con il requisito di avere un documento WSDL che ne definisce il comportamento questo può essere trasformato in una Thing proxy che viene salvata su Modron ed utilizzabile come una normale Thing ma in più questa verrà nuovamente registrata sul Service Registry ma come una Thing e quindi possiamo interagirci anche con il Configurator andando quindi a definire il nostro tool che permette una comunicazione bidirezionale tra i due mondi. Questa proposta quindi rende possibile l'integrazione del mondo WoT all'interno di Arrowhead e viceversa ma c'è un bisogno di utilizzare uno standard condiviso in primis per la descrizione dei Service e anche per l'utilizzo dei metadata in modo corretto al momento della registrazione di un service all'interno del Service

Registry del cloud locale Arrowhead. Forse però il problema principale per rendere questa proposta realmente utilizzabile è il bisogno di dover ampliare i casi possibili di WSDL tenendo conto per esempio anche di altri possibili protocolli di comunicazioni diversi da HTTP l'unico protocollo di cui si è tenuto conto in questa proposta.

Chapter 5

Risultati

Dopo aver implementato il sistema e testato il suo funzionamento per validare la mia tesi ho eseguito dei test riguardo il tempo impiegato dal sistema nel riuscire a convertire un certo numero di Thing in Service e un certo numero di Service in Thing. Nello specifico attraverso i test sono stati misurati i tempi di conversione riguardo prima 10 poi 50 poi 100 ed infine 200 Thing e lo stesso vale per i Service, in entrambi i casi sono stati scelti una Thing e una Service di esempio che rispettasse gli standard richiesti da questa implementazione ed in seguito moltiplicata come ne fossero 10,50,100 oppure 200. Tutti i test sono stati eseguiti una quantità pari a 50 volte in modo tale che attraverso la media si ottenesse un risultato più veritiero possibile, questo perché come spiegato nel capitolo precedente entrambe le conversioni richiedono operazioni che possono essere influenzate da situazioni esterne che possono condizionare il risultato e quindi ottenere un valore in tempo sempre diverso. Per esempio abbiamo operazioni eseguite attraverso Internet per richiedere ed ottenere la lista delle Thing dal server Modron oppure la lista dei Service dal Service Registry, queste operazioni possono essere influenzate dallo stato della rete e anche dallo stato di carico del server che potrebbe impiegare qualche momento in più per elaborare la richiesta, un altro possibile influenza sul risultato può essere dato dalla macchina su cui sono stati eseguiti i test, in questo caso sono stati eseguiti dal mio laptop personale quasi tutti nello stesso istante quindi questo non avrebbe dovuto influenzare molto il risultato. In questi tipi di test in cui l'ordine dei risultati è di sec-

ondi o millisecondi una piccolo rallentamento per colpa della rete, del server o della macchina può influenzare il risultato ma andando ad eseguire il test più e più volte e ottenendo la media dei risultati vediamo che il risultato finale è simile a quello atteso.

Qui di seguito abbiamo i due grafici che evidenziano i risultati dei test e sono indicati in grigio anche gli intervalli di confidenza per ogni test ottenuti andando ad utilizzare un livello di confidenza del 95% .

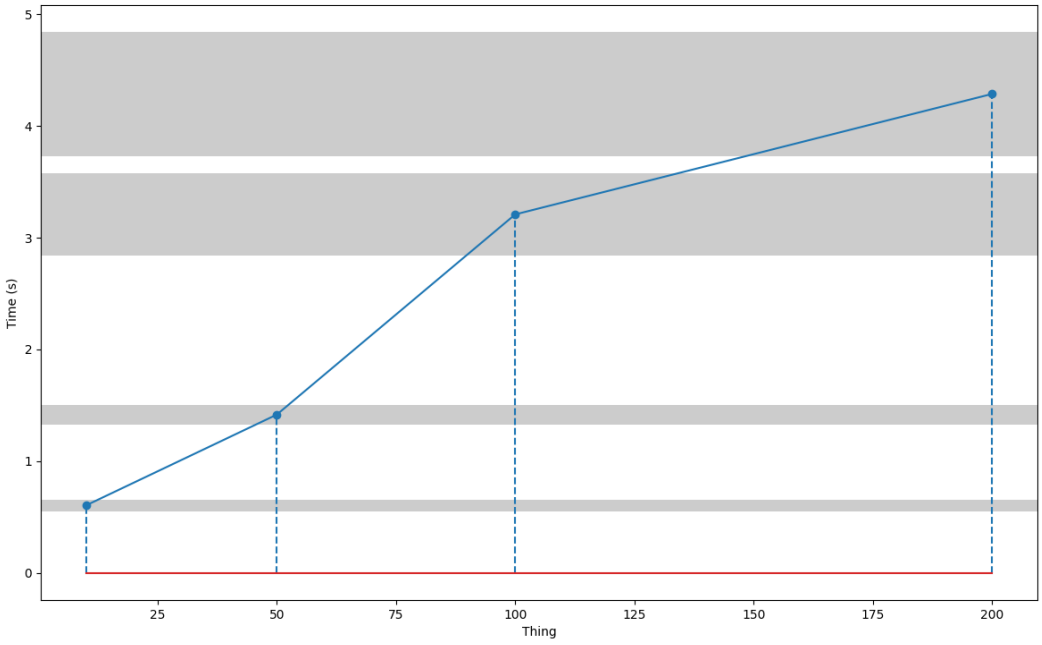

Figure 5.1: Conversione da Thing a Service

Nella figura 5.1 possiamo vedere i risultati ottenuti nella misurazione del tempo impiegato nella conversione di una Thing in una Service che non è altro che la registrazione del Service con i metadata corretti all'interno del Service Registry di Arrowhead, infatti possiamo vedere dal grafico che 10 Thing sono state convertite in meno di 1 secondo, 50 Thing poco più di 1 secondo, 100 Thing in praticamente 3

secondi e infine abbiamo che 200 Thing sono state convertite in poco più di 4 secondi.

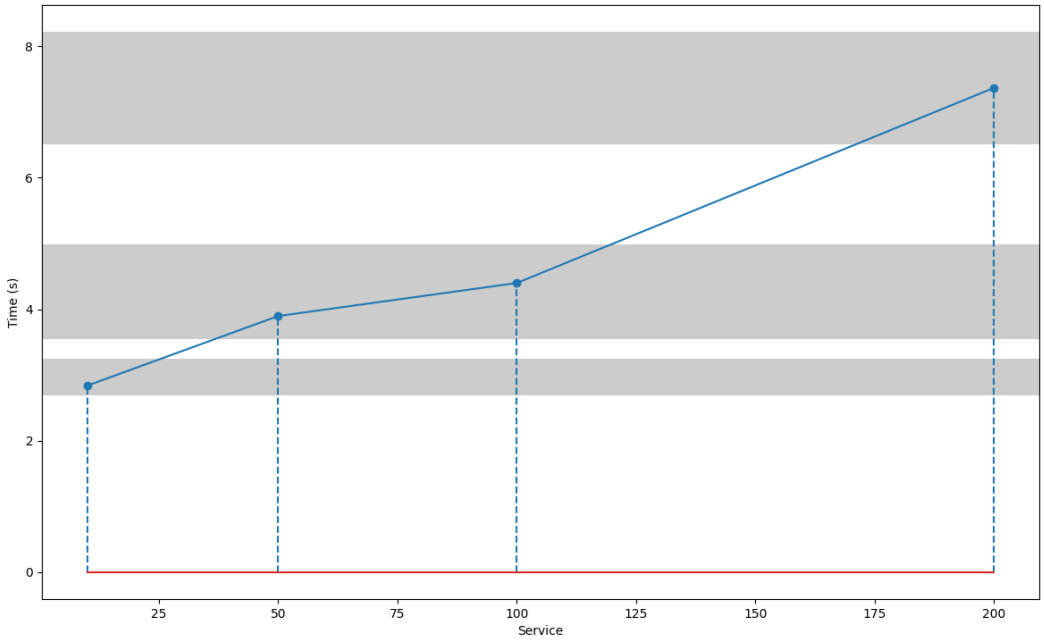

Figure 5.2: Conversione da Service a Thing

Nella figura 5.2 invece vediamo il tempo impiegato per la conversione dei Service in Thing, questo ha richiesto più tempo rispetto all'altro test, ma questo era pronosticabile in quanto c'è bisogno una volta ottenuti i Service della conversione del WSDL in una Thing Description e infine dell'esposizione della Thing che richiede più tempo. In questo caso infatti abbiamo che 10 Service sono convertiti in quasi 3 secondi ne abbiamo 50 convertiti in quasi 4 secondi, 100 Service in poco più di 4 secondi e 200 Service in praticamente 7 secondi.

In conclusione però possiamo ritenerci soddisfatti di entrambi i risultati ottenuti, nonostante qualche caso in cui il valore ottenuto è stato nettamente superiore rispetto

al valore atteso ottenuto nelle altre misurazioni probabilmente dovuto a fattori esterni spiegati in precedenza quali qualità della rete o carico dei server abbiamo ottenuto valori che ci permettono di dimostrare che il sistema è in grado di convertire grandi numeri di Service o Thing in pochi secondi e soprattutto possiamo notare che all'aumentare del numero di elementi da convertire il tempo aumenta in modo quasi lineare che ci permette quindi di validare la nostra tesi in quanto riesce a lavorare con tanti elementi senza avere problemi e quindi avere un possibile aumento esponenziale del tempo richiesto che non sarebbe che non sarebbe accettabile dovendo il nostro sistema lavorare ipoteticamente con possibili grandi quantità di elementi.

Chapter 6

Conclusioni e sviluppi futuri

In questa tesi ho voluto presentare un possibile modo per implementare in modo bidirezionale il mondo WoT con il framework Arrowhead uno dei tanti standard IoT creati in seguito alla diffusione dell'Internet of Things nella vita di tutti di i giorni ma soprattutto nel mondo lavorativo. Riuscire ad integrare piattaforme IoT che spesso sono definiti con standard proprietari con il mondo WoT è uno degli obiettivi del W3C per riuscire ad unificare il mondo Internet of Things all'interno del mondo Web of Things. Attraverso questa proposta riusciamo ad integrare il mondo WoT con il mondo Arrowhead attraverso una integrazione bilaterale permettendo la conversione di una Thing in un Service Arrowhead ma anche la possibilità di convertire un Service Arrowhead in una Thing che rispetta i requisiti del mondo WoT e quindi funzionante con cui possiamo interagire in modo semplice attraverso i framework WoT. I risultati ottenuti validano la nostra tesi che conferma la possibilità di unire i due mondi ma il lavoro non è completo, la mia come già detto è una proposta che è stata testata attraverso l'utilizzo di Thing e Service create ad hoc per testare il sistema quindi sicuramente ci sono possibili lavori futuri con cui implementare e migliorare il sistema proviamo a descriverne alcuni nel prossimo paragrafo.

6.1 Lavori futuri

Il sistema sviluppato è testato e funzionante però ha sicuramente alcuni limiti dovuti sia per quanto riguarda il mondo WoT sia per il mondo Arrowhead, alcune problematiche sono:

1. All'interno di questa proposta non è stato trattato l'analogia di elementi di una Thing con elementi di un Service per esempio non abbiamo parlato dell'analogia di un Events di una Thing con un possibile elemento descritto all'interno del WSDL di un web service

2. In questo caso sono stati utilizzati Thing e Service con annesso WSDL creati ad hoc non andando quindi a considerare alcuni casi possibili all'interno di un WSDL o di una Thing Description

3. In questa proposta si è tenuto conto di un solo protocollo di comunicazione quale HTTP

4. Alcuni operazioni di un web service REST non sono implementabili attraverso le operazioni fornite da un Thing

Questi sono alcuni dei limiti di questa proposta che quindi potrebbe essere migliorata attraverso alcuni possibili lavori futuri:

- Implementare i casi di cui non si è tenuto conto come la possibile presenza di alcuni tipo di dato particolare e quindi rendere il sistema utilizzabile con ancora più dispositivi possibili che porterebbe alla risoluzione dei problemi 1) e 2)

- Implementare convertitori in grado di gestire altri possibili protocolli di comunicazione dovendo probabilmente definire una nuova idea alla base della conversione essendo che gli altri metodi diversi da HTTP potrebbero richiedere dati o header o altro definiti in modo diverso da HTTP. Questo porterebbe alla risoluzione del limite 3)

- Per quanto riguarda invece il problema 4) credo sia quello più difficile da risolvere probabilmente essendoci alcune incongruenze alla base della definizione dei due standard che potrebbero portare quindi alla non esistenza di una possibile soluzione o se esistente questa è ottenibile solo attraverso un grosso lavoro di implementazione e computazionale.

Bibliography

[1] Express. https://www.npmjs.com/package/express.

[2] Fast-xml-parser. https://www.npmjs.com/package/fast-xml-parser.

[3] Graphql, a query language for your api. https://graphql.org/.

[4] Isomorphic-fetch. https://www.npmjs.com/package/isomorphic-fetch.

[5] Js-base64. https://www.npmjs.com/package/js-base64.

[6] Npm. https://www.npmjs.com/.

[7] Object-hash. https://www.npmjs.com/package/object-hash.

[8] Prompt-sync. https://www.npmjs.com/package/prompt-sync.

[9] Thingweb.node-wot. https://github.com/eclipse/thingweb.node-wot.

[10] Cristiano Aguzzi, Lorenzo Gigli, Luca Sciullo, Angelo Trotta, Federica Zonzini, Luca De Marchi, Marco Di Felice, Alessandro Marzani, and Tullio Salmon Cinotti. Modron: A scalable and interoperable web of things platform for structural health monitoring. In *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–7. IEEE, 2021.

[11] Arrowhead. Arrowhead framwork wiki. https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page.

[12] Arrowhead. Arrowhead project. http://www.arrowheadproject.eu/.

[13] Christine Bannan. The iot threat to privacy. https://techcrunch.com/2016/08/14/the-iot-threat-to-privacy/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAF4lDfsUt0V6EjjAfk6SCl3q4YGV_DUNk0XDASiiHcVjkBhQ7ZRGrEVJZJQ0q0NiboQnfzB3ENDBQai_aQYidegVcx-l3PBEvYVmgLSeA1-5znXNKBq2_yx736bi7Ozhwl8AUt3_DfHUU0EzRHct8F31_ExFCDJWZBqktYO4PvoF.

[14] Reshetova Elena and McCool Michael. Web of things (wot) security and privacy guidelines. https://www.w3.org/TR/2019/NOTE-wot-security-20191106/.

[15] Sjoerd Langkemper. The most important security problems with iot devices. shorturl.at/eirAQ.

[16] Koster Michael and Korkan Ege. Web of things (wot) binding templates. https://www.w3.org/TR/2020/NOTE-wot-binding-templates-20200130/.

[17] Luca Sciullo, Federico Montori, Angelo Trotta, Marco Di Felice, and Tullio Salmon Cinotti. Discovering web things as services within the arrowhead framework. In *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, volume 1, pages 571–576. IEEE, 2020.

[18] Käbisch Sebastian, Kamiya Takuki, McCool Michael, Charpenay Victor, and Kovatsch Matthias. Web of things (wot) thing description. https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/.

[19] Statista. Internet of things (iot) and non-iot active device connections worldwide from 2010 to 2025. https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/.

[20] W3C. Thing description directory. https://www.w3.org/TR/wot-discovery.

[21] W3C. Web of things (wot). https://www.w3.org/WoT/.

[22] W3C. Web of things (wot) architecture. https://www.w3.org/TR/wot-architecture/.

[23] W3C. World wide web consortium (w3c). https://www.w3.org/.

[24] Wikipedia. Kevin ashton. https://it.wikipedia.org/wiki/Kevin_Ashton.

[25] Wikipedia. Web services description language. https://it.wikipedia.org/wiki/Web_Services_Description_Language.

[26] Wikipedia. Xml. https://it.wikipedia.org/wiki/XML.

[27] Kis Zoltan, Peintner Daniel, Aguzzi Cristiano, Hund Johannes, and Nimura Kazuaki. Web of things (wot) scripting api. https://www.w3.org/TR/wot-scripting-api/.

[28] Ivan Zyrianoff, Lorenzo Gigli, Federico Montori, Carlos Kamienski, and Marco Di Felice. Two-way integration of service-oriented systems-of-systems with the web of things. In *IECON 2021–47th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–6. IEEE, 2021.