Scuola di Scienze Matematiche, Fisiche e Naturali

Relazione Digital Twin FIWARE

Software Architectures and Methodologies

PIETRO BERNABEI

MATRICOLA 7064862

Academic Year 2022-2023

Sommario

Introduzione

Con la maturazione delle tecnologie dell IoT e delle IA, nasce una delle più promettenti tecnologie, i Digital Twin. Questa nuova frontiera è il frutto dell'unione delle misurazioni dell'ambiente reale, con i modelli di intelligenza artificiale per creare repliche virtuali di oggetti o sistemi fisici.

Nella prima parte della relazione sarà presentata una definizione di Digital Twin e le principali componenti a livello teorico che li compongono. Successivamente sarà presentata una panoramica sui concetti chiave del framework FIWARE e tutte le scelte di design che gli hanno permesso di raggiungere le proprietà che oggi può offrire nel campo della riusabilità e dell'interoperabilità, nello specifico i tre pilastri della strategia: Smart Data Models, architettura e NGSI-LD. Dopo questa panoramica sui concetti chiave del framework, sarà presentata un'analisi di una soluzione matura di un context broker, ovvero ScorpioBroker. Come conclusione viene proposto lo sviluppo di un prototipo di context broker progettato per un ambiente industriale con lo scopo di mantenere la flessibilità di impiego fornito dal framework.

Definizione tratta da AWS

Un gemello digitale è un modello virtuale di un oggetto fisico. Esegue il ciclo di vita dell'oggetto e utilizza i dati in tempo reale inviati dai sensori sull'oggetto per simulare il comportamento e monitorare le operazioni. I gemelli digitali possono replicare molti elementi del mondo reale, da singole apparecchiature in una fabbrica a installazioni complete, come turbine eoliche e persino intere città. La tecnologia dei gemelli digitali consente di supervisionare le prestazioni di una risorsa, identificare potenziali guasti e prendere decisioni più informate sulla manutenzione e sul ciclo di vita.

Da queste definizioni è chiaro che ci sono tre componenti importanti nel DigitalTwins di un oggetto:

- un modello dell'oggetto,
- un insieme in evoluzione di dati relativi all'oggetto, e
- un mezzo per aggiornare o adattare dinamicamente il modello in base ai dati.

Nello specifico il Digital Twin viene modellato da parte di IA e di specialisti, mentre la simulazione del comportamento di un modello è a cura di IA che elaborano i dati provenienti dai sensori IoT presenti sull'oggetto da digitalizzare e non.

Nei prossimi capitoli sarà presentato un framework per lo sviluppo di un sistema che permetta l'interazione tra sorgente dei dati e modello, e successivamente una sua implementazione.

Framework FIWARE

Standard

FIWARE definisce un framework per la progettazione di piattaforme e componenti relative ai DT, con l'obiettivo di semplificare il loro sviluppo e la loro interoperabilità. Il suo obiettivo principale è lo sviluppo di un ambiente uniformato nella strutturazione e nello scambio delle informazioni (Web dei DT). Per realizzare questo progetto, FIWARE ha impostato la sua strategia su tre macro-temi:

- Architettura,
- NGSI-LD e,
- Smart Data Models.

Per quanto riguarda l'**architettura**, che sarà approfondita successivamente, FIWARE definisce come struttura di riferimento per la gestione delle Context Information, CoBrA (Context Broker Architecture). L'organizzazione però non chiude ad altre realtà lo sviluppo di soluzioni che compiono scelte architetturali diverse, fornendo uno standard ETSI sull'argomento: NGSI-LD ISG CIM - Context Information Management (NGSI-LD più brevemente).

Lo standard **NGSI-LD** rappresenta l'evoluzione standardizzata del NGSIv2, uno standard interno precedentemente in uso in FIWARE. NGSI-LD regola diversi aspetti come: il modello di informazione, le API per le interazioni, e un'architettura per diverse situazioni: locali, distribuite e federate. Con la strutturazione dei modelli di informazione e le rispettive API qualsiasi componente, legacy o di terze parti, può comunicare e partecipare alla rete FIWARE, implementando dei connettori conformi allo standard.

L'ultimo punto della strategia sono gli **Smart Data Models**, ovvero modelli di informazione che descrivono i Digital Twin, definendo strutture standard condivisibili. Questo permette la facile condivisione dei modelli tra organizzazioni differenti.

Smart Data Models

Come già anticipato lo scopo degli Smart Data models è di definire dei data model comuni e compatibili, fornendo una rappresentazione armonizzata dei formati e della semantica per la pubblicazione e il consumo dei dati. La standardizzazione dei formati permette la creazione di un ambiente di replicazione e interoperabilità di soluzioni in settore diversi.

Nello specifico i Data Models sono raggruppati per tipi, dove ciascun tipo può essere afferente ad almeno un dominio o a un settore industriali. Al momento della scrittura di questa relazione i domini ufficialmente riconosciuti sono:

- SMART CITIES
- SMART ENVIRONMENT
- CROSS SECTOR
- SMART AGRIFOOD
- SMART SENSORING
- SMART ROBOTICS
- SMART WATER
- SMART AERONAUTICS
- SMART HEALTH
- SMART ENERGY
- SMART DESTINATION
- SMART MANUFACTURING

Gli Smart Data Models però non si limitano a definire delle semplici strutture statiche e immutabili, ma anzi spinge l'uso dei modelli adattandoli ai singoli casi d'uso, permettendo o un uso più contenuto dell'espressività del singolo modello o permettendo l'arricchimento dello stesso con informazioni specifiche alle singole applicazioni.

I data models sono definiti nel formato JSON/JSON-LD secondo lo standard NGSI v2/-LD rispettivamente e sono pubblicati su delle repository git suddivisi secondo i domini precedentemente elencati.

Struttura di un Data Model

I data model sono strutturati in 6 documenti:

- schema.json,
- ADOPTERS.yaml,
- CONTRIBUTORS.yaml,
- examples*,

- notes.yaml,
- notes_context.jsonld.

Di questi documenti, l'unico non obbligatorio è: notes_context.jsonld.

Nello specifico **schema.json** definisce gli attributi, il tipo degli attributi, una loro descrizione e altre strutture. Esso rappresenta lo scheletro principale di un data model, permettendo di validare un qualsiasi payload rispetto a sé, mentre il payload a sua volta può generare lo schema stesso.

ADOPTERS.yaml e **CONTRIBUTORS.yaml** sono due documenti che possono essere volontariamente compilati da coloro che: utilizzano il modello nel proprio progetto o che hanno contribuito a definire il modello. In entrambi i casi, potranno essere fornite le informazioni degli autori, dei loro progetti e della loro organizzazione se presente.

Per quanto riguarda i file **example***, esprimono un esempio dei payload validi rispetto allo schema. Questi possono essere di 4 tipologie, in combinazione: normalizzato e key-value, LD e v2. Di tutti questi è necessario fornire un esempio sia in formato JSON e JSON-LD.

notes.yaml è un documento per la personalizzazione della specifica e del README del data model.

Per maggiori informazioni:
https://docs.google.com/presentation/d/e/2PACX-1vTs-Ng5dIAwkg91oTTUdt8ua7woBXhPnwavZ0FxgR8BsAI_Ek3C5q97Nd94HS8KhP-r_quD4H0fgyt3/pub?start=false&loop=false&delayms=3000&slide=id.p1

Struttura generale di un digital twin

Come sarà esposto successivamente, la rappresentazione dei data model e di tutte le definizioni NGSI-LD si basa su tre meta-model: Entità, Relazioni e Proprietà. Questi rappresentano i blocchi alla base della strutturazione di qualsiasi informazioni.

Ogni Data Model si compone di un Id e di un tipo (campi obbligatori) e di una collezione di proprietà e relazioni. Questa generalizzazione nella sintassi e nella forma dei dati fornisce ai modellisti il compito di raffinare la struttura per rappresentare il modello, scegliendo gli attributi e le relazioni necessarie. D'altra parte, permette a tutte le componenti di poter definire logiche di manipolazione indipendenti dal singolo data model, data la comune forma di rappresentazione dei context information.

Esempio di una instanza Machine Manifacturing

Di seguito viene fornito un esempio di una Machine, ovvero un'istanza di uno schema di uno Smart Data Model di tipo Machine. L'esempio scelto è stato generato automaticamente da https://smart-data-models.github.io/dataModel.ManufacturingMachine/ManufacturingMachine/examples/example-normalized.jsonld e la rappresentazione scelta è quella del formato Normalized. Sono previste due diverse modalità di rappresentazione Normalized e Key-Value. La Key-Value fornisce una visione più concisa dei dati, siccome cancella tutte le diciture ripetute nella rappresentazione Normalized come "type","value", "Property", "Relationship", ecc.

```
{
    "id": "urn:ngsi-ld:Machine:9166c528-9c98-4579-a5d3-8068aea5d6c0",1
   "type": "ManufacturingMachine",2
   "source": { 3
      "type": "Property",
      "value": "https://source.example.com"
```

```
},
"machineModel": {4
    "type": "Relationship",
    "object": "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd69c3"
},
"installedAt": {5
    "type": "Property",
    "value": {
        "@type": "DateTime",
        "@value": "2017-05-04T10:18:16Z"
    }
},
"countryOfManufacture": {6
    "type": "Property",
    "value": "UK"
},
"location": {
    "type": "GeoProperty",7
    "value": {
        "type": "Point",
        "coordinates": [
            -104.99404,
            39.75621
        ]
    }
},
"description": {
    "type": "Property",
    "value": "Industrial machine to create plastic bottles"
},
"osVersion": {
    "type": "Property",
    "value": "10A"
},
"supportedProtocol": {8
    "type": "Property",
    "value": [
        "HTTP",
        "HTTPS",
        "FTP"
    ],
"building": {
    "type": "Relationship",
    "object": "urn:ngsi-ld:Building:8683b757-649c-49e0-ac89-ad392c9a0d0c"
},
"voltage": {9
    "type": "Property",
    "value": 220,
    "unitCode": "VLT",
    "observedAt": "2016-08-08T10:18:16Z"
},
"rotationalSpeed": {
    "type": "Property",
    "value": 10,
```

```
      "unitCode": "RPM",
      "observedAt": "2016-08-08T10:18:16Z"
   }
}
```

Figura 1: Esempio di una istanza del modello Machine Manufacturing

Come si può notare nell'esempio appena fornito, l'entità Manufacturing Machine dispone di un identificativo 1 e di un tipo 2. Successivamente, l'entità è popolata con le proprietà (e.g. 3) e le relazioni (e.g. 4) che ne arricchiscono la rappresentazione. Come si vedrà poi successivamente nel capitolo NGSI-LD, le proprietà potranno assumere diversi valori da temporali 5 a testuali 6, o essere di diverse tipologie, come per esempio Geo Property 7, ma anche composto da valori di diverse tipologie come array di elementi 8 o di valori diversi 9.

Esempio di uno schema di uno Smart Data Models

Tra i documenti obbligatori per la definizione di uno Smart Data Models, è presente lo schema.json. Questo documento rappresenta lo scheletro di un data model, siccome fornisce la descrizione precisa di tutti i campi che lo compongono e le regole che ne definiscono la natura.

Di seguito viene fornito un esempio di schema di Manifacturing Machine, riducendolo ai soli attributi presenti nell'esempio precedente Esempio di una instanza Machine Manifacturing.

Nella parte iniziale dello schema 0 sono definite tutti i metadata: dalla versione dello schema, alla sua collocazione tra gli smart data models, al suo titolo ed ecc. Con 1 sono definiti i valori obbligatori dello schema, ovvero gli id e il tipo, che come già anticipato sono comuni a tutti i modelli. Con il campo successivo, 2, vengono elencati tutti gli attributi permessi, tra relazioni e proprietà, che possono essere popolati da una instanza. I primi due elementi elencati rappresentano un collegamento a risorse esterne, nello specifico 3 rappresenta la definizione specifica della proprietà GeoProperty Location. Attraverso l'uso dei linked data, come in questo caso, viene fornita al modellista la flessibilità di poter riusare definizioni precedentemente strutturate all'interno del proprio modello riducendo così errori, tempo speso alla definizione e aumentando l'usabilità del modello stesso.

Anche se il campo tipo è un campo obbligatorio comune a tutti i modelli, lo schema forza il tipo accettato, accettando come unico valore "ManifacturingMachine" 4.

Con il campo MachineModel 5, ci viene fornita una descrizione della attributo Relationship, fornendoci prima la struttura del valore accettato 6, o un URI o una stringa che soddisfi la Regex, e successivamente il significato della relazione, con il campo description 7, fornendo così a coloro che vorranno popolare il significato associato al campo.

Come per l'attributo relazione precedente 5, l'attributo "countryOfManufacture" 8, viene fornita la struttura del campo, definendo il tipo di valore accettato e una definizione testuale del campo, specificando che sarà una proprietà e la sua semantica.

Per una visione completa di questo esempio con tutti i campi presenti nell'istanza è presente nel Annex A - Esempio completo dello schema ManifacturingModel

```
{
   "$schema": "http://json-schema.org/schema",0
   "$schemaVersion": "0.0.1",
   "$id":
"https://smart-data-models.github.io/dataModel.ManufacturingMachine/ManufacturingMachin
e/schema.json",
   "modelTags": "GSMA",
   "title": "Smart Data models - Manufacturing Machine dataModel schema,",
```

```
    "description": "Description of a generic machine",
    "type": "object",
    "required": [ "id", "type" ],1
    "allOf": [2
        {
            "$ref":
"https://smart-data-models.github.io/data-models/common-schema.json#/definitions/GSMA-C
ommons"
        },
        {
            "$ref": 3
"https://smart-data-models.github.io/data-models/common-schema.json#/definitions/Location-
Commons"
        },
        {
            "properties": {
                "type": {
                    "type": "string",
                    "description": "Property. NGSI entity type. It has to be ManufacturingMachine",
                    "enum": [
                        "ManufacturingMachine"4
                    ]
                },
                "machineModel": {5
                    "anyOf": [
                        {6
                            "type": "string",
                            "minLength": 1,
                            "maxLength": 256,
                            "pattern": "^[\\w\\-\\.\\{\\}\\$\\+\\*\\[\\]`|~^@!,:\\\\]+$",
                            "description": "Property. Identifier format of any NGSI entity"
},                        },
                        {
                            "type": "string",
                            "format": "uri",
                            "description": "Property. Identifier format of any NGSI entity"
                        }
                    ],
                    "description": "Relationship. A reference to the associated Machine Model for this
machine"7
                },
                "countryOfManufacture": {8
                    "type": "string",
                    "description": "Property. Model:'https://schema.org/Text'. The country where this
machine was manufactured"
                },
                …
        }
    ]
}
```

Figura 2: Esempio di schema.json del modello Machine Manufacturing

Architetture

Nei successivi capitoli saranno esposte: una breve descrizione di due architetture per la gestione delle context information e la soluzione consigliata dal framework FIWARE.

Panoramica delle architetture per la gestione delle context information

In letteratura sono state definite almeno due architetture per la gestione di context information e di seguito per ciascuna sarà presentata una breve descrizione:

- Context Toolkit Architecture
- Context Broker Architecture.

Context Toolkit Architecture

La **Context-Toolkit Architecture** mira a facilitare lo sviluppo e l'implementazione di applicazioni sensibili al contesto. I componenti di base di questa architettura sono i context widget, interpreters, aggregators e discoverer. I context widget sono componenti software che forniscono l'accesso alle informazioni di contesto. Gli aggregators raccolgono tutte le informazioni di contesto rilevanti per un'applicazione specifica. Gli interpreters forniscono l'astrazione delle informazioni di contesto ragionando su di esse. Infine, un discoverer mira a determinare quale contesto può essere attualmente percepito dall'ambiente. L'applicazione può accedere all'aggregato, all'interprete o a un widget. Inoltre, il Context Toolkit consente il controllo degli accessi di base per la protezione della privacy.

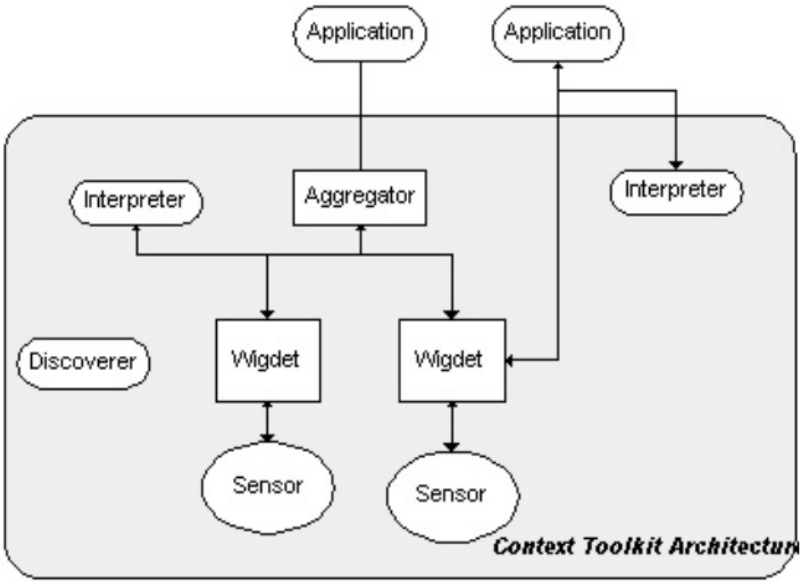

Figura 3: Schema Context Toolkit Architecture

CoBrA - Context Broker Architecture

La **Context Broker Architecture** (CoBrA) propone un'architettura per la discovery, l'acquisizione e il reasoning sulle informazioni di contesto. Include anche meccanismi per la protezione della privacy delle informazioni di contesto. CoBrA presuppone che tutti i fornitori di informazioni di contesto abbiano una conoscenza preliminare della presenza del broker e che comunichino con il context broker tramite un protocollo standardizzato.

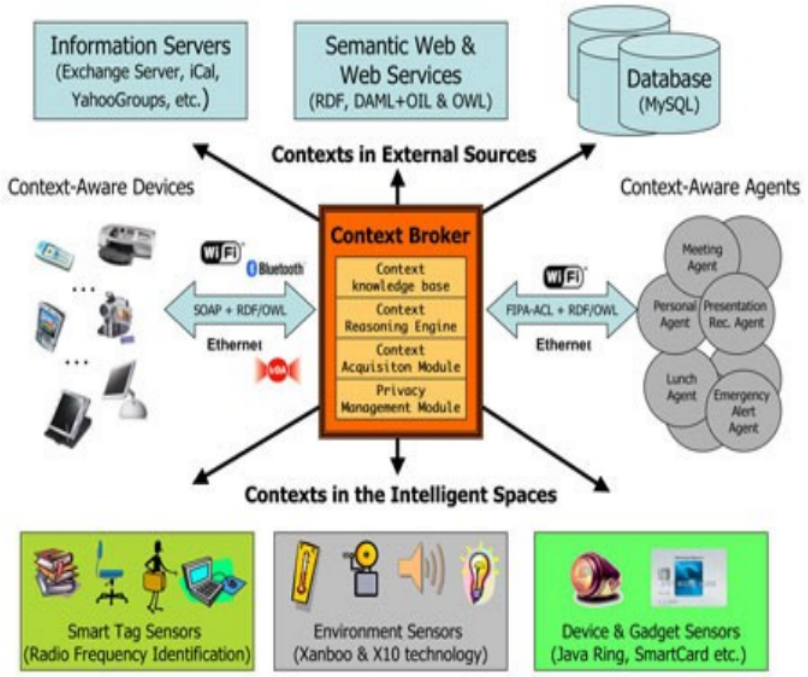

Figura 4: Schema Context Broker Architecture

FIWARE Architecture reference

FIWARE propone nel suo standard per lo sviluppo di applicazioni "powered by FIWARE" l'impiego dell'architettura CoBrA, come anticipato. Nello specifico la componente centrale rappresenta il **Context Broker**, che ha lo scopo di gestire: la ricezione e la somministrazione di informazioni, la registrazione dei producers e dei consumers, e le funzionalità di notifica.

Nel livello inferiore dell'architettura le componenti: NGSI api per Robotics e IoT agents e le interfacce ad altri DataSource System, permettono la connessione delle copie fisiche con gli omologhi digitali. Questi ultimi sono rappresentati nella parte superiore dell'architettura insieme a tutte le altre componenti che hanno lo scopo di elaborare e rappresentare le informazioni collezionate.

Lateralmente, sono identificate tutte quelle soluzioni non strettamente collegate al funzionamento dei digital twin ma che offrono servizi complementari e rilevanti come security, deployment e pubblicazione delle informazioni.

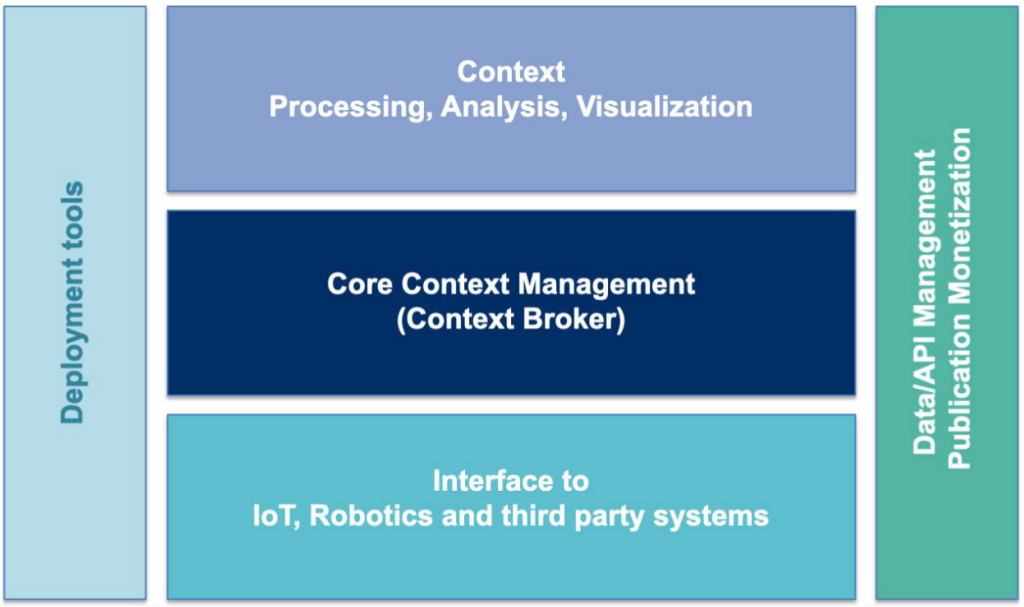

Figura 5: Schema FIWARE Architecture

NGSI-LD

Alla base di tutta la strategia di FIWARE per la realizzazione di un sistema interoperabile, c'è la definizione di un **linguaggio di comunicazione** comune tra tutte le componenti. La sua realizzazione è avvenuta attraverso una specifica di gruppo sviluppata da ETSI ISG CIM, NGSI-LD.

NGSI-LD nasce dalla standardizzazione della precedente NGSIv2, con lo scopo di definire un'API che consenta alle applicazioni di eseguire diverse operazioni sulle context information come:

- interrogazioni e aggiornamenti delle context information;
- subscription/notification;
- registrazione dei context source al sistema per interrogarli per possibili aggiornamenti.

Il modello informativo NGSI-LD definisce anche la struttura delle informazioni di contesto che devono essere supportate da un sistema NGSI-LD e specifica i meccanismi di rappresentazione dei dati che saranno utilizzati dall'API stessa.

Di seguito sarà discusso i concetti alla base della struttura generale dei modelli che saranno scambiati, e la loro implementazione nel linguaggio JSON-LD.

Information Model Structure

Tra gli obiettivi principali di FIWARE, c'è quello di sviluppare un ambiente per lo scambio di Digital Twin nella maniera più aperta possibile e per realizzare ciò, la loro strategia si è fondata in parte sulla definizione di un linguaggio. Più precisamente con l'avvento dello standard NGSI-LD, viene definito un linguaggio comune a tutti coloro che vogliono far parte della rete FIWARE, per la strutturazione e la condivisione di context information.

Questo linguaggio, o meglio questo *information model*, è stato suddiviso in tre sezioni concentriche, con al centro gli elementi alla base di tutto, ovvero il **Meta Model**, e nelle sezione esterne, prima la **cross domain ontology** e poi la **domain specific ontology**. In breve il meta model stabilisce le informazioni più comuni a tutti gli elementi, mentre nel domain specific ontology sono definiti i termini più specifici ai singoli domini. Seguendo questa struttura viene

raggiunta una maggiore profondità di rappresentazione, mantenendo al tempo stesso una base generica al fine di svolgere le attività più comuni, con il risultato di ridurre l'accoppiamento tra i singoli domini, le funzionalità e il linguaggio.

Meta Model

Il **Meta Model** rappresenta il cuore della rappresentazione delle informazioni e si basa su RDF (Resource Description Framework).
Il Meta Model definisce quattro elementi:

- Entity,
- Relationship,
- Property
- e Value.

I **value** possono essere sia letterari sia strutture più complesse.

Per quanto riguarda **property** e **relationship** entrambe possono contenere le due tipologie di elementi. La differenza tra i due, oltre che nella semantica, è che property contiene un valore, mentre relationship può contenere una entity.

La profondità nella rappresentazione di una **entity** è data dalla presenza e dall'accuratezza delle varie property e relationship che elemento contiene.

Di seguito è presente un diagramma che espone la relazione tra tutti i vari elementi.

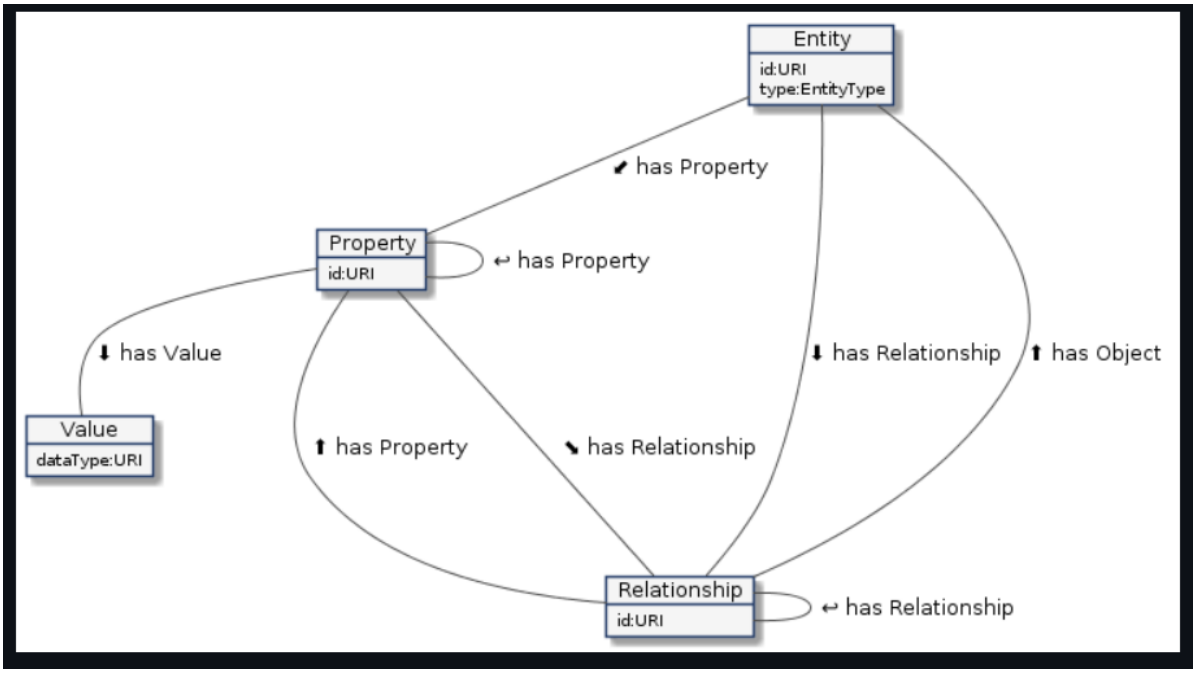

Figura 6: Schema NGSI-LD Meta Model

Cross Domain Ontology

Mentre il meta model definisce i concetti alla base di questa ontologia, il **Cross Domain Ontology** aggiunge le seguenti proprietà:

- **GeoProperties**, proprietà con informazioni geospaziali;

- **Temporal Properties:** proprietà non reificate che trasmettono informazioni temporali per permettere la cattura delle evoluzioni temporali di altre proprietà.
- Più altre che saranno solamente citate come: **Language properties, uniCode Property, Geometry Values,** ecc…

Domain Specific Models

I **Domain Specific Models** di NGSI-LD consentono di definire termini più specifici e affini al singolo dominio. Questo arricchimento permette di rappresentare con maggiore profondità le caratteristiche delle entità fisiche reali.

Per maggiori informazioni: ETSI GS CIM 009 V1.7.1 (2023-06) cap 4.2

Example

Nella Figura 7 viene mostrato un esempio dell'information model, con una specifica per un possibile dominio SmartCities.

Nella parte superiore dell'immagine è presente il livello RDF/RDFS Grounding, sul quale si basa in generale l'Information Model. Infatti le tre componenti principali, Entity, Relationship e Property sono sottoclassi di Resource. Da Property derivano le principali categorie introdotte dal CrossDomainLogic, come GeoProperty e TemporalProperty, nei termini di Location e CreatedAt per esempio (Figura 1 punto 7, è presente un esempio della Location Property). L'ultimo layer rappresenta il Domain Specific Models, il quale introduce specifici termini come le entità: Car, Gate Street, Parking, le due relazioni adjacentTo e hasOpening , e le due proprietà hasState e reliability.

Nell'ambiente delle MachineManufactoring, sono introdotti ulteriori termini come: la relazione building e gli attributi voltage e rotationalSpeed.

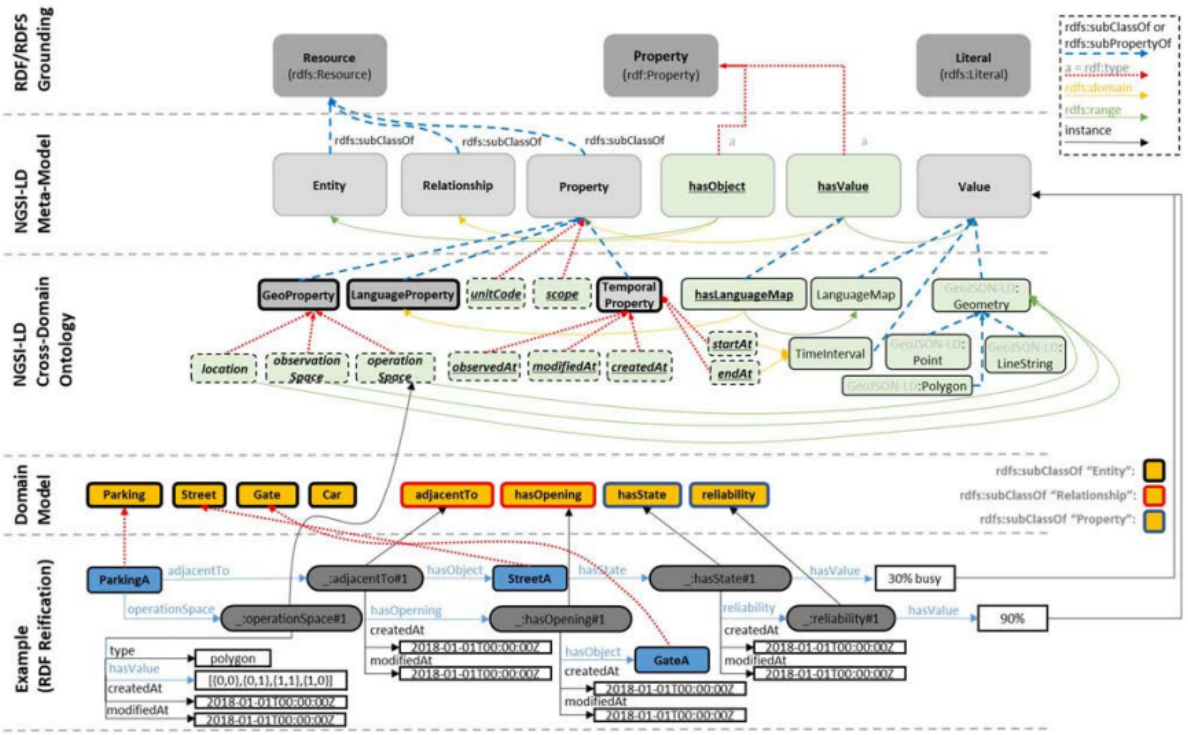

Figura 7: Information Model di Smart Cities

JSON-LD

JSON-LD ha lo scopo di condividere informazioni tra attori diversi distribuiti in realtà diverse, con il fine di garantire una interoperabilità. JSON-LD estende il concetto di JSON con i Linked Data ovvero, introduce all'interno dei JSON link a risorse esterne. Questo al fine di condividere dati referenziati tra loro. Oltretutto dato questo scambio di linked data presenti su domini diversi, JSON-LD introduce il concetto di context all'interno del formato JSON.

Il context permette al producer di associare a tutti i termini presenti nel JSON una definizione certa, al fine di ridurre l'ambiguità nella loro interpretazione. Questa pratica permette così di garantire l'interoperabilità tra sistemi oltre alla riduzione degli errori causati da una errata interpretazione o da un errato popolamento dei dati. Come per un dato normale anche il @context può fare uso di linked data a risorse esterne per una più facilmente condivisibile.

ScorpioBroker

ScorpioBroker è un **context broker** implementato secondo le specifiche dello standard NGSI-LD. Come da documentazione FIWARE, ScorpioBroker è un core component che permette di raccogliere, elaborare, notificare e archiviare i dati provenienti da varie fonti IoT e non, al fine di renderli disponibili ad applicativi di varie entità.

Come sarà esposto in seguito, il core component in questione è stato sviluppato secondo l'**architettura a microservizi**, al fine di garantire una maggiore scalabilità e gestione dei carichi, per andare in contro a sistemi sempre più distribuiti e con carichi più consistenti.

Architettura

ScorpioBroker è un context broker sviluppato con *Spring Boot* e *Quarkus*, che integra la tecnologia di *Apache Kafka* per implementare la comunicazione tra i microservizi.

Nella figura seguente viene mostrata la struttura dei componenti di Scorpio Broker, suddivisa in gruppi con scopi simili. Il Broker implementa l'interfaccia NGSI-LD per comunicare con tutti i produttori e consumatori di dati. Utilizza il Service discovery per individuare i servizi preposti e i canali Kafka per condividere le richieste. Per quanto riguarda la persistenza delle risorse, sono fornite diverse soluzioni, sia locali che distribuite (federate).

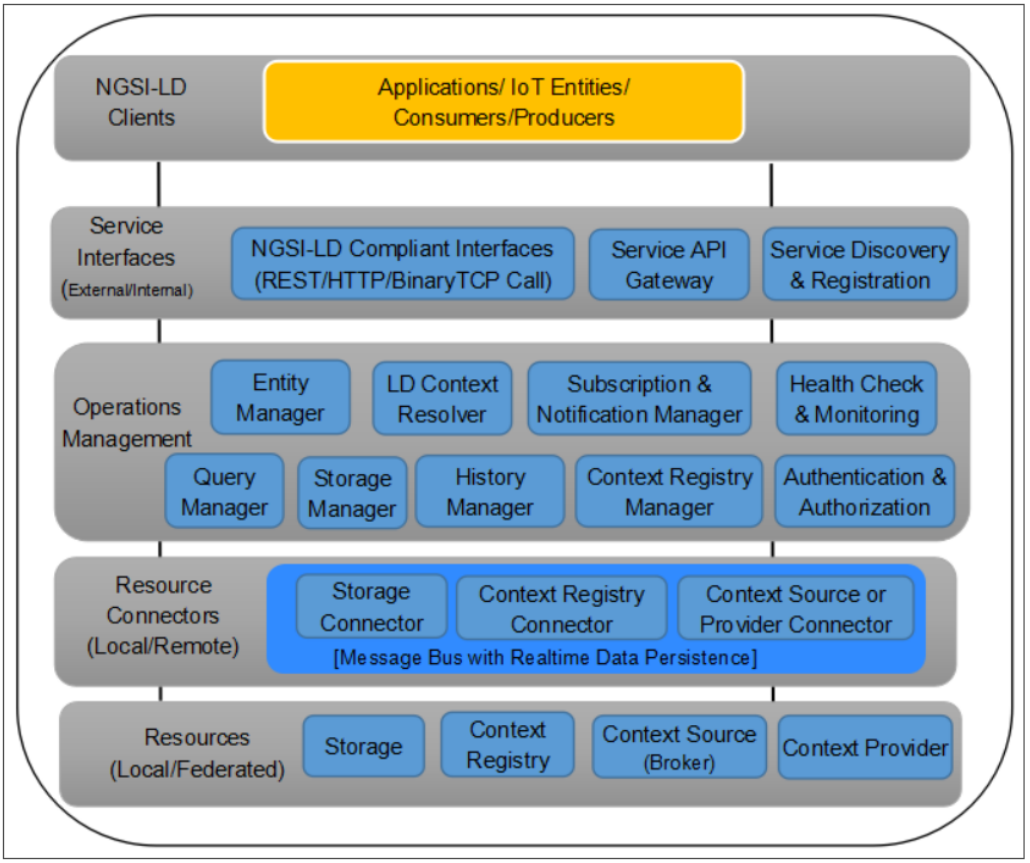

Figura 8: Schema ScorpioBroker Architecture

Di seguito una più specifica descrizione dei principali layer e componenti in campo:

- **NGSI-LD Clients:** Tutte le entità ivi elencate, rappresentano i possibili interlocutori del sistema, e come tali sono fonte e consumatori di context information.
- **Service interface:** Il context broker mette a disposizione degli interlocutori una serie di interfacce, che forniscono dalle più basilari interrogazioni CRUD, alle più complesse richieste temporali e geospaziali. Data la natura minuziosa del broker, al client viene esposta un'**interfaccia REST** attraverso la quale può accedere ai servizi interni. A loro volta i servizi sono registrati nel **Service Discovery & Registration,** e in combinazione al **Service API Gateway,** viene permesso l'instradamento a runtime delle richieste alle istanze dei servizi disponibili**.** Nello specifico ogni micro servizio fornisce una interfaccia REST che il service API Gateway espone a tutti gli attori esterni attraverso un unico punto d'accesso, oscurando la struttura interna del sistema.

- **Operation Management:** i servizi che lo compongono spaziano dalla gestione delle operazioni CRUD, all'espansione delle componenti LD, alla gestione delle query e dei sistemi di storage. Nello specifico:
    - **Entity Manager:** gestisce tutte le operazioni CRUD sulle entità con l'ausilio di altre componenti.
    - **LD Context Resolver:** componente responsabile delle attività di espansione e riduzione dei documenti JSON-LD con i riferimenti @Context presenti nelle richieste.
    - **History Manager:** o HistoryEntityManager che gestisce la rappresentazione temporale dell'entità, delle relazioni e delle proprietà.
    - **Context Registry Manager**: Questa componente è responsabile della gestione delle operazioni di CRUD per quanto concerne le csource. Nota: Le csource o ContextSource sono delle entità esterne al broker che si rendono disponibili a fornire del Context Information attraverso delle interfacce NGSI-LD).
- **Resource Connector e Resources:** ScorpioBroker implementa una serie di connettori logici a risorse locali e remote, e meccanismi di interfacciamento a context registry. Le informazioni o le richieste elaborate dai microservizi precedenti Questi due layer possono essere associati al repository layer

Oltre alla capacità fornite come context broker, ScorpioBroker integra delle componenti per la coordinazione e funzionalità, come il Service Discovery&Registration, l'Health Check & Monitoring, Authentication & Authorization, e i Message Bus with Realtime Data Persistence.

- **Service Discovery & Registration:** Essendo l'applicativo sviluppato secondo l'architettura a microservizi, tutte le componenti che lo costituiscono, possono essere duplicate al fine di scalare. Per questo motivo tutte le istanze dei vari servizi si registrano al Service Discovery & Registration, fornendo così una visione dello stato dell'architettura. Nel momento in cui l'API Gateway riceve una richiesta da parte di un attore esterno, richiede al servizio in questione un riferimento ad un servizio al quale possa essere inoltrata la richiesta.
- **Health Check & Monitoring:** Il servizio ha il compito di verificare e monitorare lo stato attuale delle componenti al fine di identificare possibili malfunzionamenti, e provvedere a mitigarli.
- **Authentication&Authorization:** è la componente dedicata all'integrazione nel context broker delle funzionalità di autenticazione e autorizzazione alle richieste pervenute. Nello specifico se la funzionalità di sicurezza è attiva la componente chiede all'interlocutore di identificarsi attraverso una pagina di login, verificandone l'identità e i privilegi per l'accesso alle risorse.
- **Message Bus with Real Time Data Persistence:** è il sistema di messaggistica dell'infrastruttura che permette alle componenti di poter interagire al fine di assolvere ai propri compiti. Questo è implementato attraverso i canali Apache Kafka.

Per maggiori informazioni: Scorpio Broker

Progetto Sperimentale

Il progetto in questione ha lo scopo di sviluppare un prototipo di un applicativo per la gestione di trivelle composite. Più precisamente è stato preso sotto esame il caso d'uso di una trivella per

lo scavo di tunnel infrastrutturali, come la talpa Iris in uso per la realizzazione della TAV a Firenze.

Prima di proseguire è necessario fornire una breve descrizione sul contesto. La trivella Iris, è una tipologia di trivella particolarmente complessa e completa, composta da diversi componenti che operano all'unisono nella costruzione di tunnel sotterraneo. Semplificando la sua struttura, il macchinario è la combinazione di più macchinari: da una trivella, che erode il terreno, a un sistema di locomozione che permette l'avanzamento del complesso e il sistema di costruzione del tunnel stesso che installa strutture cementizie alle pareti terrose. Per lo scopo di questo progetto, le entità in campo sono state volutamente semplificate, al fine di mettere in risalto le capacità della tecnologia Fiware anche al di fuori del semplice contesto dei digital twin. Infatti sono state estese le capacità del context broker come la gestione di sistemi complessi e diversificati come appunto una trivella multi componente.

Per la realizzazione del prototipo saranno impiegati principalmente i data model del Smart Manifacturing, ovvero i macchinari, le operazioni che questi eseguono e i modelli dei macchinari (al seguente link sono disponibili i sorgenti sopra citati https://github.com/smart-data-models/dataModel.ManufacturingMachine/tree/bacdd3606b132f87db0ef4a78e1ea5f5f0302908).

Analisi dei requisiti

- Funzionali:
  - permettere la creazione di entità semplici e composite (e.g. composizione di macchinari)
  - permettere la definizione di operazioni semplici (per singoli macchinari) e composite (afferenti ad entità composite).
  - fornire le seguenti funzionalità:
    - CRUD su entità, relazione e attributi.
  - Accettare JSON-LD.
- Non funzionali:
  - L'applicazione deve fornire API REST secondo lo standard NGSI.
  - Flessibile:
    - compatibile con tutti gli Smart Data Models,
    - compatibile con Data Models non standardizzati.
  - Multipiattaforma.
  - Reliable
  - Stateless.
- Di dominio:
  - Sviluppato secondo NGSI-LD standard.
- Vincoli:
  - Payload validi rispetto rispetto agli schema pubblici.
  - Decoupled dallo specifico caso, ma affine al meta model NGSI-LD.

Diagramma dei Casi d'Uso progettati

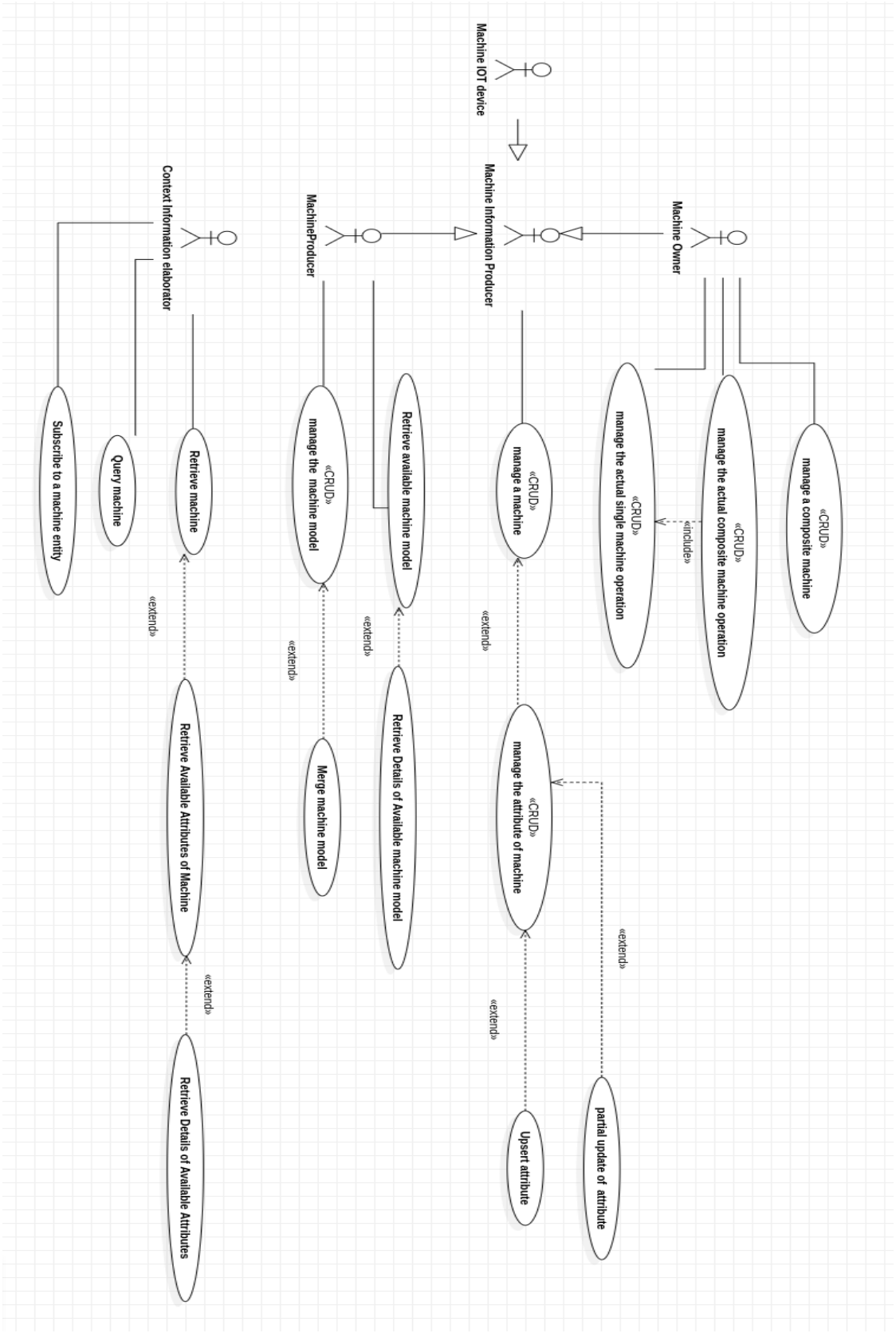

Figura 9: Use Case Diagram

Per una migliore visione si consiglia di visualizzare l’immagine al link associata alla figura.

Modello di dominio

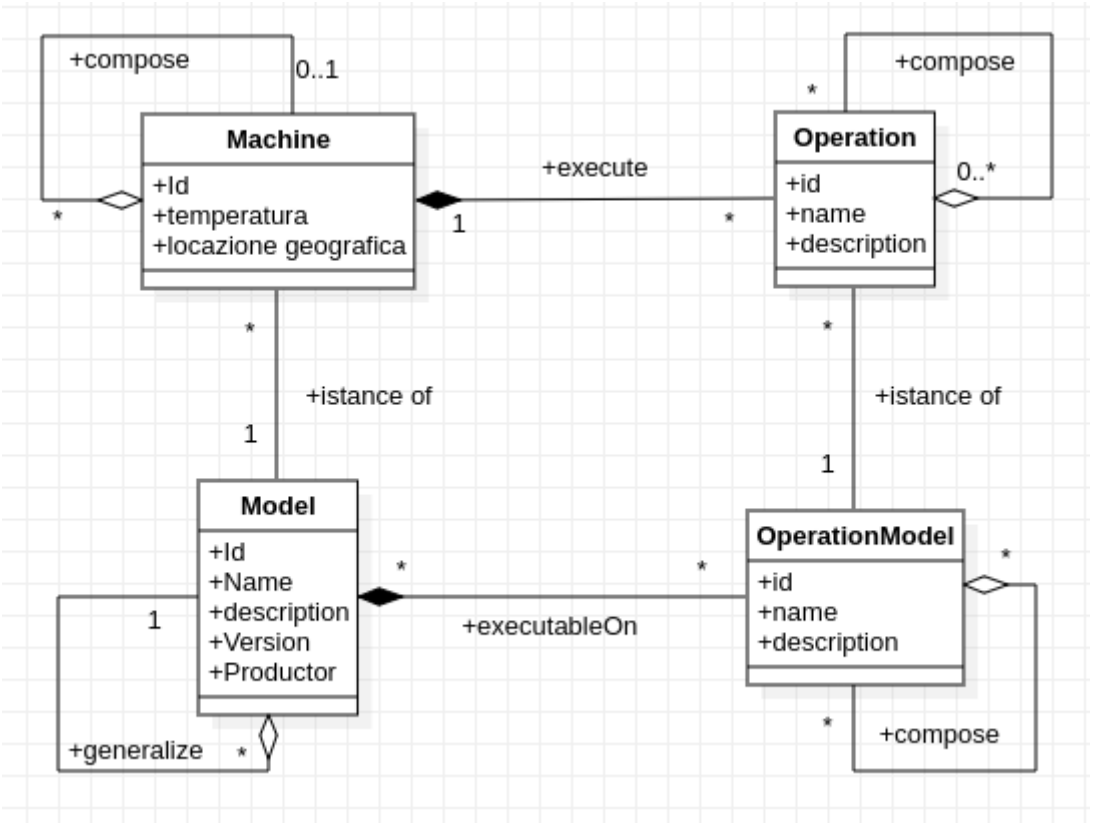

Figura 10: Class Diagram dello scenario della repository MachineManufacturing

Nella Figura 10 è descritto un eventuale scenario di un'industria con diversi modelli di macchine che eseguono diverse operazioni. Più precisamente lo scenario in questione prevede la strutturazione di macchinari composti da altri macchinari indipendenti dal punto di vista funzionale che cooperano insieme. Per realizzare ciò, è stato esteso lo Smart Data Models di riferiemento, MaufactoringMachine, introducendo una entità e delle relazioni aggiuntive.

Ogni macchinario (**Machine**) è descritto da un identificativo univoco e da diverse proprietà fisiche, tra cui la temperatura e la localizzazione nello spazio. In generale, al singolo macchinario saranno associate tutte le proprietà che rappresentano il suo stato e quindi più dinamiche nel tempo. Il singolo macchinario può essere a sua volta parte di un macchinario più complesso.

Per quanto riguarda i possibili attributi più generali alla categoria di macchinari, questi sono descritti dal MachineModel (nel diagramma **Model**).

Il **MachineModel** fornisce un identificativo del modello, il nome e altre informazioni generali, come la versione e il produttore. I modelli a loro volta possono essere delle personalizzazioni di altri modelli, i quali potranno fornire a loro volta ulteriori informazioni.

Mentre il modello gestisce un aspetto più statico (dal punto di vista della frequenza di aggiornamento delle informazioni) del macchinario, le **operation** permettono di caratterizzare e rappresentare lo stato attuale delle attività svolte dalla singola unità produttività. Come per i precedenti data model, anche le operazioni sono caratterizzate da un identificativo e da ulteriori informazioni significative, come il nome dell’operazione e una sua descrizione. Nel caso di un macchinario composto, le sue operazioni tracceranno le operazioni di ciascun macchinario che

compone il sistema. Come per i macchinari con i machine model, anche le operation risultano essere delle istanze degli OperationModel.

La quarta entità presente nel diagramma, **OperationModel**, rappresenta i modelli delle operazioni, la quale contiene, oltre alle consuete informazioni generali, i riferimenti per l'esecuzione delle singole operazioni, e delle operazioni composite. Attraverso l'OperationModel, i gestori dei macchinari possono scegliere quale operazione eseguire sui macchinari.

È importante notare come nel diagramma precedente, la caratterizzazione dei modelli sia stata mantenuta semplice. Il fine di questa scelta è di esaltare le interazioni tra i vari elementi, invece della loro rappresentazione interna, poiché questa è descritta in maniera più precisa e completa negli *schema.json* dei relativi Smart Data Models.

Essendo le entità che possono operare con il prototipo risultare non standardizzate, ovvero presentare più o meno gli stessi attributi e relazioni, è stato scelto di non definire delle classi per ciascun data model, ma di operare indistintamente attraverso l'uso di mappe e liste. Un ulteriore motivo è la scelta di non implementare un database interno al prototipo ma di fare affidamento al context broker per le operazioni di immagazzinamento delle informazioni.

Esempio

In questa sezione sono riportati degli esempi di context information scambiate tra il prototipo, scorpio broker ed altri componenti laterali. Questi esempi fanno riferimento al concetto di macchinario composito, essendo quest'ultimo una estensione di tutti i DataModel presi in considerazione, siccome aggiunge a ciascuno un nuovo campo per la rappresentazione della struttura complessa.

ManufactoringMachine

Nella figura 11, viene proposto un esempio di MachineManufactoring in formato .jsonld, nello specifico la context information di una composizione di macchinari. L'esempio sottostante amplia la rappresentazione offerta dallo DataModels di riferimento (MachineManufactoring), introducendo la keyword "machineComponent". Quest'ultima permette di rappresentare la composizione di un macchinario, che nel nostro esempio è identificata come una trivella composta da tre elementi, attraverso un'associazione tra un macchinario che lo compone e il suo modello.


```
{
    "@context": [

"https://raw.githubusercontent.com/smart-data-models/dataModel.ManufacturingMachine/master/context.jsonld"
    ],
    "assetIdentifier": "ID12350",
    "description": "Composite Auger",
    "firstUsedAt": "2017-05-04T10:18:16Z",
    "id": "urn:ngsi-ld:Machine:9166c528-9c98-4579-a5d3-8068aea5d9b0",
    "installedAt": "2017-05-04T10:18:16Z",
    "location": {
        "coordinates": [
            -104.99404,
            39.75621
        ],
        "type": "Point"
    },
    "machineModel": "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd6490",
    "manufacturedAt": "2017-05-04T10:18:16Z",
```

```
    "power": 4.4,
    "advancementSpeed": 10,
    "supplierName": "ACME NorthEast Inc.",
    "machineComponent": {
        "value":{
          "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd65k9":
"urn:ngsi-ld:Machine:9166c528-9c98-4579-a5d3-8068aea5d7k0",
          "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd64t9":
"urn:ngsi-ld:Machine:9166c528-9c98-4579-a5d3-8068aea5d6c0",
          "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd69c3":
"urn:ngsi-ld:Machine:9166c528-9c98-4579-a5d3-8068aea5dq90"
        },
    "type":"Property"
    },
    "type": "ManufacturingMachine",
    "voltage": 220
}
```


Figura 11: Esempio di MachineManufactoring

MachineManufactoringModel

Di seguito, viene riportato un esempio di **MachineManufactoringModel** che descrive una trivella composta da più componenti. Rispetto al data model di riferimento, introduce i campi “componentMachineModel” e “operationalModel”.Il **componentMachineModel** mantiene tutti i riferimenti dei modelli dei macchinari di cui potrebbe essere composto, mentre **operationalModel** fornisce un associazione tra le azioni che il modello di macchinario può eseguire e l’OperationalModel (quest’ultimo sarà descritto successivamente).


```
{
 "@context": [

"https://raw.githubusercontent.com/smart-data-models/dataModel.ManufacturingMachine/ma
ster/context.jsonld"
      ],
    "id": "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd6490",
    "type": "ManufacturingMachineModel",
    "name": "Auger Composite Machine",
    "description": "A composite machine",
    "brandName": "Transformer",
    "version": "v1",
    "root": "false",
    "machineModelParent":
"urn:ngsi-ld:MachineModel:4146335f-839f-4ff9-a575-6b4e6232b734",
    "processDescription": "A composite machine that drill to construct tunnel",
    "componentMachineModel": {
    "type":"Relationship",
    "object":
    [
        "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd65k9",
        "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd64t9",
        "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd69c3"
```

```
    ]
    },
   "operationModel": {
        "type": "Property",
        "value": [
          { "Forward":
"urn:ngsi-ld:MachineOperationModel:27577638-bd8a-4732-b418-fc8b949a0t90"}
        ]
    }
}
```

Figura 12: Esempio di MachineManufactoringModel

ManufacturingMachineOperationModel

Rispetto a tutti i data model, che sono stati presi in considerazione in questo progetto, **OperationalModel,** è l'unico che non ha riferimenti stretti con nessun SmartDataModel. Lo scopo di questo data model è quello di rappresentare per il data model Operation, quello che MachineModel rappresenta per Machine, ovvero la rappresentazione di tutte quelle informazioni statiche, e comuni a tutte le operation. Nello specifico fornisce la tipologia di operazione che rappresenta, "operation", e i riferimenti alle operazioni che dovranno essere eseguite su tutti i modelli delle componenti che compongono il MachineModel (campo "machineModel"). Questo particolare campo risulta essere di particolare interesse nel prototipo per implementare la funzionalità di esecuzione di una operazione su un macchinario composito.

```
{
    "@context": [
"https://raw.githubusercontent.com/smart-data-models/dataModel.ManufacturingMachine/ma
ster/context.jsonld"
    ],
    "id": "urn:ngsi-ld:MachineOperationModel:27577638-bd8a-4732-b418-fc8b949a0t90",
    "type": "ManufacturingMachineOperationModel",
    "machineModel": "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd6490",
    "operationType": "process",
    "description": "Forward",
    "operation": "Forward",
    "operationComposite": {
        "type": "Property",
        "value":
            {
            "Forward": "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd69c3",
            "Install": "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd64t9",
            "Drill": "urn:ngsi-ld:MachineModel:00b42701-43e1-482d-aa7a-e2956cfd65k9"
            }

    }
}
```

Figura 13: Esempio di MachineManufactoringOperationalModel

ManufacturingMachineOperation

Nella figura 14 è descritto un esempio di operation eseguita su un macchinario multicomponente, il quale si discosta dallo smart data models di riferimento, siccome viene introdotto il campo **operationComposite**. Attraverso quest'ultimo è possibile mantenere traccia delle operazioni che sono eseguite sui suoi sotto macchinari.

```
{
    "@context": [
"https://raw.githubusercontent.com/smart-data-models/dataModel.ManufacturingMachine/master/context.jsonld"
    ],
    "id": "urn:ngsi-ld:MachineOperation:27577638-bd8a-4732-b418-fc8b949a0t90",
    "type": "ManufacturingMachineOperation",
    "machine": "urn:ngsi-ld:Machine:9166c528-9c98-4579-a5d3-8068aea5d6c0",
    "operationType": "process",
    "description": "Forward",
    "result": "ok",
    "plannedStartAt": "2016-08-22T10:18:16Z",
    "plannedEndAt": "2016-08-28T10:18:16Z",
    "status": "finished",
    "startedAt": "2016-08-22T10:18:16Z",
    "endedAt": "2016-08-28T10:18:16Z",
    "commandSequence": "Forward",
    "operationComposite": {
        "type": "Relationship",
        "object": [
            "urn:ngsi-ld:MachineOperation:27577638-bd8a-4732-b418-fc8b949a0b0f",
            "urn:ngsi-ld:MachineOperation:27577638-bd8a-4732-b418-fc8b949a0c7c",
            "urn:ngsi-ld:MachineOperation:27577638-bd8a-4732-b418-fc8b949a0t90"
        ]
    }}
```


Figura 14: Esempio di MachineManufactoringOperational

Analisi delle interazioni più significative

Essendo il context broker di tipo REST (stateless), le interazioni tra il client e il prototipo sono singole. Lato backend, le richieste sono processate principalmente tutte come di seguito.

Esecuzione di operazioni complesse (composite)

La figura 15 rappresenta il sequence diagram dell'esecuzione di una operazione complessa ovvero che prevede l'esecuzione di multiple operazioni sulle varie componenti del macchinario composito.

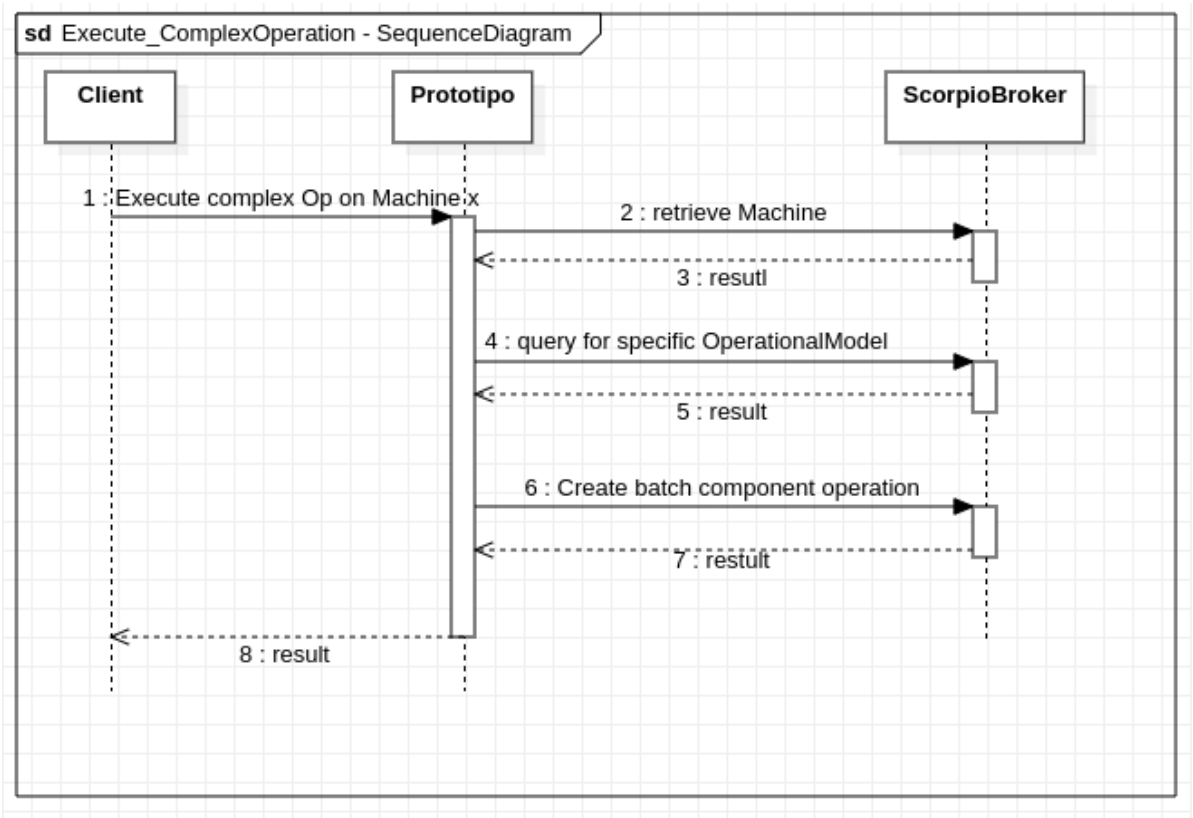

Figura 15: Sequence Diagram Esecuzione Operazioni Complesse

Esecuzione di un'operazione - interazioni ad alto livello

Nella figura 16 viene riportata la sequenza delle interazioni tra il prototipo, che si interfaccia con l'utente e il broker, e il macchinario il quale si registra sul broker, al fine di monitorare la creazione di nuove operazioni a lui riferite. Al momento dell'invio delle nuove operazioni da eseguire da parte del prototipo, Scorpio Broker notifica le nuove entità ai macchinari interessati.

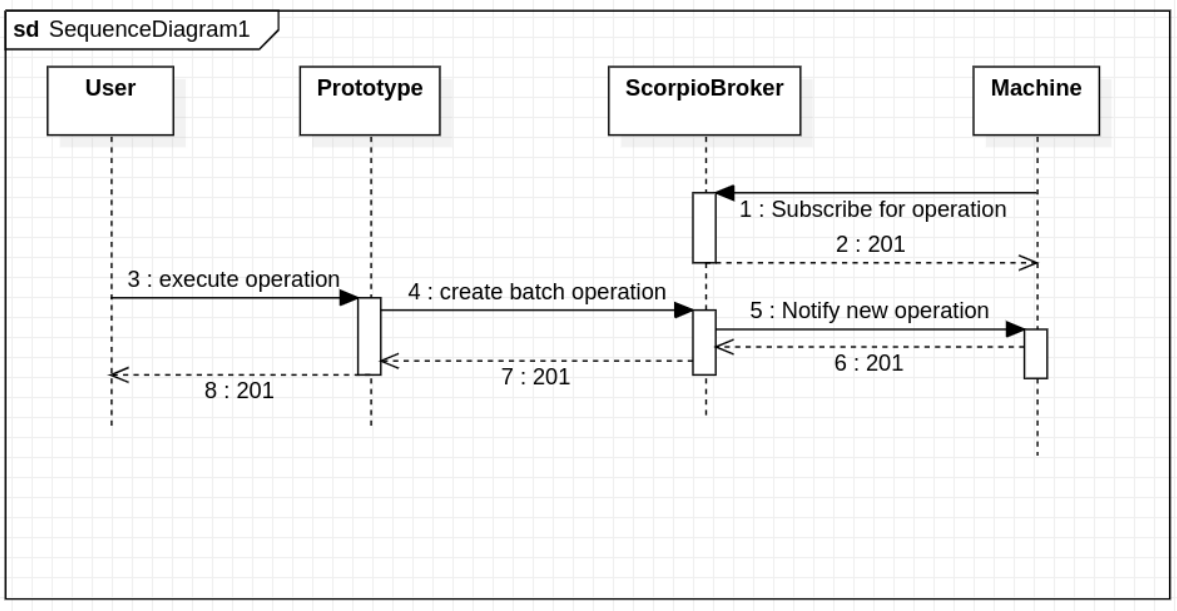

Figura 16: Sequence Diagram esecuzione di una operazione - interazione tra prototipo e macchinario

Architettura

Di seguito viene fornita una breve descrizione delle componenti principali in campo:

- **Prototype:** Sistema sviluppato per la gestione delle componenti base e composite. Le sue componenti principali sono:
    - **RESTController:** Componente che pubblica le API e gestisce le richieste REST
    - **ServiceLayer:** Componente che implementa la logica del servizio, eseguendo dalle operazioni CRUD a quelle più complesse
    - **RESTClient:** Componente dedito all'invio delle richieste al ContextBroker (e.g. ScorpioBroker).

- **ScorpioBroker:** Il prototipo basa il suo funzionamento sull'interazione con un ContextBroker, (in questo caso ScorpioBroker) per tutte quelle funzionalità di gestione delle Context Information e per le interazioni con le altre componenti.

- **Machine:** Per arricchire il contesto è stata sviluppata una semplice componente software che simula tutti i vari macchinari in campo.

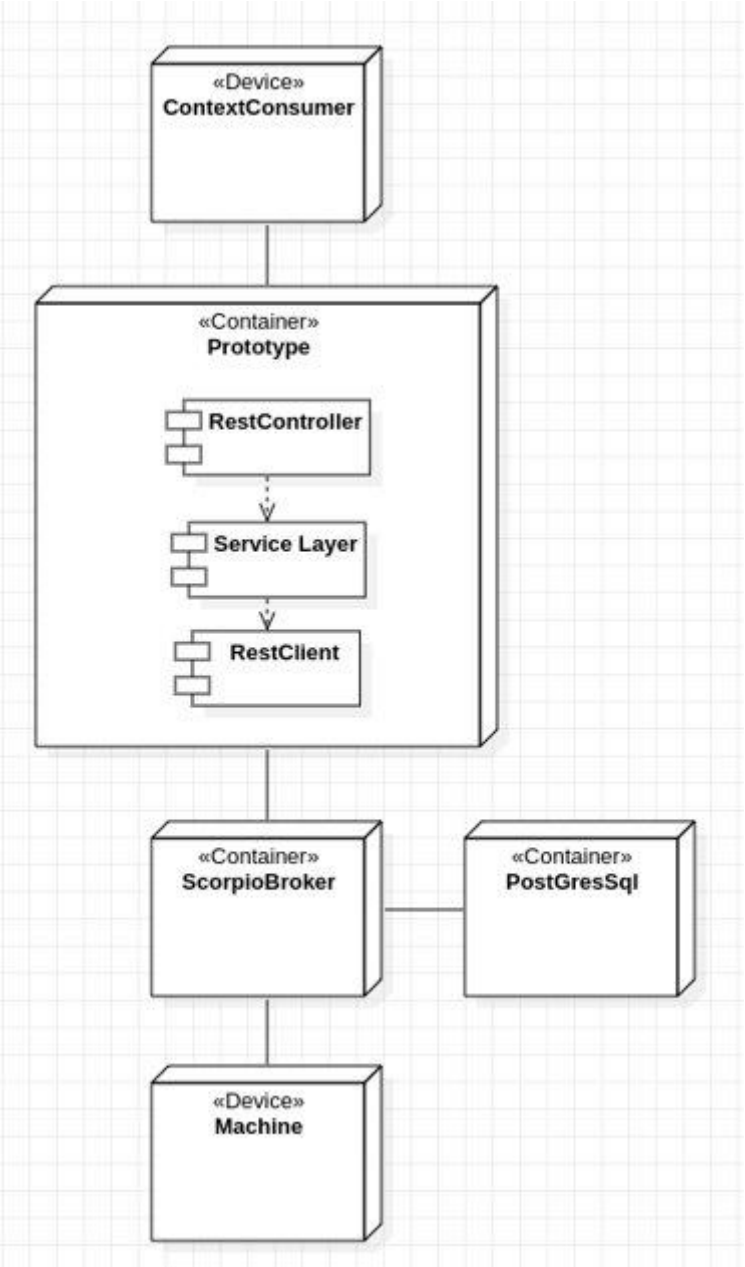

Figura 17: Deployment Diagram Prototipo

Il prototipo è stato sviluppato per lavorare in un ambiente con una frequenza di aggiornamento delle operazioni limitata, per questo motivo non è stato integrato nessun sistema di persistenza locale, ma è stato delegato a ScorpioBroker (o a un qualsiasi ContextBroker). Se l'ambiente industriale presentasse eventi che richiedano un alto livello di risposta dal sistema o la limitazione delle interazioni con il context broker potrebbe essere introdotto un database locale per la gestione delle richieste di lettura o le più frequenti e statiche.

Hands-on

Il progetto ha lo scopo di mettere in luce le capacità di FIWARE di permettere l'operazione tra sistemi diversi. Come già espresso in precedenza FIWARE concretizza questa capacità attraverso tutte le soluzioni già descritte come: SmartDataModels, NGSI-LD, ecc. Il risultato di queste scelte di design risultano in un disaccoppiamento tra gli attori stessi in funzionalità e scopi, astraendo le informazioni scambiate in strutture informative omogenee limitando

qualsiasi sorta di meccanismo di dipendenza o relazione. Come sarà dimostrato con questo progetto FIWARE permette la cooperazione all'interno dello stesso ambiente (ovvero sugli stessi componenti) di due sistemi diversi (macchinari composti e base), richiedendo minimi accorgimenti per l'integrazione dei due.

Di seguito saranno mostrati **N** passi che mostrano a livello operativo come sia realizzata l'interoperabilità:

- Rappresentazioni delle entità.
- Esecuzione delle operazioni e visione delle operazioni.
- Interazioni con il core component.

DataModels standard e personalizzati

Per dimostrare la flessibilità della tecnologia di FIWARE, nel progetto si è cercato di fornire una visione di questa tecnologia da un punto di vista non prettamente limitata al concetto di rappresentazione fisica dei digital twin o della loro simulazione ma a quello della gestione del sistema fisico da un punto di vista operativo (ovvero dell'esecuzione di operazioni).

Nel dominio della Smart Manufacturing, i data models: Machine, MachineModel, e Operation fanno riferimento allo stato attuale del macchinario per quanto concerne la sua configurazione, la sua locazione, il suo stato elettronico, e la sua attività attuale, fornendo effettivamente una visione fisica di un sistema reale. Da questa visione orientata alla simulazione del sistema stesso, il progetto si è concentrato nell'implementazione di tutto l'impianto di relazioni necessarie a fornire anche una gestione operativa.

Fermandosi a riflettere è possibile già intravedere un possibile effetto della libertà che la filosofia degli SmartDataModels introducono, ovvero la definizione di modelli armonizzati e standard al fine di permettere una interoperabilità, ma anche la possibilità di personalizzarli per i casi specifici. Come già ampiamente descritto, nella sezione Esempio, tutti i modelli introdotti per la rappresentazione dei macchinari compositi, si basano sui data model del dominio MachineManufactoring. Questa scelta si traduce nell'aver sviluppato modelli specifici che condividono un struttura centrale standard, rendendola così accessibile a tutte quelle componenti che operano con gli SmartDataModel. Solamente le componenti interessate a quelle estensioni nei data model, implementeranno tutte le logiche per sfruttare le informazioni aggiuntive a loro vantaggio.

Rappresentazione delle Entità

La rappresentazione di macchinari differenti, non si riflette nella strutturazione delle context information in data model diversi ma nel popolamento degli stessi modelli con dati specifici. Un parallelismo calzante per comprendere questo concetto potrebbe essere individuato nel dualismo tra classe e oggetto, dove la classe è identificata nel data model e l'oggetto, nella sua inizializzazione con i dati.

Nella figura 18 viene riportato un esempio semplificato di questa relazione tra data model e istanze nel contesto del prototipo. Nello specifico e visibile come le varie istanze siano il popolamento dei data model, machine e model. A loro volta per il context broker qualsiasi tipologia di istanza di data model viene gestita come semplice entità, ovvero una collezione di attributi e relazioni.

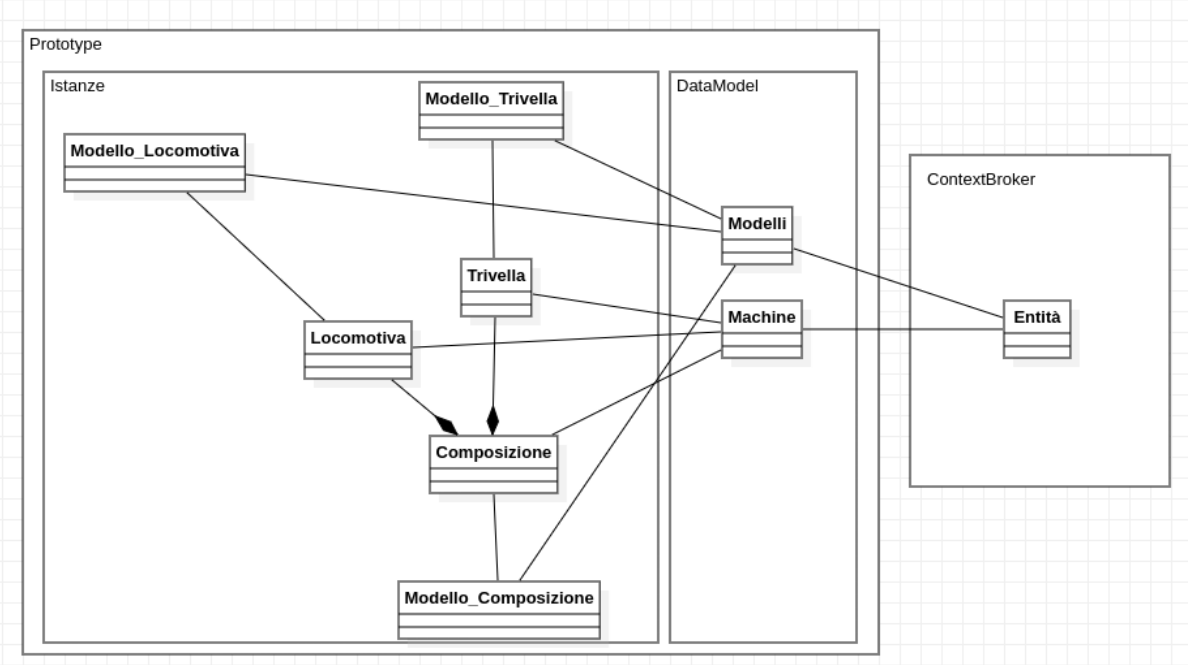

Figura 18: Rappresentazione della gestione delle ContextInformation

Siccome il ContextBroker non è interessato alla semantica dei vari campi dell'entità, ma che questi rispettino la sintassi, è compito degli stessi attori che interagiscono nel sistema a dover definire la semantica dei contenuti. Un esempio della gestione del ContextBroker delle entità e del suo approccio alla verifica delle stesse, è mostrata nella figura 19, dove sono state riportate le varie interfacce REST POST del Client REST implementate nel prototipo.

```
@POST
@Path("/entityOperations/create")
public Response postEntity(List<HashMap<String, Object>> operation);

@POST
@Path("/entities/")
public Response postEntity(HashMap<String, Object> entity);
```

Figura 19: Signature REST client POST Method

Come è accennato nella figura, ma descritto più approfonditamente dalle API esposte dal ContextBroker o ancora meglio dallo standard NGSI-LD, il context broker non necessità di essere a conoscenza della tipologia di entità ricevuta, ma verifica semplicemente che queesta rispetti la sintassi dello standard. Questo design già da parte del context broker permette l'interoperabilità trattando tutte le context information alla stessa maniera indistintamente dalla tipologia.

```
@GET
@Path("/entities/")
public List<Map<String, Object>> findAll(@QueryParam("type") String type);

@GET
@Path("/entities/{id}")
public HashMap<String, Object> findEntity(@PathParam("id") String id);
```

Figura 20: Signature REST client GET Method

Mentre nella figura 19 viene accennata la gestione da parte del contextbroker delle risorse, il prototipo è stato implementato tenendo conto della fluidità delle informazioni su cui operare, infatti come è osservabile nella figura 20 il prototipo gestisce le informazioni recuperate come Mappe di oggetti identificati da stringhe. Questa scelta permette così di ottenere pacchetti di informazioni dal contenuto mutevole sui quali per operare è necessario verificare la presenza degli attributi (o delle chiavi) di cui si ha interesse. Un ulteriore esempio di questa gestione è visibile nella figura 21, dove viene inoltrata una richiesta di tutte le entità di tipo ManufactoringMachine, le quali sono gestite successivamente come liste di mappe di stringhe (le proprietà) e di oggetti (il contenuto delle proprietà).

```
public List<Map<String, Object>> getAllMachine() {
                // TODO Auto-generated method stub
                return restClient.findAll("ManufacturingMachine");
        }
```

Figura 21: Funzione GET per recuperare tutte le entità di tipo ManufactoringMachine

Esecuzione delle operazioni

Come già anticipato, l'applicazione ha il suo scopo principale di creare un command and control centralizzato di sistemi complessi.

In sintesi il sistema opera, recuperando all'inizio tutte le informazioni necessarie ovvero il macchinario su cui eseguire le operazioni e il modello dell'operazione eseguibile sul quel modello di macchinario, e a seconda della tipologia, ovvero macchinario di base o composito, sono create tutte le operazioni necessarie all'esecuzione della richiesta.

Di seguito viene riportata un'analisi dell'implementazione di questa funzionalità (per una maggiore comprensione il metodo viene suddiviso in sezioni per una maggiore comprensione).

Recupero delle informazioni e verifica della consistenza rispetto al compito

```
public Response executeOperation(String machineUri, String action, HashMap<String, Object>
details)throws Exception {
        HashMap<String, Object> machine = restClient.findEntity(machineUri);
        if (machine.get("type").toString().compareTo("ManufacturingMachine") != 0) {1
                throw (new Exception("Entity retrieved is not a ManufacturingMachine"));
        }
        HashMap<String, Object> operationalModel =restClient.findEntityByFilter(2
                                Map.of("type", "ManufacturingMachineOperationModel"),
createFilter("description", action), createFilter("machineModel", getMap(machine,
"machineModel").get("value").toString())).get(0);
```

Figura 22: Recupero informazioni necessarie

Nella figura 22 viene mostrato la prima fase di recupero delle entità necessarie all'esecuzione della funzione, ovvero del macchinario su cui sarà eseguito la funzione, e successivamente una identificazione del ManufactoringMachineOperationalModel, riferito all'azione da eseguire su quello specifico modello di macchinario.

Analizzando lo snippet sotto la lente dell'interoperabilità, è possibile notare due diverse strategie di imposizione del tipo: nel 1 è lo stesso prototipo che ne verifica il tipo (ManufactoringMachine) nel 2 viene delegato al context broker attraverso l'imposizione di filtri.

Creazione delle operazioni

Nella figura 23 viene descritto la seconda parte della funzione, ovvero la creazione ed esecuzione delle operazioni sulle componenti del macchinario.

- 0 Verifica che il modello dell'operazione ottenute sia un modello di un macchinario composito o no. Questa informazione è inferibile dalla presenza del campo operationComposite;
- 1 vengono identificate tutte le operazioni associate al modello delle sottocomponenti;
- 2 sono identificati tutte le componenti base che compongono l'entità complessa e i loro modelli;
- 3 sono create le operazioni per ciascuna componente attraverso le informazioni precedentemente identificate.

- 4 Successivamente viene completata la lista delle operazioni da eseguire e inviate al context broker, che si occuperà di consegnare ai rispettivi macchinari.

```
if (operationalModel.containsKey("operationComposite")) {0
        Map<String, Object> elements = getValueOfPropertyField(operationalModel,
"operationComposite");1
        Map<String, Object> machineComponent = getValueOfPropertyField(machine,
"machineComponent");2
        List<HashMap<String, Object>> operationList = elements.entrySet().stream()
                .map((Entry<String, Object> x) -> {return
createOperation(machineComponent.get(x.getValue()).toString(), x.getKey(),
details);}).collect(Collectors.toList());3
        operationList.add(addListOfId(createOperation(machineUri, action, details),
operationList, "operationComposite"));
                        return restClient.postEntity(operationList);4
                }
```

Figura 23: Creazione delle operazioni in una entità complessa

In questo snippet è possibile notare come non siano stati effettuati determinati controlli che risultano essere necessari per garantire la consistenza delle informazioni consegnate, come la verifica dell'effettiva esistenza dei macchinari che lo compongono per esempio. Questa scelta è delegata alla singola componente software al di fuori del context broker, dando così la possibilità al singolo progettista di identificare il giusto rapporto tra libertà di esecuzione, semplicità della logica e imposizione della consistenza.

Meccanismo di sottoscrizione e notifica a una entità

ScorpioBroker, ma in generale tutti i broker compliant con la NGSI-LD, predispongono una interfaccia REST per permettere la sottoscrizione a una risorsa, per quanto concerne le operazioni di creazione e di modifica (non la cancellazione) di una entità o attributo.

Il meccanismo prevede la creazione da parte del richiedente (in questo caso del macchinario) di sottoscriversi al sistema attraverso il POST del file JSON secondo la figura 24, indicando i fattori filtranti 1 (tecnicamente NGSI-LD e scorpio broker predispongono molti più parametri per la personalizzazione delle richieste) e l'endpoint al quale inviare le notifiche 2

```
{
  "id": "urn:subscription:1",
  "type": "Subscription",
  "entities": [{
```

```
            "type": "ManufactoringMachineOperation"1
  }],
  "notification": {
       "endpoint": {
            "uri": "http://172.17.0.1:8090/notification",2
            "accept": "application/json"
       }
  },
  "@context": ["https://pastebin.com/raw/Mgxv2ykn"]
}
```

Figura 24 JSON+LD Subscription request

Il context broker comunicherà all'endpoint 2, secondo le modalità richieste, ovvero HTTP o MQTT gli eventuali cambiamenti nelle context information. La notifica avrà la seguente forma:

```
{
  "id" : "notification:-5808910960552836161",1
  "type" : "Notification",
  "subscriptionId" : "urn:0ce864b3-766f-4a31-9a00-89bf3d0c70f4",
  "notifiedAt" : "2023-10-25T08:02:02.653000Z",
  "data" :2
[ {
    "id" : "urn:ngsi-ld:MachineOperation:317b26cd-ccdc-45cc-9002-5ee426316763",
    "type" : "ManufacturingMachineOperation",
    "description" : {
      "type" : "Property",
      "value" : "Drill"
    },
    "machine" : {
      "type" : "Property",
      "value" : "urn:ngsi-ld:Machine:9166c528-9c98-4579-a5d3-8068aea5d7k0"
    }
  },{..}
]
}
```

Figura 25 JSON+LD Notification answer

Nello specifico 1 identifica l'intestazione della notifica con l'identificativo della notifica e della subscription e al 2 una lista di tutte le variazione nelle contest information.

Sezione @Context

Come è emerso dalle sezioni precedenti, l'ecosistema FIWARE, ma nello specifico i core context, garantiscono l'interoperabilità fornendo un alto grado di libertà nell'interazione con le context information. Da questo presupposto anche la stessa gestione della sezione @Context è liberamente gestita, ovvero il core context si occupa di associare a tutte le context information ricevute e nello specifico ai campi standard di associarsi il dizionario standard degli smart data models. Per tutti gli altri termini non standard invece la loro gestione è direttamente demandata ai singoli attori che interagiscono, i quali possono o non fornire un dizionario dei termini personalizzati presenti. Data questa possibilità, nel progetto è stato scelto di fornire semplicemente il dizionario di riferimento per il dominio delle MachineManufactoring (https://raw.githubusercontent.com/smart-data-models/dataModel.ManufacturingMachine/master/context.jsonld) , ma non per quei termini personalizzati al fine di: da una parte avere uno sviluppo più semplificato potendo non sviluppare un sistema di personalizzazione interattivo del dizionario e dall'altra di poter effettivamente testare un ulteriore livello di libertà permesso.

La definizione di dizionario specifico non risulta essere particolarmente complessa ma avrebbe richiesto il suo hosting su una piattaforma esterna al fine di generare un link data da condividere nella rete. Oltre a fornire una semantica per i termini condivisi, i dizionari condivisi nella sezione @Context (o anche nel header Link) sono impiegati da parte del Context Broker per espandere e contrarre i termini. Per espansione dei termini si intende la sostituzione del termine compatto con il suo linked data. Al momento di una richiesta GET se non è fornito il dizionario, sono ritornate le informazioni nel formato espanso (per tutti i termini non standard) demandando così all'interlocutore la compattazione dei termini . Per quanto riguarda le attività di POST o comunque di invio di informazioni al context broker, la mancanza dei dizionari di riferimento, si traduce nella non espansione dei termini. A differenza delle richieste POST, che prevedono due diverse metodologie di invio dei riferimenti ai dizionari (come @Context o come Link Data) il metodo GET prevede solo l'invio attraverso l'header Link. Da questa limitazione è necessario che il riferimento al dizionario sia o ospitato da un server predisposto o è possibile utilizzare strumenti come pastebin.

Conclusioni

Lo scopo di questa relazione era quello di fornire una visione chiara e comprensibile di come FIWARE stia gettando le basi per lo sviluppo di un ambiente aperto e, più specificamente, di un internet dei digital twins. Per capire come questo fosse possibile, è stato necessario fornire una breve definizione di cosa fosse un digital twins.

FIWARE si è occupata di stilare un framework per lo sviluppo di soluzioni in ambito dei Digital Twin, che permettano la concretizzazione di questa tecnologia, con il non solo scopo di standardizzare un processo per la realizzazione dei digital twin, ma anche quello di creare un ambiente distribuito, aperto e interoperabile per la condivisione dei risultati raggiunti. Per raggiungere questo obiettivo, FIWARE ha sviluppato il suo framework su questi tre concetti base: architettura, linguaggio e modello.

Come ampiamente descritto, FIWARE consiglia una architettura precisa per lo sviluppo dei sistemi, ovvero CoBrA (Context Broker Architecture). Questa architettura presenta una componente centrale, o core component, che gestisce tutte le interazioni con i producer e consumer. Tuttavia, FIWARE allarga il concetto permettendo, da una parte, la federazione degli stessi Context Broker, creando così vere e proprie overlay network composte da CoBr, e dall'altra, permettendo una comunicazione diretta tra producers e consumers.

FIWARE non chiude il suo ecosistema a possibili integrazioni a componenti legacy o di altri attori, ma permette l'interoperabilità fornendo le regole comunicative per realizzarla, ovvero il linguaggio. In collaborazione con ETSI, FIWARE ha redatto l'attuale standard ETSI NGSI-LD per il management delle context information. Questo standard definisce: gli attributi, le entità, le relazioni, i termini che compongono i contenuti scambiati, ma anche le interazioni tra le componenti e tutte le relative api.

A conclusione di questa strategia, gli SmartDataModels rappresentano l'ultimo passo alla interoperabilità dei sistemi. Con questa iniziativa FIWARE permette la condivisione di modelli studiati e raffinati con la comunità, al fine di permettere una più facile condivisione delle informazioni e una maggiore riusabilità, riducendo i costi di sviluppo ex novo. Al tempo stesso, gli SmartDataModels permettono l'estensione con elementi affini al campo di utilizzo, ma specifici alla realtà in cui un attore interagisce. Questa capacità è resa possibile attraverso il concetto di Linked Data e di @Context, i quali permettono di associare alle informazioni trasmesse la semantica dei termini che compongono un modello (es di attributi e di relazioni), riducendo così possibili fraintendimenti con un interlocutore.

Concludendo, il progetto è stato coronato con un semplice prototipo con lo scopo di evidenziare le capacità di interoperabilità da parte del framework FIWARE, attraverso l'uso del context broker ScorpioBroker. Nel progetto, il context broker si presenta come da architettura FIWARE,

il core component, fornendo tutte le funzionalità di persistenza, gestione delle risorse e sistema di notifica.

Proprio nella gestione delle informazioni, è stato possibile mettere in risalto l'interoperabilità del framework, ovvero è stata evidenziata la libertà lasciata ai consumer e ai producer, o più in generale agli attori che compongono la rete, di gestire i propri dati. La gestione autonoma permette da un lato la possibilità di utilizzare gli SmartDataModels come strutture dati condivise e universalmente riconosciute e dall'altro di poter estendere queste strutture personalizzandole e sviluppando modelli nuovi. La massima dimostrazione di questa libertà è la stessa specifica del context broker (ovvero l'implementazione dello stesso Context Information Management descritto dal NGSI-LD) che gestisce tutte le context information indistintamente dal loro contenuto o dominio di appartenenza, risultando uno strumento particolarmente versatile.

La fluidità delle strutture dei data model con cui il prototipo (o un attore in generale) devono interagire, guidano il design del progetto nella sua gestione nella rappresentazione delle informazioni all'interno delle componenti. Se da una parte è consigliabile trattare questa varietà di strutture, dall'altra si rende necessario un restringimento del numero di attributi su cui operare.

Proprio da questa parzialità sull'utilizzo dei data models è possibile massimizzare l'interoperabilità, siccome se ogni attore opera solo su quei contenuti a lui interessanti, diminuisce la superficie eterogenea dei dati, aumentando l'operabilità della stessa componente dovendo verificare la consistenza di una minor quantità di dati.

Concludendo Fiware sta raggiungendo il suo obiettivo di creare un internet of Digital Twin fornendo una serie di strumenti base con cui creare l'effettiva rete (Federazione di Context Broker e SmartDataModels) che gestiscono le context information lasciando un alto grado di libertà agli attori che interagiscono, delegando a loro il compito di determinare il loro livello di interoperabilità, attraverso la loro implementazione, con il resto del sistema.

Annex A - Esempio completo dello schema ManufacturingMachine

Di seguito è fornito l'esempio completo dello schema.json dello Smart Data Models ManufacturingMachine. La versione qua fornita è una versione ridotta a solo quei campi presenti nel Esempio di una instanza Machine Manifacturing.

Lo schema originale completo è visibile ai seguenti link:

- https://smart-data-models.github.io/dataModel.ManufacturingMachine/ManufacturingMachine/schema.json
- https://github.com/smart-data-models/dataModel.ManufacturingMachine/blob/master/ManufacturingMachine/schema.json


```
{
    "$schema": "http://json-schema.org/schema",
    "$schemaVersion": "0.0.1",
    "$id":
"https://smart-data-models.github.io/dataModel.ManufacturingMachine/ManufacturingMachine/schema.json",
    "modelTags": "GSMA",
    "title": "Smart Data models - Manufacturing Machine dataModel schema,",
    "description": "Description of a generic machine",
    "type": "object",
    "required": [ "id", "type" ],
```

```
    "allOf": [
        {
            "$ref":
"https://smart-data-models.github.io/data-models/common-schema.json#/definitions/GSMA-C
ommons"
        },
        {
            "$ref":
"https://smart-data-models.github.io/data-models/common-schema.json#/definitions/Location-
Commons"
        },
        {
            "properties": {
                "type": {
                    "type": "string",
                    "description": "Property. NGSI entity type. It has to be ManufacturingMachine",
                    "enum": [
                        "ManufacturingMachine"
                    ]
                },
                "machineModel": {
                    "anyOf": [
                        {
                            "type": "string",
                            "minLength": 1,
                            "maxLength": 256,
                            "pattern": "^[\\w\\-\\.\\{\\}\\$\\+\\*\\[\\]`|~^@!,:\\\\]+$",
                            "description": "Property. Identifier format of any NGSI entity"
                        },
                        {
                            "type": "string",
                            "format": "uri",
                            "description": "Property. Identifier format of any NGSI entity"
                        }
                    ],
                    "description": "Relationship. A reference to the associated Machine Model for this
machine"
                },
                "countryOfManufacture": {
                    "type": "string",
                    "description": "Property. Model:'https://schema.org/Text'. The country where this
machine was manufactured"
                },
                "installedAt": {
                    "type": "string",
                    "format": "date-time",
                    "description": "Property. Model:'https://schema.org/Text'. Indicates the date/time
at which date and time the machine was installed (nominally in UTC)"
                },
                  "osVersion": {
                    "type": "string",
                    "description": "Property. Model:'https://schema.org/Text'. The (manufacturer
specific) operating system version of this machine"
                },
                "supportedProtocol": {
                    "type": "array",
```

```
                "description": "Property. Model:'https://schema.org/Text'. Supported protocol(s) or networks",
                "items": {
                    "type": "string"
                }
            },
            "building": {
                "anyOf": [
                    {
                        "type": "string",
                        "minLength": 1,
                        "maxLength": 256,
                        "pattern": "^[\\w\\-\\.\\{\\}\\$\\+\\*\\[\\]`|~^@!,:\\\\]+$",
                        "description": "Property. Identifier format of any NGSI entity"
                    },
                    {
                        "type": "string",
                        "format": "uri",
                        "description": "Property. Identifier format of any NGSI entity"
                    }
                ],
                "description": "Relationship. Reference to the building entity instance into which this machine is installed"
            },
            "voltage": {
                "type": "number",
                "description": "Property. Model:'https://schema.org/Number'. The nominal required supply voltage, in volts. Units:'Volts"
            },
            "rotationalSpeed": {
                "type": "number",
                "description": "Property. Model:'https://schema.org/Number'.  \tThe maximum rotational speed in rpm (for machines such as drills, lathes). Units:'rpm"
            }
        }
    }
]
}
```


Figura 26: schema.json del modello MachineManufacturing