Smart IoT Service Builder Platform

Filipe Cruz

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Software Engineering

Supervisor: Dr. Nuno Silva

Evaluation Committee:
President:
TODO, Professor, DEI/ISEP

Members:
TODO, Professor, DEI/ISEP
TODO, Professor, DEI/ISEP
TODO, Professor, DEI/ISEP

Porto, July 13, 2022

Dedicatory

TODO

Abstract

Today there are more smart devices than people. According to **statista-number-devices** the number of devices worldwide is forecast to almost triple from 8.74 billion in 2020 to more than 25.4 billion devices in 2030.

The Internet of Things (IoT) is the connection of millions of smart devices and sensors connected to the Internet. These connected devices and sensors collect and share data for use and evaluation by many organizations. Some examples of intelligent connected sensors are: GPS asset tracking, parking spots, refrigerator thermostats, soil condition and many others. The limit of different objects that could become intelligent sensors is limited only by our imagination. But this devices are mostly useless without a platform to analyse, store and present the aggregated data.

Recently, several platforms have emerged to address this need and help companies/governments to increase efficiency, cut on operational costs and improve safety. Sadly, most of this platforms are tailor made for the devices that the company offers. This dissertation presents a platform and its development that assembles multiple services related to IoT into a single application. All the services provided by this platform attempt to be sensor-neutral and are to be exhibited under the same unified application.

Keywords: Internet of Things, Stream Processing, Big Data, Configurability, Real Time Systems

Resumo

Atualmente, existem mais sensores inteligentes do que pessoas. De acordo com **statista-number-devices**, o número de sensores em todo o mundo deve quase triplicar de 8,74 bilhões em 2020 para mais de 25,4 bilhões em 2030.

O conceito de IoT está relacionado com a interacção entre milhões de dispositivos inteligentes através da Internet. Estes dispositivos e sensores conectados recolhem e disponibilizam dados para uso e avaliação por parte de muitas organizações. Alguns exemplos de sensores inteligentes e seus usos são: dispositivos GPS para rastreamento de activos, monitorização de vagas de estacionamento, termostatos em arcas frigoríficas, condição do solo e muitos outros. O número de diferentes objectos que podem vir-se a tornar sensores inteligentes é limitado apenas pela nossa imaginação. Mas estes dispositivos são praticamente inúteis sem uma plataforma para analisar, armazenar e apresentar os dados por eles agregados.

Recentemente, várias plataformas surgiram para responder a essa necessidade e ajudar empresas/governos a aumentar a sua eficiência, reduzir custos operacionais e melhorar a segurança dos espaços e negócios. Infelizmente, a maioria dessas plataformas é feita à medida para os dispositivos que a empresa em questão oferece. Esta tese apresenta uma plataforma que permite a criação e agregação de vários serviços relacionados com IoT num ambiente único. Todos os serviços fornecidos por esta plataforma procuram ser agnósticos em relação aos dispositivos inteligentes suportados.

Acknowledgement

TODO

Contents

List of Figures

List of Tables

List of Algorithms

List of Source Code

List of Symbols

a	distance	m
P	power	W ($\mathrm{J\,s^{-1}}$)
ω	angular frequency	rad

Chapter 1

Introduction

Chapter 2

State of the Art

2.1 Internet of Things

2.1.1 Brief Description

2.1.2 Practical Applications

2.1.3 Enterprise Challenges

2.1.4 Renowned Solutions

2.2 Big Data

2.2.1 Brief Description

2.2.2 Challenges

2.3 Synopsis

Chapter 3

Analysis

3.1 Business Analysis

3.1.1 Fleet Management

3.1.2 Smart Irrigation

3.1.3 Fire Outbreak Surveillance

3.2 Technical Analysis

3.2.1 Data Aggregation

3.2.2 Data Filtering

3.2.3 Data Storage

3.2.4 Data Transformation

3.2.5 Data Analysis

3.2.6 Data Presentation

3.2.7 Trigger Warning System

3.2.8 User Authentication/Authorization

3.3 Synopsis

Chapter 4

Requirements Elicitation

4.1 Functional Requirements

4.2 Non Functional Requirements

4.3 Synopsis

Chapter 5

Design

This section goal is to describe the overall system design to the reader. First the reference architectures used for this project will be presented. Then the various system scopes will be introduced, followed by section regarding the domain model. After this the system's architectural design will be presented and major decisions/alternatives discussed. At last, a synopsis of this chapter can be read.

5.1 Reference Architecture

5.2 System Scopes

The system designed can be divided is three main scopes as disclosed in the Figure 5.1.

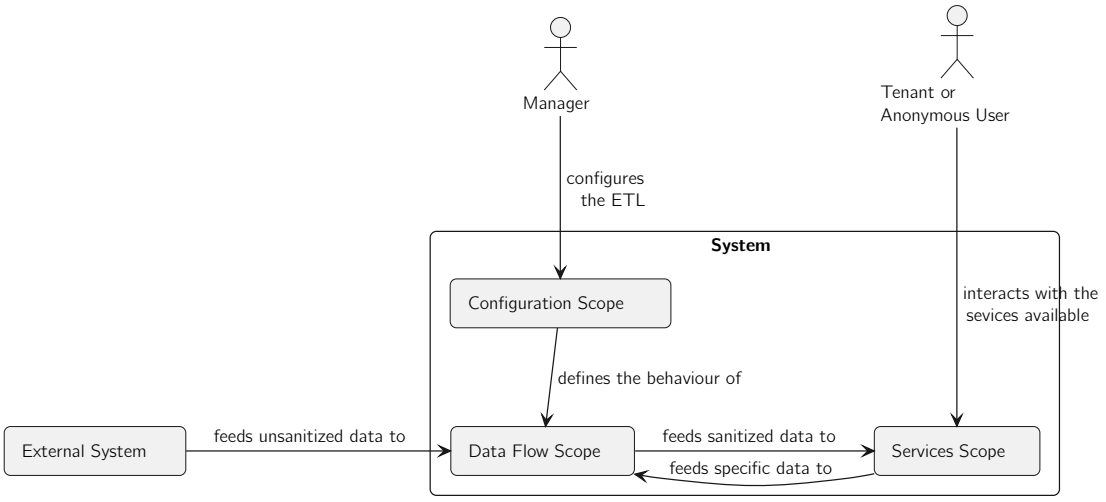

Figure 5.1: System Scopes

The **Configuration Scope** adheres to the configuration and visualization of internal processes/contexts. This processes, such as: (i) data decoders, (ii) device inventory, (iii) warning rules definition and (iv) device ownership, are related to the **Data Flow Scope**. It is also possible to manage tenants' access and permissions in the system in this scope.

The **Data Flow Scope** behaves according to what is defined in the **Configuration Scope** and acts as a pipeline where raw device data goes though various stages till it is sanitized and ready to be supplied to the **Services Scope**. The **Data Flow Scope** is where internal

processes occur, such as: (i) data transformation, (ii) data enrichment, (iii) data validation, (iv) data ownership clarification and (v) warnings dispatching.

The **Services Scope** is comprised of services that present and act according to the sanitized data that was supplied to them. This services applicability range from (i) smart irrigation, (ii) fleet management, (iii) fire detention, (iv) physical security access monitoring, (v) air quality monitoring and anything else deemed interesting.

5.2.1 Configuration Scope

The **Configuration Scope** is responsible for managing the following contexts:

- **Data Processor**: manages simple data mappers;
- **Data Decoder**: manages scripts to transform data;
- **Device Management**: manages device information such as name, metadata, static data and other notions;
- **Identity Management**: manages device ownership and users permissions;
- **Rule Management**: manages scripts that consume device data and produce alerts.

Each context allows an authorized user to manage its resources, e.g. the data processor context manages the creation, deletion and renovation of data mappers.

This operations require various verifications, alter the system internal state and are therefore prolonged operations.

5.2.2 Data Flow Scope

The **Data Flow Scope** is responsible for processing incoming data according to what is defined in the **Configuration Scope**. Both scopes share the same contexts, apart from the data validation context (only present in this scope).

The data validation context preforms basic data filtering based on static rules, e.g. battery percentage reported has to be in between 0 and 100.

This scope applies changes to the device data that flows though the system. This changes are stateless and don't change the overall state of the internal system state.

This scope was decoupled from the **Configuration Scope** even though they both work with the same contexts. The decision was taken based on the pretext that despite the similarities in context the operation/business processes of this two scopes were conflicting.

The **Configuration Scope** requires scarce but heavy computations that alter the internal system state while the **Data Flow Scope** requires plentiful but light computations that don't alter the internal system state as summarized in the Table 5.1.

Due to this discrepancy it's expected for each scope to have different requirements regarding horizontal scaling. With the addition of more devices to the platform, and subsequently higher ingress volume, **Data Flow Scope** will need to scale. Since the **Configuration Scope** is intended mostly for the manager of the platform, a small user pool, the need to scale is smaller.

Comparison of Operations	Configuration Scope	Data Flow Scope
Alter internal system state	yes	no
Alter sensor data	no	yes
Required computation power/time	high	low
Frequency of usage	low	high

Table 5.1: Comparison of Operations in Data Flow and Configuration Scopes

5.2.3 Service Scope

The **Service Scope** is responsible for presenting Internet of Things (IoT) business cases to end users. This scope is comprised of services that consume and publish data to **Data Flow Scope**. Currently, as a Minimal Value Product (MVP) the following business cases implemented are:

- **Fleet Management**: basic service to monitor a fleet of cars regarding their location;
- **Smart Irrigation**: service to automate and monitor the irrigation of zones based on sensor readings;
- **Notification Management**: service to view and manage the delivery of triggered alerts.

Each service is bounded to what type of data receives and sends back to the **Data Flow Scope** as detailed in Sections 5.3.1 and 5.3.2.

5.2.4 Synopsis

This section introduces the system as three separated scopes each with different needs and responsibilities. Despite this they all have a common domain model. The Section 5.3 addresses this shared domain and each context peculiarity.

5.3 Domain

This system's domain model will be discussed here. The idea behind this section is to introduced core business concepts to the reader and explain how they map to the contexts present in the system. To represent this ideas the Unified Modeling Language (UML) notation is used.

This section is split into four pieces: (i) concepts, (ii) shared model, (iii) bounded contexts and (iv) synopsis.

5.3.1 Concepts

In order for the reader to better understand how the system functions some concepts need to be better explained:

- **Device**: A device is a "Thing" that can collect data and submit it to **Sensae Console** via an external system though **Uplink**s. A device can, optionally, receive **Downlink**s;
- **Controller**: A controller is a **Device** that controls and aggregates data from various **Device**s;

- **Records/Metadata**: Records, or Metadata are labels associated to a **Device** that help an organization to classify and add some context to the owned **Device**s;
- **Downlink**: A downlink is a term commonly used in radio communications to denote the transmission from the network to the end user. In this case the network is the **Sensae Console** and the end user is a **Device**;
- **Uplink**: An uplink is the opposite of a **Downlink**, it's the transmission from a **Device** to the **Sensae Console**;
- **Data Unit**: A device data or measure is the collected data that is submitted via an **Uplink** to the **Sensae Console**. This data should be, at least, enriched with an unique identifier of the **Uplink** and **Device** that sent it;
- **Device Command**: A device command is an abstraction on top of a **Downlink** intended to order a **Device** to execute a specific action. As an example, one could send a command to open or close a valve that is incorporated into a **Controller**;
- **Decoder**: A decoder is a function that translates a **Data Unit** into something that **Sensae Console** understands;
- **Domain**: A domain represents a department in a organization. An organization is composed of several domains structured in a tree like format;
- **Tenant**: A tenant is a user that belongs to one or more **Domain**s;
- **Alert/Warning**: A report about a detected condition based on the gather **Data Unit**;
- **Topic**: A Topic is a subcategory of the type of contents that are traded between the various containers in the system.

Currently the **Topic**s that flow in the system are:

- **Data**: This topic references the **Data Unit** concept and is intended to be consumed by the **Service Scope**;
- **Command**: This topic references the **Device Command** concept and is intended to be used mainly by the **Service Scope**;
- **Alert**: This topic references the **Alert/Warning** concept and is intended to be consumed mainly by the **Service Scope**;
- **Internal**: This topic references the internal state maintained in the **Configuration Scope** and **Data Flow Scope**.

This concepts are referenced across the document.

5.3.2 Shared Model

The shared model is comprised of concepts that transverse the entire **Sensae Console** business model. Therefore, it is built as a separated project, *iot-core*, that can be imported in each micro service. It is comprised of three big components: (i) data model, (ii) message envelop model, and (iii) routing model.

Data Model

The data model represents the **Data Unit** that **Sensae Console** is currently capable of understanding. The following diagram, Figure 5.2, introduces a high level specification of it.

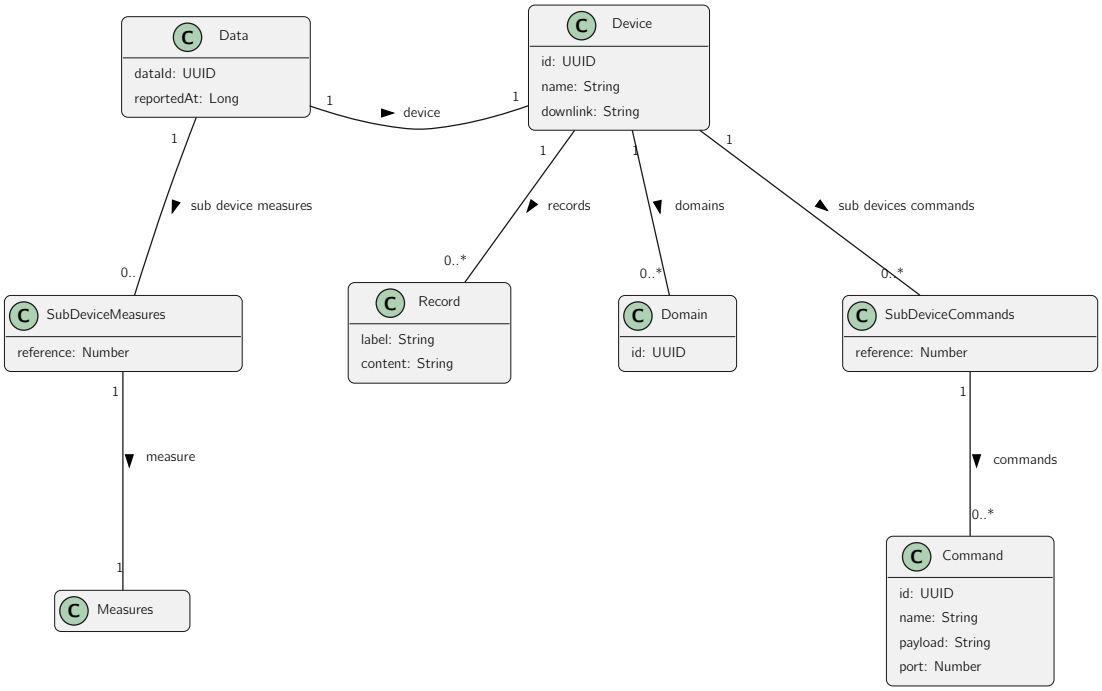

Figure 5.2: Shared Model

As a brief description:

- **Data Unit** is represented in the diagram as *Data* and is the entry point to the shared model;
- The *reportedAt* field represents the unix timestamp when the **Data Unit** was captured, in milliseconds;
- The *Device* concept represents the **Device** that sent the *Data*, and therefore the *Data*;
- The *Record* concept represents an entry of **Records/Metadata**;
- The *Domain* concept references the **Domain** that owns the *Device*;
- The *SubDeviceMeasures* concept introduces an approach to handle **Controller**s by mapping readings captured by a sub device to a *reference* that can later be resolved;
- The *SubDeviceCommands* concept introduces an approach to handle **Controller**s by mapping commands tailored for a sub device to a *reference* that can later be resolved;
- The *Measures* concept contains various common data types related to IoT.

As explained, *Measures* contains various data types. Currently the supported types are presented in the Table 5.2.

Data Type *Property*	*Sub Property*	**Description**	**Unit**
GPS		Point reference in the Geographic Coordinate System	
gps	*latitude*	Value between -90 and 90 measured in	degrees
	longitude	Value between -180 and 180 measured in	degrees
	altitude	Value determined according to the mean sea level	meters
Motion		Status related to the motion of a device	
motion	*value*	Value can be "ACTIVE", "INACTIVE" or "UNKNOWN"	n.a.
Velocity		How fast a device is moving	
velocity	*kilometerPerHour*	Value measured in	km/h
Temperature		Temperature measured by a device	
temperature	*celsius*	Value measured in	celsius
AQI		Air Quality Index according to the U.S. AQI	
aqi	*value*	Value measured in	AQI
Air Humidity		Concentration of water vapour present in the air	
airHumidity	*gramsPerCubicMeter*	Value measured in	g/m3
	relativePercentage	Value measured in	%
Air Pressure		Pressure within the atmosphere of Earth	
airPressure	*hectoPascal*	Value measured in	hPa
Battery		Battery of the device	
battery	*volts*	Value measured in	volts
	percentage	Value measured in	%
	maxVolts	Minimum volts the battery needs for the device to work	volts
	minVolts	Maximum volts the battery can hold	volts
Soil Moisture		Amount of water, including water vapor, in an unsaturated soil	
soilMoisture	*relativePercentage*	Value measured in	%
Illuminance		Illuminance level - luminous flux per unit area	
illuminance	*lux*	Value measured in	lux
Trigger		Type related to something with an on / off or open / close state	
trigger	*value*	Value true or false	boolean

Table 5.2 continued from previous page

Data Type *Property*	*Sub Property*	**Description**	**Unit**
CO2		Atmospheric Carbon Dioxide concentration	
co2	*ppm*	Value measured in	ppm
CO		Atmospheric Carbon Oxide concentration	
co	*ppm*	Value measured in	ppm
VOC		Volatile Organic Compounds concentration measured by a device	
voc	*ppm*	Value measured in	ppm
NH3		Atmospheric Ammonia concentration	
nh3	*ppm*	Value measured in	ppm
O3		Atmospheric Ozone concentration measured by a device	
o3	*ppm*	Value measured in	ppm
NO2		Atmospheric Nitrogen dioxide concentration	
no2	*ppm*	Value measured in	ppm
PM2.5		Particulate Matter in the air (size up to 2.5 micrometers)	
pm2_5	*microGramsPerCubicMeter*	Value measured in	μg/m3
PM10		Particulate Matter in the air (size up to 10 micrometers)	
pm10	*microGramsPerCubicMeter*	Value measured in	μg/m3
Water Pressure		Water Pressure measured in pipes by a device	
waterPressure	*bar*	Value measured in	bar
pH		Scale used to specify how acidic or basic a water-based solution is	
ph	*value*	Value between 0 and 14 measured in	pH
Occupation		Occupation percentage measured inside a vessel	
occupation	*percentage*	Value measured in	%
Soil Conductivity		Substances ability to conduct an electrical current in the soil	
soilConductivity	*microSiemensPerCentimeter*	Value measured in	μS/cm
Distance		Distance measured from the device to a surface	
	millimeters	Value measured in	mm
distance	*maxMillimeters*	Maximum distance the sensor can be to a given surface	mm
	minMillimeters	Minimum distance the sensor can be to a given surface	mm

Table 5.2 continued from previous page

Data Type *Property*	*Sub Property*	**Description**	**Unit**

Table 5.2: Measure Data Types

The current shared model schema can be found in ***TODO***.

Message Envelop Model

The message envelop model refers to how, coupled with the routing model in Section 5.3.2, information can easily transverse the system. The message envelop is used when a message is expected to flow though the system and is therefore used in all **Topic**s but the **Internal** one.

The diagram present in Figure 5.3 details this model.

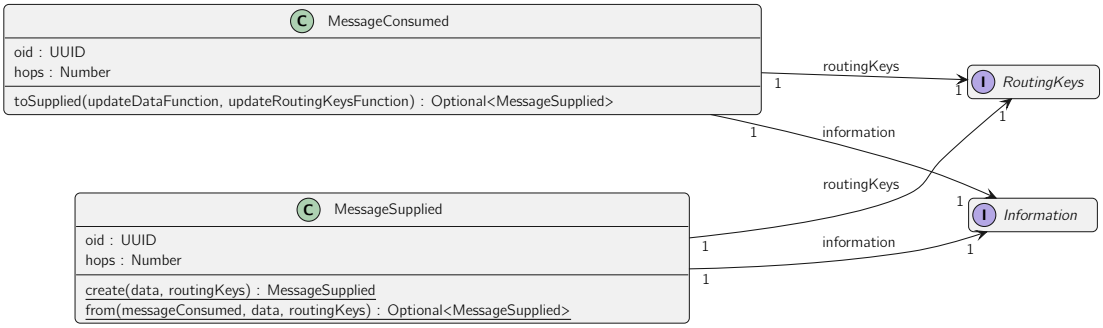

Figure 5.3: Message Envelop Model

As a brief description:

- A *MessageSupplied* is created in a container and supplied to start the flow of information in the system;
- A *MessageConsumed* is consumed by a container and can then be transformed into a *MessageSupplied* if needed;
- *Information* represents the content of the message;
- *RoutingKeys* represents the model referenced in Section 5.3.2;

This concept is mainly used to ensure that information flowing in the system is not reprocessed, by verifying the unique id - *oid*, and is drooped if it enters a routing loop by verifying that the *hops* have not reached a maximum value.

Routing Model

The routing model refers to how information can be routed through the system based on various parameters. The initial and current idea is based on the *pub/sub* pattern (**TODO**), containers subscribe to information in a **Topic** with specific *RoutingKey*s and publish information with *RoutingKey*s.

The diagram present in Figure 5.4 details this model.

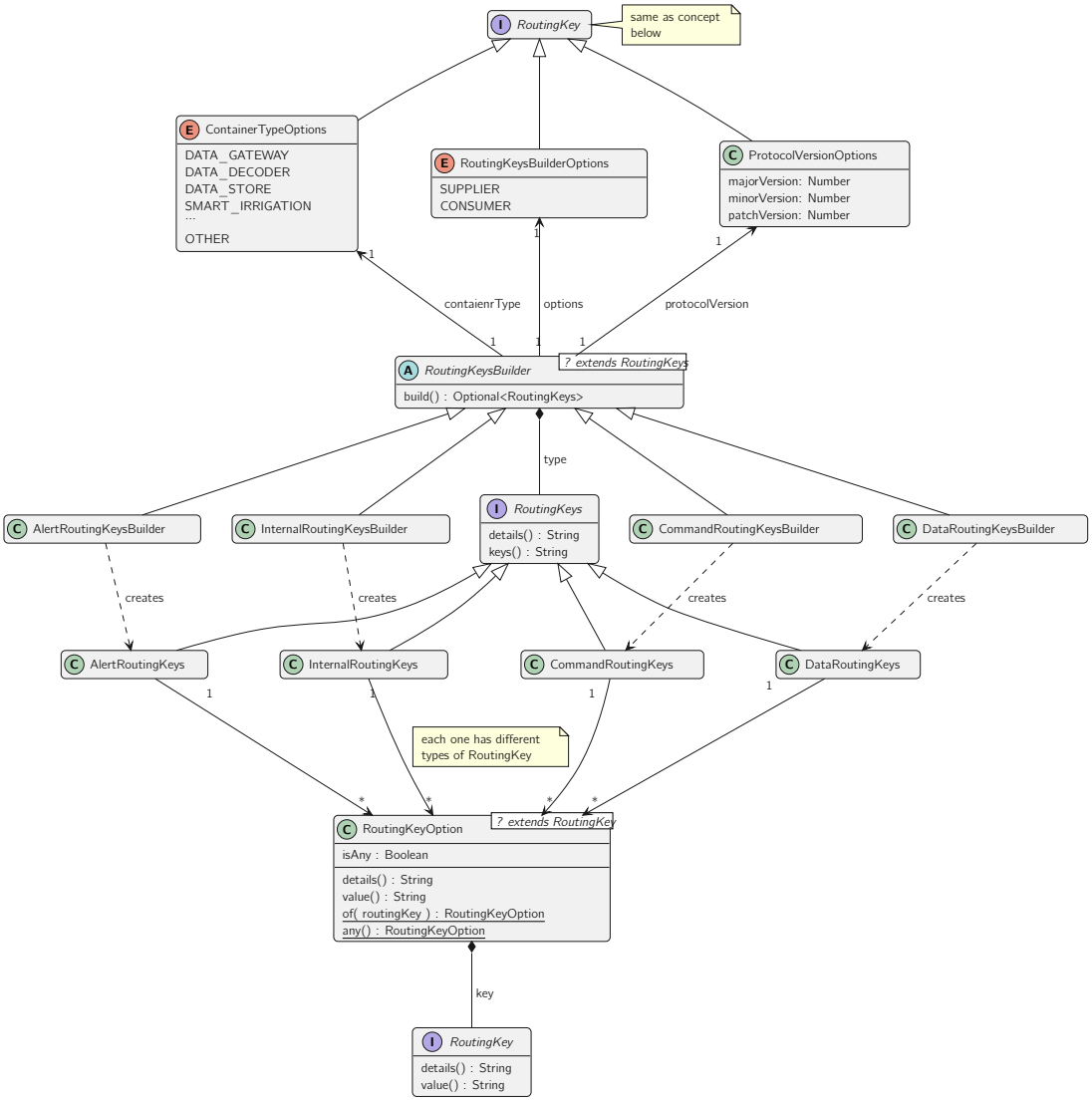

Figure 5.4: Routing Model

As a brief description:

- *RoutingKeys* is the concept referenced in Figure 5.3 and represents a collection of different *RoutingKeyOption*s;
- There are currently 4 types of *RoutingKeys*, one for each **Topic**;
- To ensure that the various containers in **Sensae Console** understand each other a *ProtocolVersionOptions* is provided. This concept follows the Semantic Versioning Specification 2.0 (Preston-Werner 2011) an is constructed according to the version of *iot-core* imported by the container;
- There are multiple *RoutingKey* types not displayed in the diagram for brevity.
- The *RoutingKeysBuilder* implements the *Builder* pattern and its single responsibility is to validate and create *RoutingKeys*;

- A *RoutingKeyOption* can have the value *any*, if the *RoutingKeysBuilderOptions* has the value *CONSUMER*. This provides a 'relaxed' mode, for containers that consume/-subscribe to messages and a 'strict' mode, where each *RoutingKey* must be specified, for containers that supply/publish messages.

In the Table 5.3 all currently used *RoutingKey* are presented.

Topic *Routing Key*	**Description**
Common	Routing Keys that belong to every Topic
Protocol Version Options	Version of the used *iot-core* package
Container Type Options	Type of the Container that published the message
Ownership Options	Does the message contains the **Domain**s that own it[1]
Topic Type Options	Topic used to publish the message
Internal	Routing Keys that belong to the Internal Topic
Operation Type Options	Intent of the message, e.g. unknown context found
Context Type Options	Type of content in the message, e.g. device information
Data	Routing Keys that belong to the Data Topic
Info Type Options	How data is shaped: (i) ENCODED, (ii) DECODED and (iii) PROCESSED
Device Type Options	Type of device, e.g. LGT-92 or EM300-TH
Channel Options	Name of channel where data flows, e.g. *smartIrrigation* or *default*
Data Legitimacy Options	Is the data legitimate: (i) UNKNOWN, (ii) CORRECT, (iii) INCORRECT and (iv) UNDETERMINED
Records Options	Does the data contains **Records/Metadata**[1]
Air Humidity Data Options	Does the data contains information about Air Humidity[12]
Air Pressure Data Options	Does the data contains information about Air Pressure[12]
Air Quality Data Options	Does the data contains information about Air Quality[12]
Battery Data Options	Does the data contains information about the device Battery[12]
CO2 Data Options	Does the data contains information about CO2 levels[12]
CO Data Options	Does the data contains information about CO levels[12]
Distance Data Options	Does the data contains information about distances to a surface[12]
GPS Data Options	Does the data contains information about the device GPS coordinates[12]
Illuminance Data Options	Does the data contains information about illuminance in the environment[12]
Motion Data Options	Does the data contains information about the device motion[12]
NH3 Data Options	Does the data contains information about NH3 levels[12]
NO2 Data Options	Does the data contains information about NO2 levels[12]
O3 Data Options	Does the data contains information about O3 levels[12]
Occupation Data Options	Does the data contains information about occupation levels[12]

Table 5.3 continued from previous page

Topic *Routing Key*	**Description**
pH Data Options	Does the data contains information about ph level[1][2]
PM2.5 Data Options	Does the data contains information about pm 2.5 concentration[1][2]
PM10 Data Options	Does the data contains information about pm 10 concentration[1][2]
Soil Conductivity Data Options	Does the data contains information about the soil conductivity[1][2]
Soil Moisture Data Options	Does the data contains information about the soil moisture[1][2]
Temperature Data Options	Does the data contains information about the temperature[1][2]
Trigger Data Options	Does the data contains information about something that works as a switch[1][2]
Velocity Data Options	Does the data contains information about the device velocity[1][2]
VOC Data Options	Does the data contains information about VOC concentration[1][2]
Water Pressure Data Options	Does the data contains information about water pressure[1][2]
Command	Routing Keys that belong to the Command Topic
Command Type Options	Type of command, e.g. Open Valve
Alert	Routing Keys that belong to the Alert Topic
Alert Category Options	Category of the alert published, e.g. Fire Detention
Alert Subcategory Options	Category of the alert published, e.g. Humidity With High Rate Of Change
Alert Severity Options	Severity of the alert published, from *Information* level to *Critical* level

Table 5.3: Routing Types

[1]has three possible values: (i) UNDETERMINED, (ii) WITH, (iii) WITHOUT
[2]related to the explored Data Types

Routing keys help to strengthen the boundaries that a container is expected to have. As an example, a Service in the **Service Scope** related to Waste Management would subscribe to the *Data Topic* with the following *Routing Keys*:

- *Info Type Options*: PROCESSED;
- *Channel Options*: 'wasteManagement';
- *Data Legitimacy Options*: CORRECT;
- *GPS Data Options*: WITH;
- *Occupation Data Options*: WITH;
- *Records Options*: WITH;
- *Ownership Options*: WITH;

And would, for example, subscribe to the *Alert Topic* with the following *Routing Keys*:

- *Alert Category Options*: 'wasteManagement';
- *Alert SubCategory Options*: 'trashAlmostFull';
- *Ownership Options*: WITH;

As expected, the structure and semantics of the information subscribed to are known upfront with the help of the package *iot-core*.

5.3.3 Bounded Contexts

The **Bounded Context** concept, defined by Evans 2014, refers to an unified model - with well-defined boundaries and internally consistent - that is a single piece in a larger system composed by various bounded contexts.

The **Sensae Console** is composed by the following bounded contexts:

- In **Configuration/Data Flow Scopes**:
  - Data Processor;
  - Data Decoder;
  - Device Management;
  - Identity Management;
  - Rule Management.
- In **Service Scope**:
  - Smart Irrigation;
  - Fleet Management.

Each of this contexts will be briefly addressed in the following sections.

Data Processor

The **Data Processor** context refers to simple data mappers that translate inbound information to **Data Unit**s, discussed in Section 5.3.2.

The received information must be *decoded*, meaning that the inbound information simply has a different structure than **Data Unit**.

The diagram in Figure 5.5 displays the noteworthy concepts in this context.

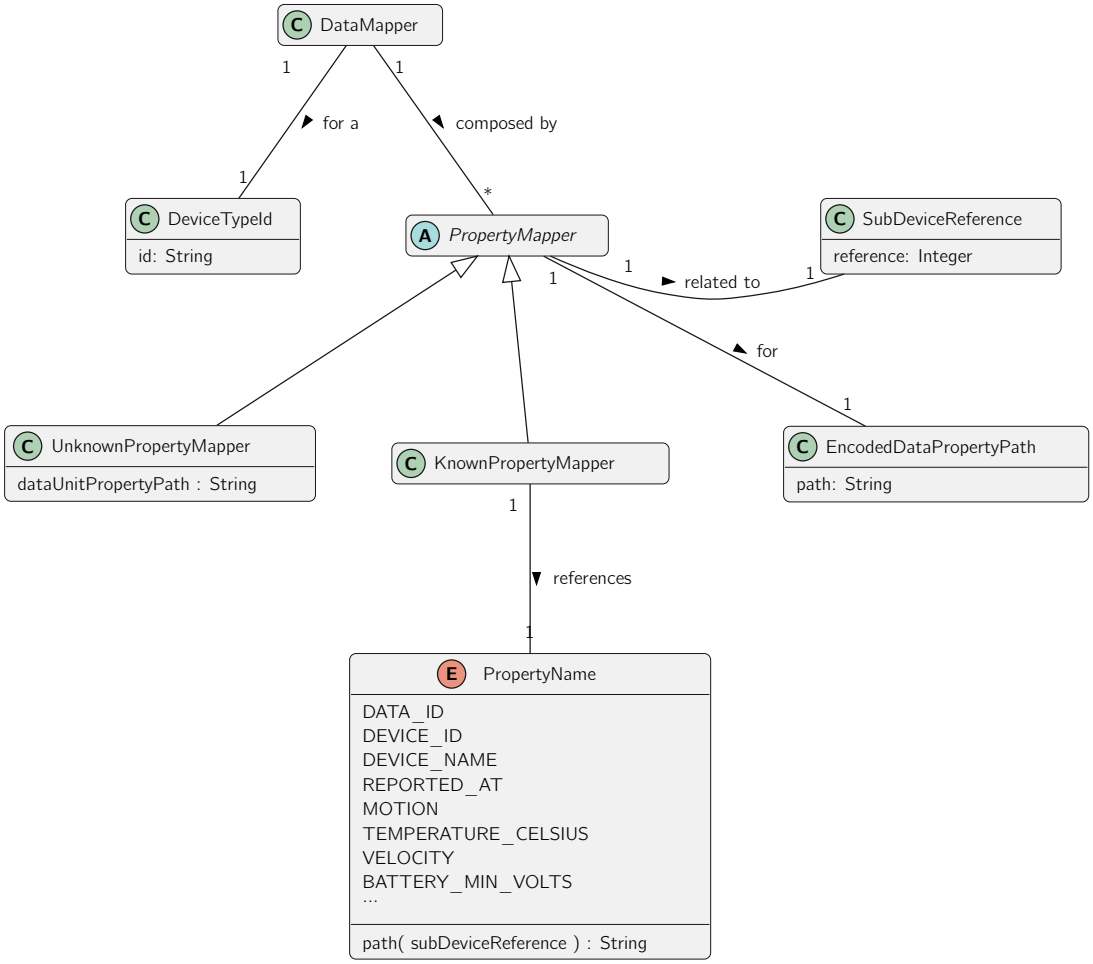

Figure 5.5: Data Processor Context Model

As a brief description:

- **DataMapper**, the root entity in this context is identified by a **DeviceTypeId** and has various instructions to map properties from the inbound information to a **Data Unit** properties;
- **DeviceTypeId** corresponds to the **Routing Key** *Device Type Options* mentioned in Table 5.3;
- **SubDeviceReference** represents a number that will be used later to reference a sub device when dealing with **Controller**s. For simple **Devices** the used and default value is *0*;

- **PropertyName** has much more properties that haven't been presented for brevity.

As an example, one could define an inbound information as a JSON document with the structure in the example 5.1.

To map the *temperature* value to the **TEMPERATURE_CELSIUS** property of a **Data Unit** the **EncodedDataPropertyPath** would be *decoded.data[0].temperature*.

```
{
    "uuid": "de1a9d15-c018-4547-8453-87111cb4f81b",
    "id": "d81e6e69-1955-48a1-a1dd-4c812c15ebac",
    "time": 1657646955748,
    "decoded": {
        "data": [
            {
                "temperature": 18,
            }
        ]
    }
}
```

Listing 5.1: Inbound Information Example

This process is simple since it expects the inbound information to be predisposed, but when working with IoT Devices, to optimize the bandwidth used, it is common to send information encoded. The following section presents an alternative to this process.

Data Decoder

The **Data Decoder** context refers to a more complex data mapper that translates inbound information to **Data Unit**s, discussed in Section 5.3.2. It was created to deal with the limitations mentioned in Section 5.3.3.

The received information is usually *encoded*, meaning that the inbound information is received as it was sent by the **Device**, commonly as a *Base64* encoded string that needs to be processed so that information can be extracted.

The diagram in Figure 5.6 displays the noteworthy concepts in this context.

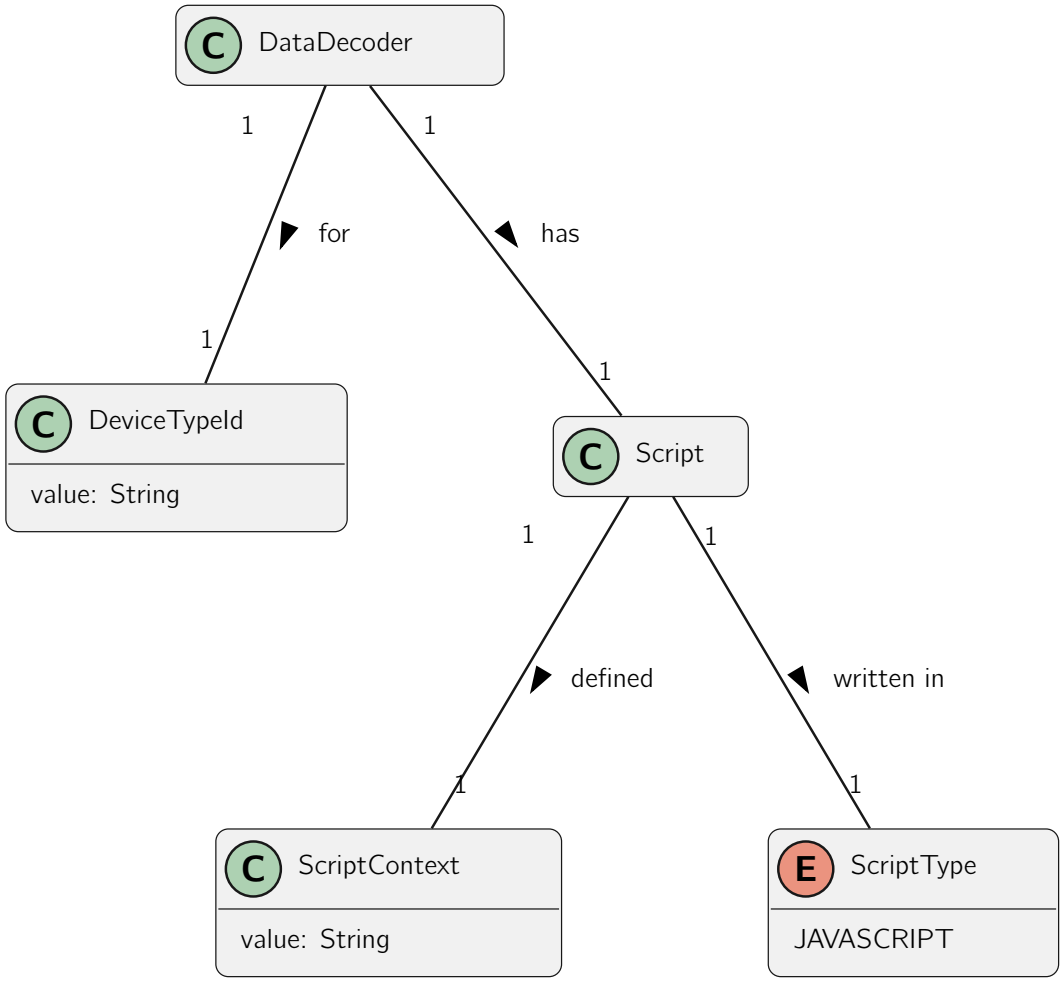

Figure 5.6: Data Decoder Context Model

As a brief description:

- **DataDecoder**, the root entity in this context is identified by a **DeviceTypeId** and has a **Script**;
- Currently a **Script** can only be written in *JavaScript* but in the future more languages like *Python* or *Groovy* can be added;
- The **ScriptContent** contains the code that will run for each inbound information that matches the **DeviceTypeId**.

This process requires some programing language knowledge but is much more flexible than the **Data Processor** operation.

Device Management

The **Device Management** context refers to the inventory of all registered **Device**s in the **Sensae Console**.

The diagram in Figure 5.7 displays the noteworthy concepts in this context.

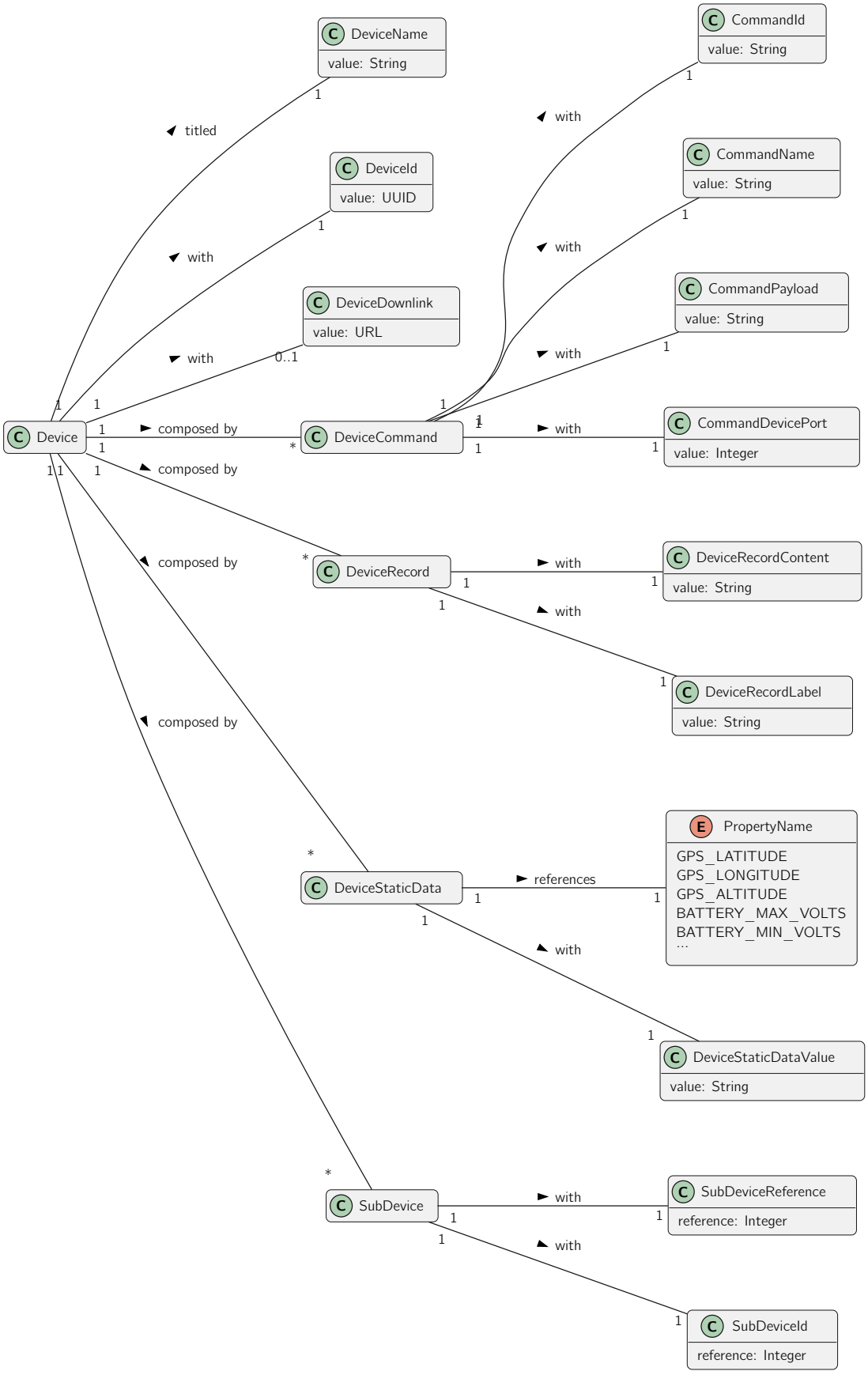

Figure 5.7: Device Management Context Model

As a brief description:

- A **Device** is uniquely identified by a **DeviceId**, has a **DeviceName** and may have a **DeviceDownlink**;
- A **DeviceCommand** defines how to send a **Downlink** with a specific action;
- A **DeviceStaticData** helps to define data such as the device location;
- A **DeviceRecord** enriches the device information with anything deemed important. This can also help to group devices by projects, type of utility and others;
- A **SubDevice** references another **Device** by its **DeviceId**. This, coupled with the concepts **SubDeviceMeasures** and **SubDeviceCommands** presented in Figure 5.2 help to split a **Controller**'s **Data Unit** into various **Data Unit**, one for each referenced **SubDevice**.

Identity Management

The **Identity Management** is concerned with identifying **Tenant**s, defining their permissions and what **Device**s they own. To simplify this a forth concept is introduced: **Domain**.

The diagram in Figure 5.8 displays the noteworthy concepts in this context.

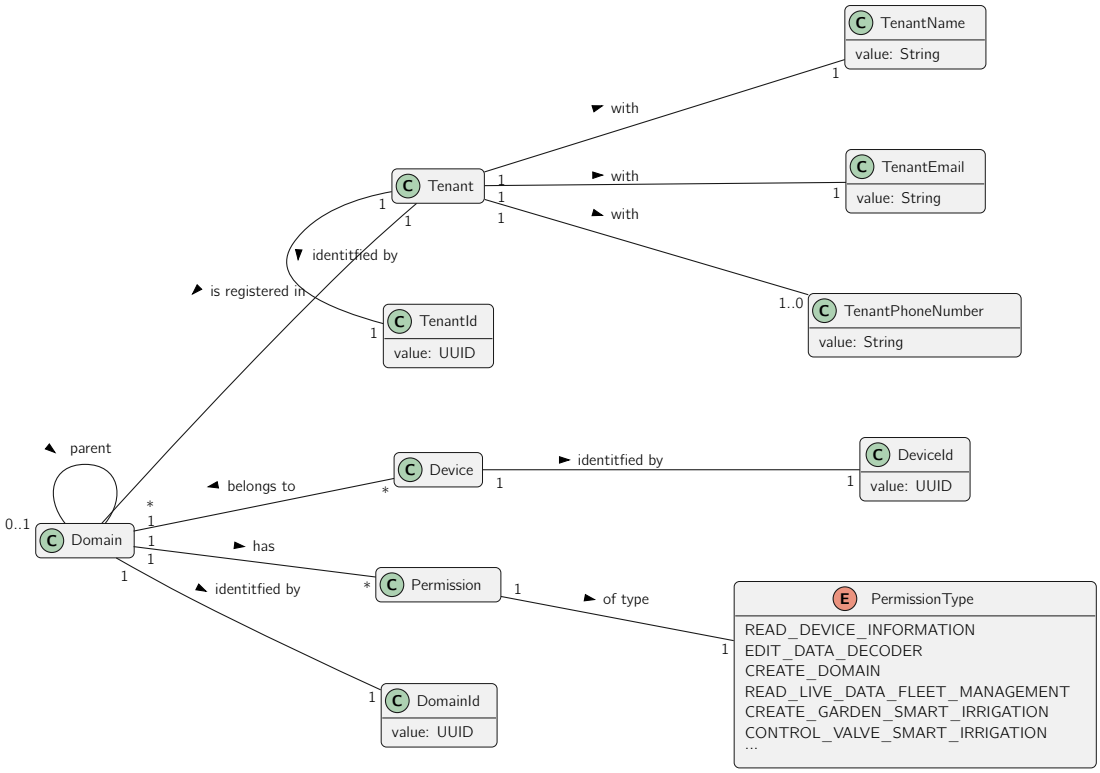

Figure 5.8: Identity Management Context Model

As a brief description:

- A **Domain** is uniquely identified by a **DomainId** and can have a parent **Domain**;

- There's a root **Domain**, the only one doesn't have a parent and has all available permissions;
- A **Tenant** has a **TenantName** and **TenantEmail**, unique **TenantId** and can have a **TenantPhoneNumber**;
- A **Device** is uniquely identified by a **DeviceId**;
- The **PermissionType** has much more types that haven't been presented for brevity.

A **Domain** represents a department in a hierarchical organization. An organization is composed by several domains in a tree like structure as presented in Figure 5.9.

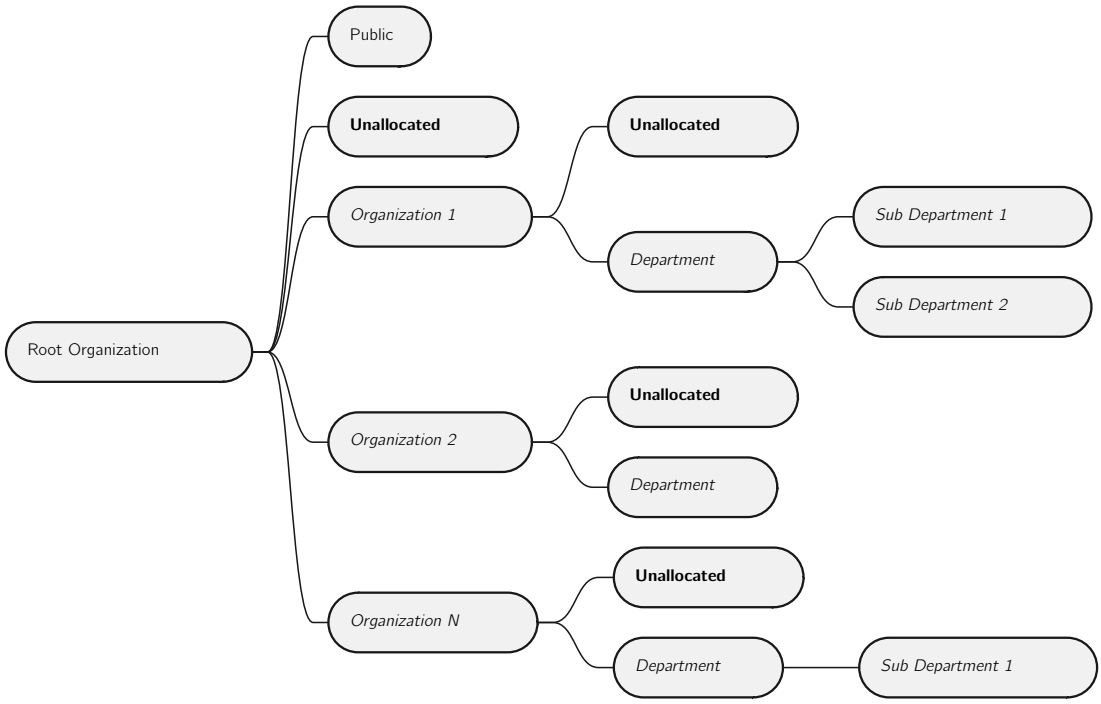

Figure 5.9: Domain Structure

Coupled with the figure above, there are other constrains:

- A domain owns all devices in it and in his subdomains;
- A domain can only inherit his parent domain permissions;
- A tenant has all the domain permissions that he is registered in;
- A tenant can only see the devices that the domains he is registered in has access to;
- All *Unallocated* domains have no permissions or devices and contain only tenants that are waiting to be assigned to a department or organization;
- The *Public* domain can be accessed by any tenant, including those who are not authenticated in the system;

Rule Management

The **Rule Management** context refers to rule scenarios that produce **Alert**s based on incoming **Data Unit**s.

The diagram in Figure 5.10 displays the noteworthy concepts in this context.

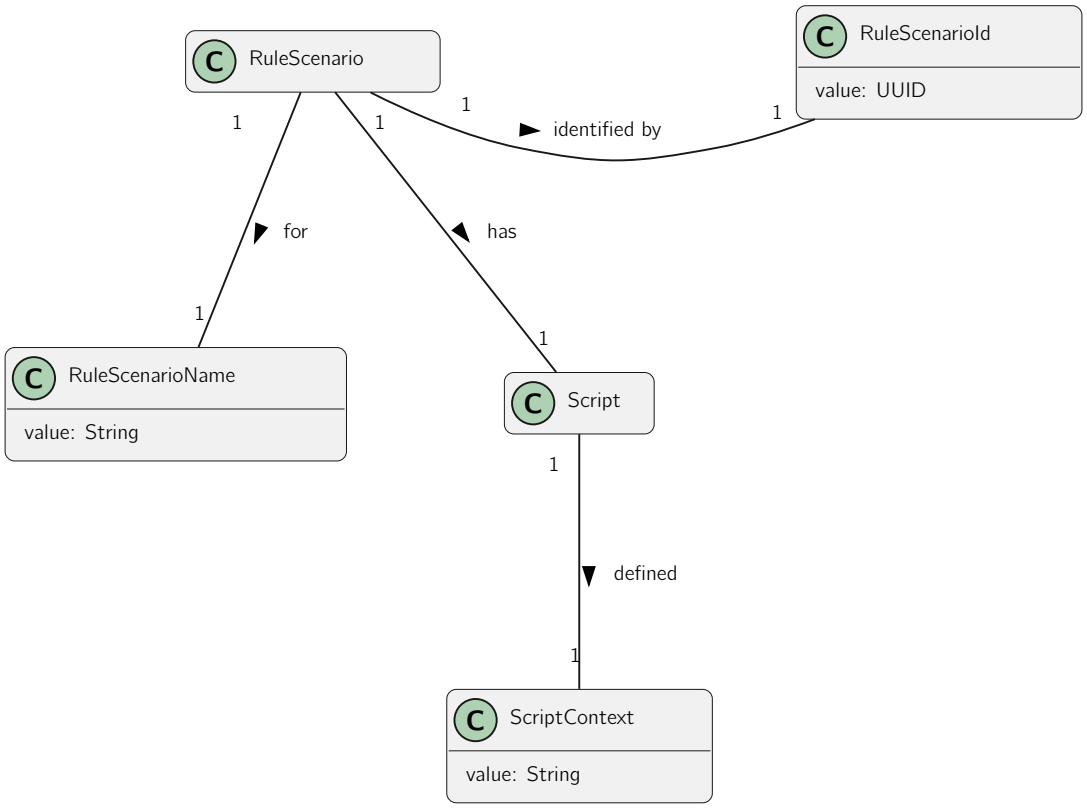

Figure 5.10: Rule Management Context Model

Notification Management

The **Notification Management** context refers to notifications and how/what types an addressee wants to receive.

The diagram in Figure 5.11 displays the noteworthy concepts in this context.

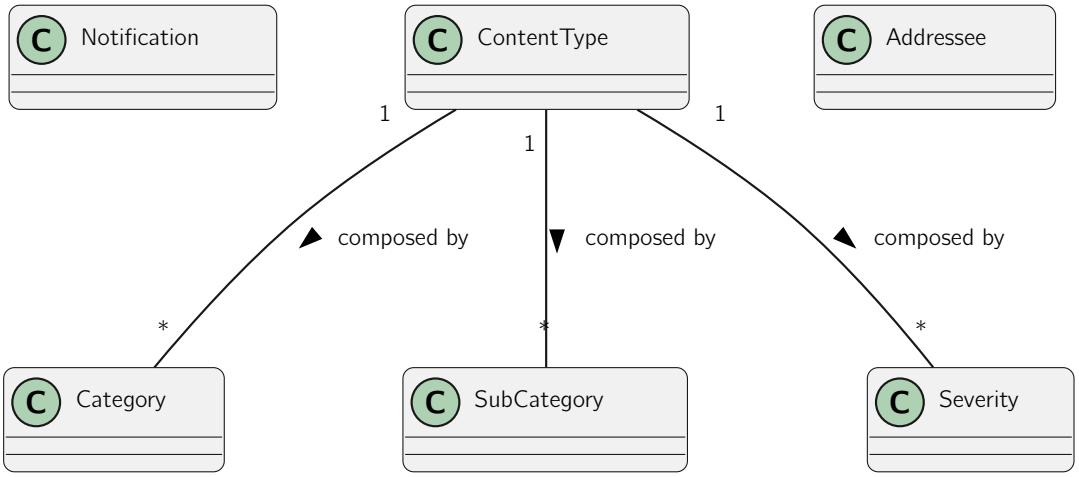

Figure 5.11: Notification Management Context Model

Smart Irrigation

Fleet Management

5.3.4 Synopsis

5.4 Architectural Design

In order to describe the system in detail at the architectural level, an approach based on the combination of two models, C4 (Brown 2018b) and 4+1 will be followed.

The 4+1 View Model (By and Jiang 1995), proposes the description of the system through complementary views thus allowing to separately analyze the requirements of various software stakeholders, such as users, system administrators, project managers, architects, and programmers.

The five views are thus defined as follows:

- **Logical view**: relative to the aspects of the software aimed at responding to business challenges;
- **Process view**: relative to the process flow or interactions within the system;
- **Development view**: relative to the organization of the software in its development environment;
- **Physical view**: relative to the mapping of the various components of the software in hardware, i.e. where the software is executed;
- **Scenario view**: related to the association of business processes with actors capable of triggering them.

The C4 Model (Brown 2018b, Brown 2018a) advocates describing software through four levels of abstraction: (i) system, (ii) container, (iii) component, (iv) code. Each level adopts a finer granularity than the level that precedes it, thus giving access to more details of a smaller portion of the system. These levels can be likened to maps, e.g. the system view corresponds to the globe, the container corresponds to the map of each continent, the

component view corresponds to the map of each of each country, and the code view to the map of roads and neighborhoods in each city.

Different levels allow you to tell different stories to different audiences.

The levels are defined as follows:

- **Level 1**: Description (context) of the system as a whole;
- **Level 2**: Description of system containers;
- **Level 3**: Description of components of the containers;
- **Level 4**: Description of the code or smaller parts of the components.

These two models can be said to expand along distinct axes, with the C4 Model presenting the system with different levels of detail and the 4+1 View Model presents the system from different perspectives. By combining the two models it becomes possible to represent the system from several perspectives, each with various levels of detail. To visually model/represent the ideas designed and alternatives considered, the UML was used.

In the following sections only combinations of perspectives and level deemed relevant for the design of the solution are presented.

The C4 level 4, code, will not be exhibited.

5.4.1 C4 Level 1 - Context

The context level aims at introducing the system as a whole. The external systems and users that communicate/interact with the system, **Sensae Console**, are demonstrated. Throughout this section the relevant C4 views of level 1 (context level) are presented.

Context Level - Logical View

The logical view of the system is introduced here, complete but not detailed, in order to answer the use cases and requirements discussed in ***TODO***. This takes into account the interactions of the platform with external systems and its interaction with the various actors of the system (Figure 5.12).

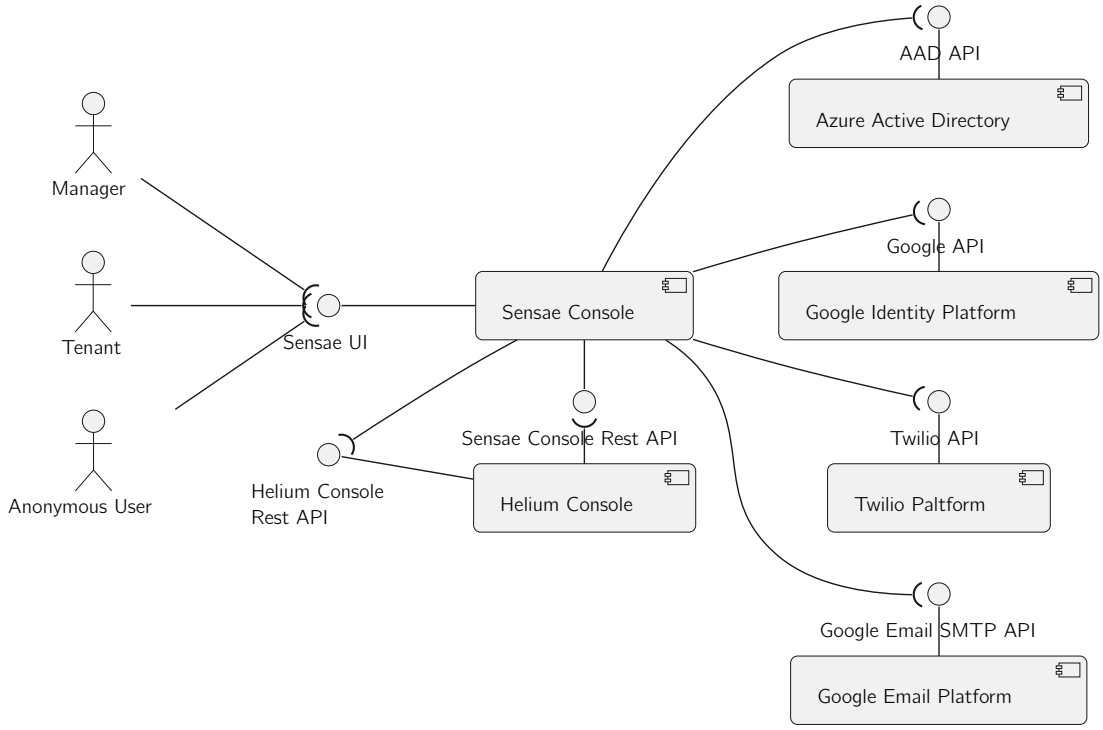

Figure 5.12: Context Level - Logical View Diagram

The external systems and its functions are as follows:

- **Helium Console**: Device data hub;
- **Azure Active Directory**: User authentication/identity;
- **Google Identity Platform**: User authentication/identity;
- **Twilio Platform**: SMS delivery;
- **Google Email Platform**: Email delivery.

The reason behind the use of external authentication/identity services is described in the Section 5.5.2.

Context Level - Physical View

Next is the physical view (Figure 5.13), intended to familiarize the reader with the idealized production environment.

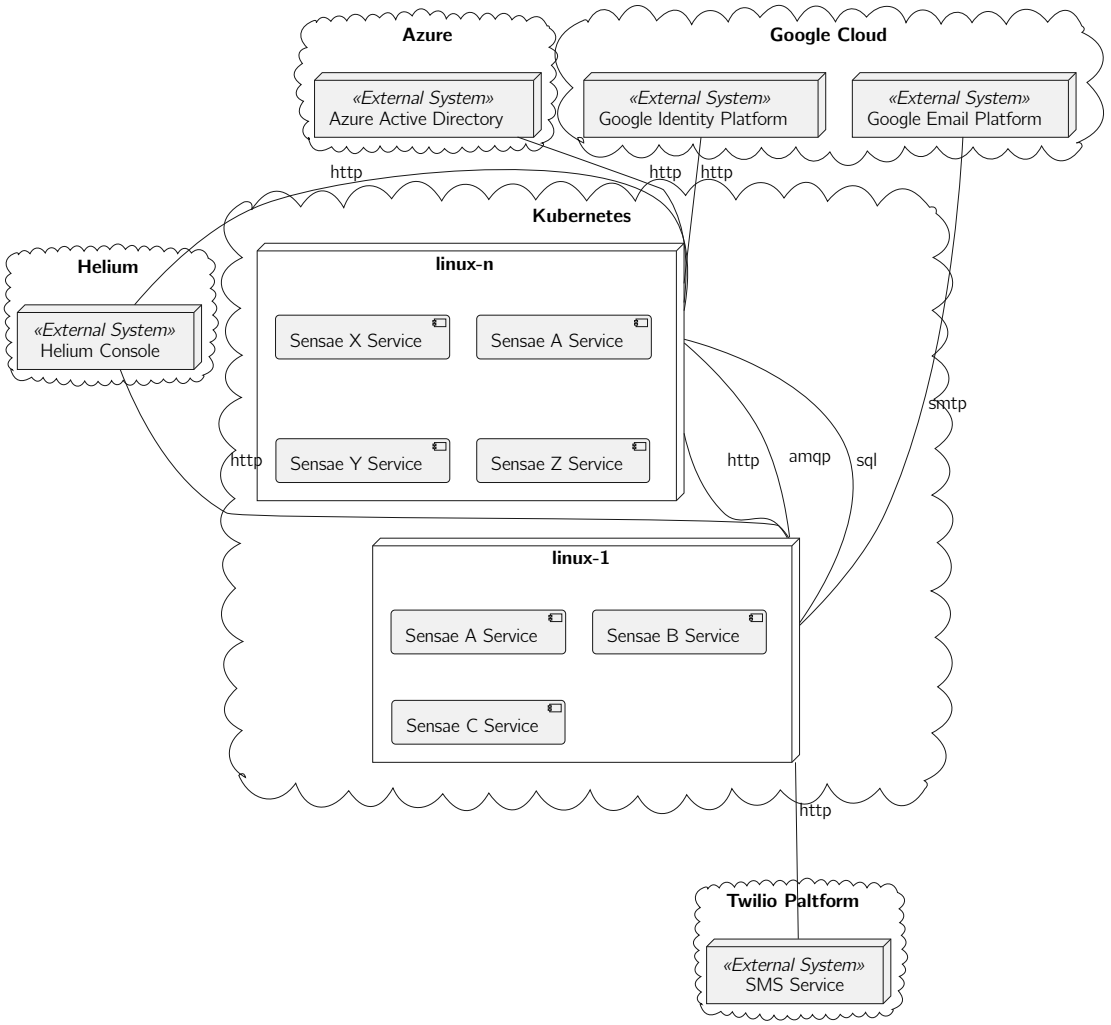

Figure 5.13: Context Level - Physical View Diagram

Due to time constrains the environment was not deployed to *Kubernetes* and the solution is instead orchestrated using *Docker Compose* in a single node/server.

Context Level - Development View

Next is the development view (Figure 5.14), intended to familiarize the reader with how the software is organized.

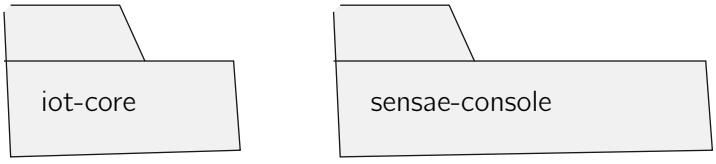

Figure 5.14: Context Level - Development View Diagram

The package *iot-core* contains the shared model discussed in the Section 5.3.2, and functions to define what type of device data/internal state a backend containers wants to subscribe or publish (discussed in Section 5.3.1).

The package *sensae-console* contains software of the various containers needed to run the **Sensae Console**. As expected *iot-core* is a core dependency for the *sensae-console* backend containers.

Context Level - Synopsis

The process view was not represented since at this level the interactions between the system, actors and external systems, are too abstract to be relevant for the reader.

5.4.2 C4 Level 2 - Containers

5.4.3 C4 Level 3 - Components

5.5 Architectural Alternatives Discussed

5.5.1 Data Streaming/Pipeline

5.5.2 User Authorization/Authentication

5.5.3 Internal Communication

5.6 Synopsis

Chapter 6

Implementation

Chapter 7

Evaluation of the Solution

Chapter 8

Conclusion

8.1 Achievements

8.2 Unfulfilled Results

8.3 Future Work

8.4 Synopsis

Bibliography

Brown, Simon (June 2018a). *The C4 Model for Software Architecture*. [Online; accessed 30. Jun. 2022]. url: https://www.infoq.com/articles/C4-architecture-model/.

– (2018b). *The C4 model for visualising software architecture*. [Online; accessed 30. Jun. 2022]. url: https://c4model.com.

By, Slides and Jack ZhenMing Jiang (Nov. 1995). "Architectural Blueprints–The "4+ 1" View Model of Software Architecture". In: [Online; accessed 30. Jun. 2022].

Evans, E. (2014). *Domain-Driven Design Reference: Definitions and Pattern Summaries*. Dog Ear Publishing. isbn: 9781457501197. url: https://books.google.pt/books?id=ccRsBgAAQBAJ.

Preston-Werner, Tom (June 2011). *Semantic Versioning 2.0.0*. [Online; accessed 30. Jun. 2022]. url: https://semver.org/.

Appendix A

Appendix Title Here

Write your Appendix content here.