BIG IoT – Bridging the Interoperability Gap of the Internet of Things

Deliverable 4.3.a:

Service Discovery and Orchestration

– first release

Version 1.0

Date: 31.12.2016

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688038.

Responsible Person and Affiliation	Yong Wang (TU Clausthal)
Contributor (s):	Darko Anicic (Siemens), Aparna Saisree Thuluva (Siemens), Achille Zappa (NUIG), Hoan Quoc (NUIG), Yong Wang (TU Clausthal), Arne Bröring (Siemens).
Reviewer(s):	Costas Pipilas (ECONAIS) Luca Gioppo (CSI)
Due Date / Delivery Date	31.12.2016
State	Final
Version	1.0
Confidentiality	Public

Version Control

Version	Description of Changes	Date of Resolution
0.1	Initial Version	27.11.2016
0.11	Partners Contributions	01.12.2016
0.2	Preview Version	02.12.2016
0.21	Technical & Quality	11.12.2016
0.3	Final Draft	21.12.2016
1.0	Final	23.12.2016

Executive Summary

This document analyses the assessed high-level requirements for automated discovery of Offerings towards developing a solution for automated Offering discovery. We develop this solution based on the BIG IoT High-Level Architecture as designed in Task 2.4 and the BIG IoT Marketplace developed in Task 4.1. The document further elaborates the concept of semantic orchestration of Offerings based on so called *Recipes*. This work is conducted within Task 4.3 and is based on results achieved in BIG IoT Task 3.2 and Task 4.2.

Table of Contents

European Commission
Horizon 2020
European Union funding
for Research & Innovation

List of Figures

Terms and Definitions

Terms	Definitions
BIG IoT API	A set of specifications for - Providers and Consumers to interact with the BIG IoT Marketplace to authenticate, register, discover and subscribe to Offerings; and perform accounting - Consumers to directly access the Resources offered by a Provider The BIG IoT API defines the supported communication protocols, data formats, semantic descriptions, etc. In order to facilitate BIG IoT Applications, Services and Platforms to implement and use the BIG IoT API, dedicated Provider and Consumer Libs (SDKs) are provided for various platforms and programming languages, offering also programming interfaces to developers.
BIG IoT Application **(or short Application)**	An application software that uses the BIG IoT API to discover Offerings on the BIG IoT Marketplace, subscribe to Offerings and access the offered Resources. A BIG IoT Application acts merely as an Offering Consumer.
BIG IoT Application / Service / Platform Developer (or short BIG IoT	A software developer that implements or integrates a BIG IoT Service, Application or Platform.

Developer)	
BIG IoT Application / Service / Platform / Marketplace Provider or Operator	The organization that operates a BIG IoT Application, Service, Platform, Marketplace instance. It is hereby not relevant if a particular instance is hosted on a provider organization's own infrastructure or a 3rd party infrastructure.
BIG IoT Core Developer	A software developer that implements or extends BIG IoT Marketplace and/or BIG IoT Lib components.
BIG IoT enabled Platform (or short BIG IoT Platform or just Platform)	An IoT Platform (or Smart Object Platform) that implements and uses the BIG IoT API to register Offerings on the BIG IoT Marketplace and provide access to the offered Resources. A BIG IoT Platform acts merely as an Offering Provider.
BIG IoT Marketplace	The BIG IoT Marketplace allows Providers to register their Offerings (based on semantic descriptions) and Consumers to discover relevant Offerings (based on semantic queries) at run-time. It also provides accounting support for Consumers and Providers to track the amount of resources accessed, as well as a web portal for developers and administrators to support the implementation and management of their Applications, Services, and Platforms
BIG IoT Service (or short Service)	A BIG IoT Service implements and uses the BIG IoT API to consume and/or provide Offerings via the BIG IoT Marketplace. A BIG IoT Service can act both as an Offering Consumer and Provider. It typically consumes basic Information or Function in order to offer "higher-value" Information or Functions on the BIG IoT Marketplace.
BIG IoT User	A user of a BIG IoT Application. A BIG IoT User is typically an employee of an Enterprise, SME or Organization (e.g. City Authority), but not limited to that.

Endpoint	An Endpoint in the context of BIG IoT is a web based interface for consumers to access Offerings via a Provider. An Endpoint description consists of properties like Endpointtype and URI.
Function	Functionality that can be invoked by Consumers and is provided by • - a task on an actuator (as part of an IoT Platform) • - a Service that provides some computational functions or higher level functionality delegating to one or more lower level Functions
Information	Data provided to Consumers by • - a sensor (as part of an IoT Platform) • - a Service that takes one or more Information sources and combines them to provide some added value
IoT Service (or short Service)	Software component enabling interaction with resources through a well-defined interface in order to access or manipulate information or to control entities. An IoT Service can be orchestrated together with non-IoT services (e.g., enterprise services). Interaction with the service is done via the network. (based on [IoT-A])
IoT Platform (= Smart Object Platform)	A computing and communication system that hosts software components enabling interaction with Smart Objects in order to access or manipulate information or to control them. An IoT Platform may be implemented on a backend or cloud infrastructure, or directly on a Smart Object. Interaction with the platform is done via the network.
Offering	BIG IoT enables Providers to offer or trade access to

	Information and Functions with Consumers via the Marketplace. An Offering is defined by an Offering description, which describes a set of Resources offered on the Marketplace. It typically encompasses a set of related Information or Functions. An Offering description provides a semantic description of the Resource(s) provided to a Consumer once the Offering is accessed. The description also entails context and meta information about the Offering, including information like the Region (e.g., a city or spatial extent) where the Resource(s) relate to, the Price for accessing the Resource(s), the License of the Information provided, input & output data fields, etc.
Offering Consumer (or short Consumer)	A BIG IoT Application or Service that is interested to discover and access IoT resources in order to provide a new service or function. A Consumer discovers and subscribes to relevant Offerings via the BIG IoT Marketplace, and accesses the offered resources via the BIG IoT API.
Offering Provider (or short Provider)	A BIG IoT Platform or Service that wants to offer or trade IoT resources via the BIG IoT Marketplace. A Provider registers its Offering(s) on the BIG IoT Marketplace, and provides access to the offered resources via the BIG IoT API.
Offering Query (or short Query)	Consumers are able to discover offerings of interest on the marketplace by providing an (Offering) Query. A Query describes the properties of Offerings a client is interested in (Offering type, input & output data fields, Price, License, etc.)
Physical Entity	Any physical object that is relevant from a user or application perspective. [IoT-A]
Smart Object (= Thing)	A Device able to compute and communicate information about itself or related artifacts (Physical Entities) to other devices or computer applications; a Smart Object is typically attached to or embedded inside a Physical Entity. Smart Objects either monitor a Physical Entity (sensing) or interact with the physical world through actuators (actuation). Those functions can be either controlled autonomously by

	local computations or triggered from remote.
Subscription	Agreement to access the Resource(s) of a single Offering. This comprises: • - A Consumer's willingness to access the Offering • (he checked License, service level, rating, description, etc.) • - The Consumer's consent to pay for the access to the Resources (according to the specified Price), if applicable.
BIG IoT Semantic Core Model	BIG IoT Semantic Core Model specifies all important domain concepts in BIG IoT project including all basic conceptual entities and their relationships. This semantic model is used as basis for (1) the Offering Description to define the capabilities of offerings provided by IoT platforms or services, and (2) the underlying data model of the BIG IoT Marketplace.
BIG IoT Semantic Application Domain Model	BIG IoT Semantic Application Domain Model defines an application-specific vocabulary that is built on both, BIG IoT Semantic Core Model and Mobility Domain Model, and can be used for annotating Offering Descriptions.
Mobility Domain Model	Mobility Domain Model defines a common terms used in the mobility domain. The model aims to improve information exchange and data interoperability between mobility systems, in particular in Internet of Things applications. This model represents an extended schema.org vocabulary for mobility (mobility.schema.org).
Semantic Offerings Composition Recipe	Semantic Offerings Composition Recipe or shorter Recipe provides description of the composition of

(or BIG IoT Recipe)	offerings. It is a specification of requirements of an added value service, and hence it represents a template that can be fulfilled by multiple offerings.
Semantic Recipe Model	Semantic Recipe Model defines a model for BIG IoT Recipes. A Recipe is a specification of requirements of an added value service, and hence it represents a template that can be fulfilled by multiple offerings. All terms and their relations, required for specifying Recipes, are defined in Semantic Recipe Model.

1 Introduction

The goal of the BIG IoT project is enabling the emergence of cross-platform, cross-standard and cross-domain IoT services and applications towards building IoT ecosystems. To achieve this goal, the BIG IoT consortium defines a shared interface, the so-called BIG IoT API comprising functionalities such as identity and management of offerings from IoT platforms and services, discovery of offerings, access to offerings, etc. A developer user can set up a BIG IoT Application utilizing the BIG IoT API to access offerings that are provided by IoT platforms or even composed of existing offerings from multiple IoT platforms.

One of the challenges of designing the BIG IoT API is finding offerings, namely, automatic detecting wanted offerings offered by different IoT platforms. Offerings from different IoT platforms have typically different syntax, while they can be semantically equivalent. Thus, in order to detect such offerings, on the one hand, they need to be described semantically and offering descriptions need to be stored in a database, on the other hand, the consumer needs to describe his requirements of wanted offerings. This document introduces methods for the description of desired combinations of offerings as well as a mechanism for the discovery to be used in the composition of offerings.

In the most IoT platforms, offerings are typically delivered point-to-point. However, the BIG IoT ecosystem environment creates the opportunity for providing added-value, integrated offerings, which are delivered by composing existing offerings. The automated and dynamic composition (orchestration) of offerings facilitates the creation of innovative offerings that suit the consumer’s needs, with a reduced time-to-market and better component reuse. A key element for the dynamic composition of offerings is the semantics for offering composition, since the envisaged mechanism should not only support the description of the syntax of offerings, but also needs to enable the representation of the meaning of the offering’s capabilities and parameters. Moreover, non-functional properties of offerings need to be considered, since it is important for discovery for offerings. The most relevant and most suitable service offering compositions, the so-called Recipe, describes semantically the request and needs of the new offering, the request of existing offerings for the new offering, and workflow for coordinating the existing offerings. This plays a decisive role for orchestration of offerings. In this document, we focus on the concept for offering composition description, which will be used to implement the offering composition.

1.1 Relation to other Deliverables

The tasks of WP3 and WP4 are highly interrelated, however, their deliverables have each their own, clear responsibilities. The figure below follows the general BIG IoT architecture description in D2.4 and illustrates at hand of this architecture the scope of each of the different deliverables of the tasks 3.1, 3.2, 4.1, 4.2, and 4.3.

D3.1 covers the description of the BIG IoT API. This entails a specification of the Web API (e.g., interaction model, and encodings) as well as description of the programmatic API that is implemented as part of consumer and provider lib of the SDK.

D4.1 describes the architecture of the BIG IoT Marketplace. This includes the general design, the workflows of interactions, the GraphQL-based API of the marketplace, as well as the user interface of the portal to utilize the marketplace.

D3.2 describes the model and schema of the core semantic model of BIG IoT. This semantic model is used as a basis for (1) the Offering Description to define the capabilities of offerings provided by IoT platforms or services, and (2) the underlying domain model of the BIG IoT Marketplace. Also contained in this deliverable is a description on how to map between the GraphQL API of the marketplace frontend and its SPARQL based triple store backend.

D4.2 builds up on the core semantic model of D3.2 and defines the application domain model, which specifies a vocabulary of terms that can be used in Offering

Descriptions, and in the triple store of the marketplace. This model concentrates on the mobility domain.

D4.3 addresses the orchestration and composition of the offerings of BIG IoT providers (i.e., platforms or services). A composition of offerings can be specified by defining a Recipe. The semantic model and examples of such recipes is the scope of this deliverable.

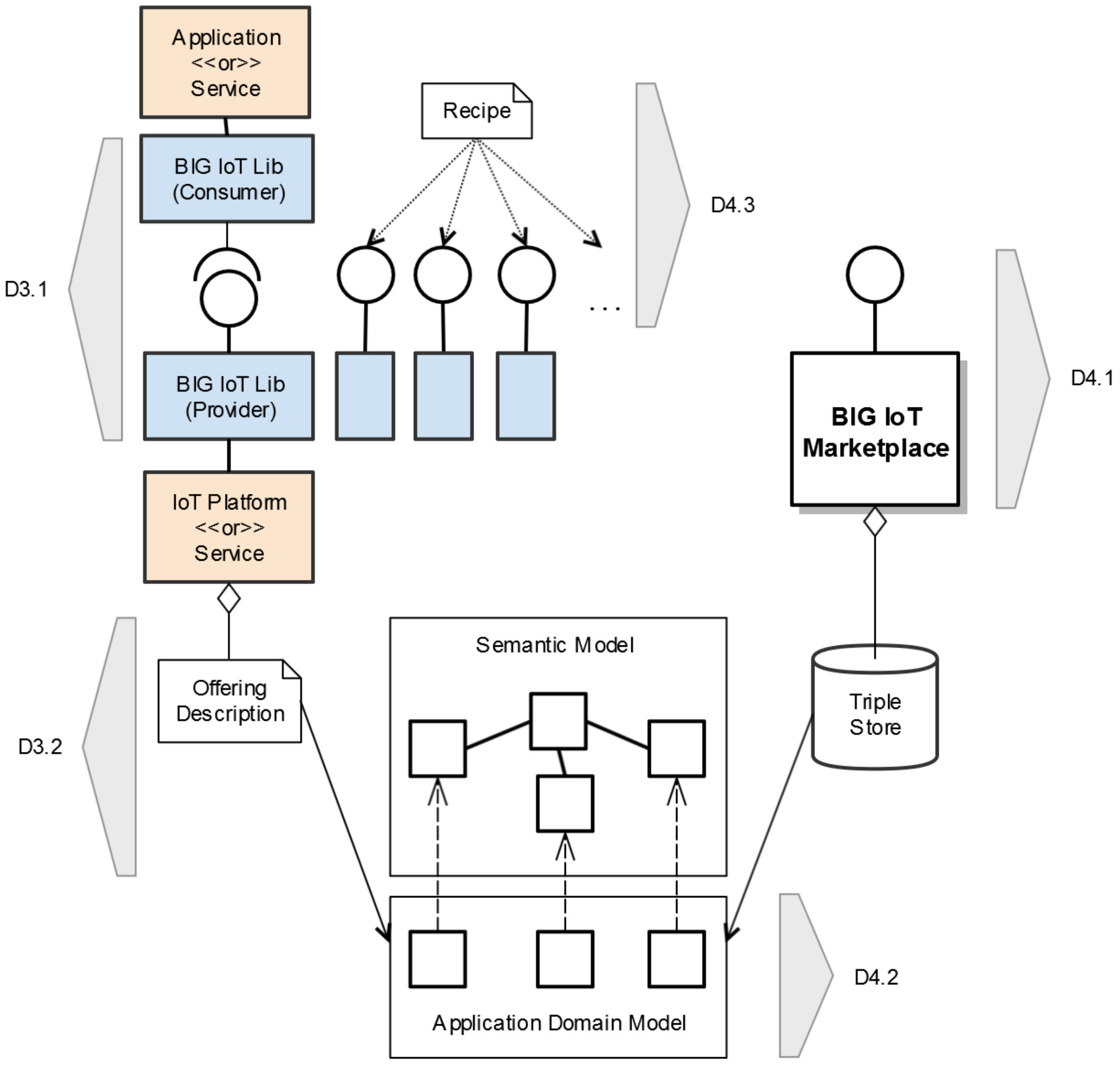

Figure 1, Relation of deliverables.

1.2 Structure of this Document

The rest of this document is structured as follows. Chapter 2 introduces related works in the BIG IoT project especially and related frameworks, technology for offering composition. Chapter 3 shows a mechanism for offering discovery by using SPARQL. Chapter 4 describes an approach on how to describe the composition of offerings, a template based semantic description. Finally, Chapter 4 provides the conclusions of this work and gives an outlook for the next steps.

2 Background and Related Work

This section first describes related tasks in the BIG IoT project including the tentative results of the architecture design, which closely relates to this task, and then describes work related to our offering composition concept and points out the many benefits for our approach.

2.1 Related Work within BIG IoT

In this document, we reuse terms and definitions that are part of the common shared glossary of the BIG IoT project, in other hand, current concepts and results of other tasks have been considered by designing the work developed in this task in order to maintain a consolidated terminology for the entire BIG IoT project. Therefore, in this section, we briefly introduce the related work from the overall BIG IoT system to put our work in context.

High-level architecture

This is defined in D2.4 and describes the basic components and their interactions within the overall system. It is a guideline for designing our concept of offering composition, and especially for implementation and integration.

BIG IoT Marketplace

The design of the marketplace is described in D4.1 and it is a composite system consisting of multiple sub-components. One important function in the marketplace is to allow registration and discovery of offerings using semantic technologies. Through this document, we contribute concepts for offering discovery and offering composition to the overall marketplace concept.

Offering Description

A goal of offering orchestration is to create new offerings based on existing ones. A new offering must conform to the format of Offering Description which is defined by Task 3.2 and in D3.2, in order to be registered and offered through the BIG IoT Marketplace.

2.2 Related Work for Offering Orchestration

In this section, we consider three aspects of related works for orchestration, technologies for description of offerings, methods to build requirements for the new offering, and frameworks which enable the composition of a service.

2.2.1 Web Service Description

In BIG IoT among the functional properties of an offering, we can see common concepts such as endpoint, input data, output data and properties coming from the Web Service domain, so that the design of offering orchestration description could get the benefits from a survey of Web Service description technologies. This section introduces, briefly, the related technologies. The more details about those technologies can be found in D2.1 [1].

Web service description has been a topic of great interest for many years, and therefore a large amount of work has been done in the area such as the Web Services Description Language (WSDL) [2], Universal Description, Discovery and Integration (UDDI) [3], Web Services Business Process Execution Language (WS-BPEL) [4], Web Service Modeling Ontology (WSMO) [5], and OWLS [6]. The Web Service description technologies should support usually two main functionalities; on the one hand, it should enable the ability to search existing services that are required by the request of other services. On the other hand, it should support the coordination and management of services so that to be able to compose them to create a new service.

There are two different concepts for service composition: syntactic based service descriptions and semantic based descriptions. For the syntactic based service description, the services use a common protocol, programming interface, etc. Exact information, such an ID or a name has been used for example in the case of UUDI and BPEL. For the semantic based method, the capabilities and properties of services as well as composited services are described with semantic contexts. Discovering an existing service which is desired by a composited service is supported by the semantic description of services. OWL-S and WSMo are well-known ontologies that can be used to describe those services.

2.2.2 Template based composition

The goal of an offering composition is to create a new offering based on a set of available existing offerings, which fulfils all specified requirements for the new offering. Such a new offering consists of a set of offerings as well as connections (interactions) between these offerings. The task of the offering discovery is therefore first to select suitable offerings from the set of all available offerings in order to create the new offering. The specification of conditions of desired existing offerings could be described as a template. The criteria for selecting suitable existing offerings for the new offering are defined by using such templates. A template method is used in a number of approaches, see for example [7].

2.2.3 Related Framework

IFTTT [8] is a service platform where users may create new services or use existing ones. It enables users to create chains of simple conditional statements, called "recipes", which are exposed as web services. They are triggered based on changes to other web services such as Gmail, Facebook, Instagram, and Dropbox and so on. Currently 287 service providers from different domains are available via IFTTT, including: business, commerce, connected car, connected home, fitness and wearables, mobile, Web etc. Similarly in BIG IoT users will be able to create new services or use services that are provided by other users.

3 Semantic based Discovery for Orchestration of Offerings

How to exactly discover necessary and desired offerings for a right composition and orchestration of offerings certainly plays a key role in BIG IoT. In D4.1.a is described the GrahpQL-based offering discovery mechanism that has been developed to search for and filter the offerings via marketplace portal. However, due to the higher requirements for offering discovery for offering orchestration we need a mechanism which supports discovery of offering in more complex situations. In this section we introduce methods for general offering discovery based on semantic technologies.

Using the BIG IoT Semantic Core Model (D3.2) and BIG IoT Semantic Application Domain Models (D4.2) we can semantically describe offerings which have typically different syntax. The semantic representation of offerings (Offering Description defined in D3.2) is stored in a triple-store in the BIG IoT Marketplace, see Figure 1. According to the intent of BIG IoT, Consumers do not need to know offerings, they can use the descriptions of requirements of offerings to discover wanted offerings on the marketplace. To enable to do that, it raises the question how to describe the requirements for the disired wanted offerings and how to discover the offering in the triple-store. In this work, we use SPARQL to describe the requirements of wanted offerings.

3.1 SPARQL supported Offering Discovery

SPARQL (SPARQL Protocol and RDF Query Language) is the most popular query language to retrieve and manipulate data stored in RDF. SPARQL defines an SQL like query language for making queries for RDF. SPARQL queries are executed against RDF datasets, containing of RDF graphs. A typical SPARQL query is a text message that retrieves a document of an RDF graph. There exist several tools to create and process SPARQL queries. You can find details about SPARQL in D2.1.

In our approach, offering discovery is based on discovery of rdfTypes (semantic annotation) of wanted offerings and rdfTypes of its input/output parameters which are defined in the Offering Description. The rdfType are category classes in semantic model which are used to define types of complex input/output parameters. Offerings which contain all rdfTypes desired by wanted offerings can be discovered. In order to be able to realise it, a SPARQL query can automatically be created by description of rdfTypes of desired offerings and his input/out parameter, when user has used recipe to create the composition of offerings. How to create a recipe will be introduced in Section 4 in details. After that, the SPARQL Engine, an implementation of SPARQL, enables the discovery of wanted offerings in the triple-store based on such a SPARQL query.

In the following, we show a SPARQL query example in Figure 2.

```
     SELECT ?offering_1
     WHERE {
         ?offering_1 bigiot:category mobility:parkingSiteMgmtCategory .
         ?offering_1 td:hasInputData ?input_1 .
         ?input_1 bigiot:rdfType "geo:SpatialThing" .
         ?offering_1 td:hasOutputData ?output_1 .
         ?output_1 bigiot:hasMembers  ?member_1.
         ?member_1 bigiot:rdfType "mobili-
ty:parkingSpaceOrGroupIdentifier" .
         ?output_1 bigiot:hasMembers  ?member_2.
         ?member_2 bigiot:rdfType "mobility:ParkingSpaceStatus" .
         ?output_1 bigiot:hasMembers  ?member_3.
         ?member_3 bigiot:rdfType "mobility:parkingStatusTime" .
     }
```

Figure 2, SPARQL example

In this example we describe wanted offerings which should belong to a category mobility:parkingSiteMgmtCategory. It should have one input parameter with type geo:SpatialThing and three output parameters, the first one with type mobility:parkingSpaceOrGroupIdentifier, the second one with type mobility:ParkingSpaceStatus, and the last one with type mobility:parkingStatusTime. According to this SPARQL query, an offering GetParkingSpotInfo which is stored in the triple-store can be discovery, since it matches all requirements in SPARQL query patterns, it means, it has the same rdfTypes which wanted offering desires. The complete example for the offering GetParkingSpotInfo can be found in Annex 3. This SPARQL query example can be found in Annex 1 and it will be introduced in Section 4 in detail.

3.2 Visualization of Discovery Results

In order to easily create recipe, a GUI will be created to easily compose offerings. Different to GUI in marketplace portal, which provides looking for and filtering offering in marketplace (see D4.1.a), this GUI is used for creating recipes (composition of offerings), used for easily managing the discovery results for recipe, and creating instances of recipe with the discovered offerings. How to create a recipe and how to create an instance of recipe will be introduced in Section 4.5. One of important tasks of this GUI is to visualize the discovery results of desired offerings for composition of offerings (recipe).

After definition of requirements for desired offerings in the recipe GUI, the system will automatically create a SPARQL query and discovery the desired offerings in triple-store. The discovery results (desired offerings) will be displayed as a list for user on the recipe GUI. User can select the offerings in the list to create an instance of recipe (creating a new offering with existing offerings).

This is on-going work. The complete results will be reported in next deliverables D4.3.b and D4.3.c.

European Commission
Horizon 2020
European Union funding
for Research & Innovation

4 Semantic based Offering Orchestration with Recipes

The BIG IoT marketplace is built to enable and monetize IoT applications. To maximize the usefulness and monetization of these marketplace applications, mechanisms are needed to enable their rapid and flexible development, composition, extension, reuse and sharing. Moreover, the competition on the marketplace is raised through multiple offerings for the same type of an application. In order to create reusable application templates that can be instantiated by multiple offerings, we introduce the concept of *Recipe*. Recipes enable rapid and flexible development, composition, extension, reuse and sharing of offerings. Further on, they help users to easily discover a right type of an application. Finally, they raise the competition on the marketplace as they enable multiple IoT platforms and developers to compete by providing their own implementations for a specific Recipe.

4.1 Overall Concept of Orchestration

In order to enable the Long Tail IoT market, the process of building new applications should be straightforward and transparent. Moreover, composition of existing services and applications (Offerings) should lead to an easy creation of new, added value offerings. Finally it is desirable to save these compositions as templates for later use; and enable effective discovery of them, and sharing on the marketplace. The question arises how to specify such a template in a convenient manner, so that new IoT applications can be created based on the reuse of existing Offerings or quick implementation of new ones. To answer this question, we provide the concept of Recipe. Recipe formalizes interactions between Offerings in a template. Interactions are based on atomic composable functionality provided by Offerings. Recipe is semantically described so that machines can discover it and interpret its content. It is easy to extend a Recipe by composing it with other Offerings. Finally, it is easy to share Recipes on a marketplace and instantiate it when implementing a new IoT application (Offering).

Figure 3 depicts an example instance of a Recipe. The Recipe has two Ingredients. The Ingredients may be instantiated by existing Offerings (i.e., GetParkingSpaceInfo and ReserveParkingSpace). BIG IoT-enabled platforms register their resources as Offerings on the BIG IoT Marketplace (see the lower part of Figure 3). These Offerings may either be consumed directly or be used to build added-value Offerings, so called recipes. For this purpose, a BIG IoT Application Developer discovers a Recipe or creates one. After this step, the developer needs to discover Offerings that fits as Ingredients of the Recipe. Ingredients, for which no Offering can be found on the marketplace, need to be implemented. A Recipe formalizes interactions between Offerings, and hence it provides help to the developer to correctly implement them (e.g., which output from an Ingredient can be fed as an input to another Ingredient). Apart from this, a Recipe specifies configuration parameters as required by an application, as well as non-functional properties (NFPs) and Quality of Service (QoS) parameters. An instantiated (implemented) Recipe may be published again as an Offering on the BIG IoT Marketplace.

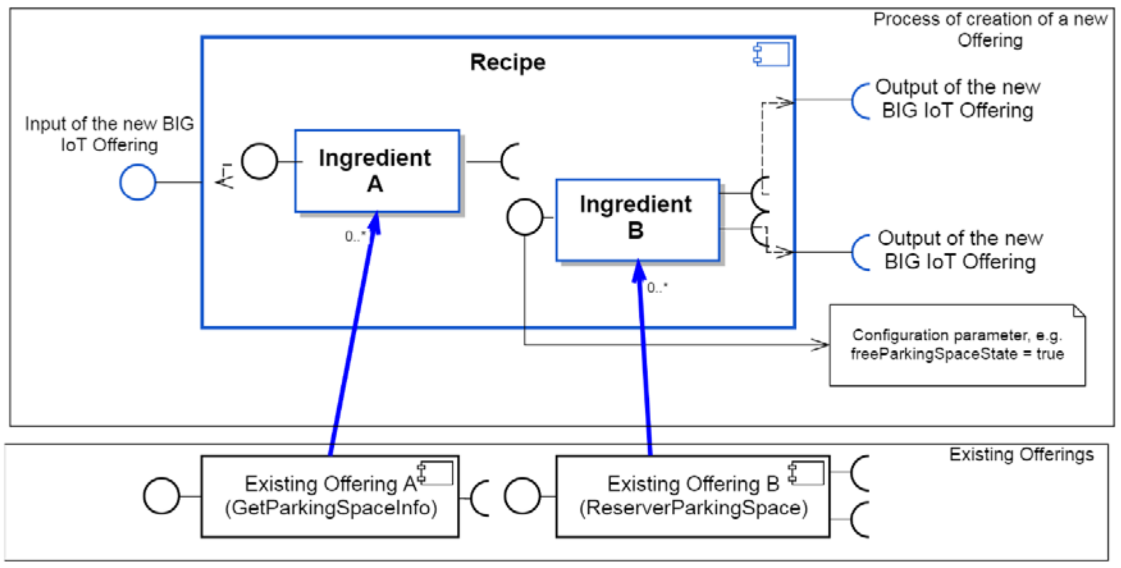
Process of creation of a new Offering
Recipe
Input of the new BIG IoT Offering
Ingredient A
Ingredient B
0..*
0..*
Output of the new BIG IoT Offering
Output of the new BIG IoT Offering
Configuration parameter, e.g. freeParkingSpaceState = true
Existing Offerings
Existing Offering A (GetParkingSpaceInfo)
Existing Offering B (ReserverParkingSpace)

Figure 3, Basic concept of Offering Composition

4.2 Recipe Model

In the process of building a BIG IoT Application, a Recipe aims to:

- Formalize interactions between Ingredients (e.g., Offerings);
- Specify configuration parameters as required by a BIG IoT Application;
- Enable definition of non-functional properties (NFPs) for a BIG IoT Application.

These three aspects of a Recipe are depicted in Figure 4. The Recipe "hasIngredients" is typically an Offering[1]. Apart from this, a Recipe specifies Interactions between Ingredients throughout a relation, called "hasInteraction". Each Interaction defines which output Data from an Ingredient needs to be fed as input Data of another Ingredient, as well as an Operation that needs to be applied (see Figure 4). Further on, non-functional properties (NFPs) for a Recipe can be defined too. Note that it is possible to define NFPs for Offerings. This can be done in an Offering description. In the scope of this work, NFP is referred to the composition of Offerings as a whole, i.e., to Recipe. Finally, it is worth noting that a Recipe itself is an Offering[2].

[1] Later on in this Section we will see another type of Ingredients, called Recipe Patterns.
[2] Apart from the „hasIngredient" relation, Recipe and Offering are related via a sub-class relation.

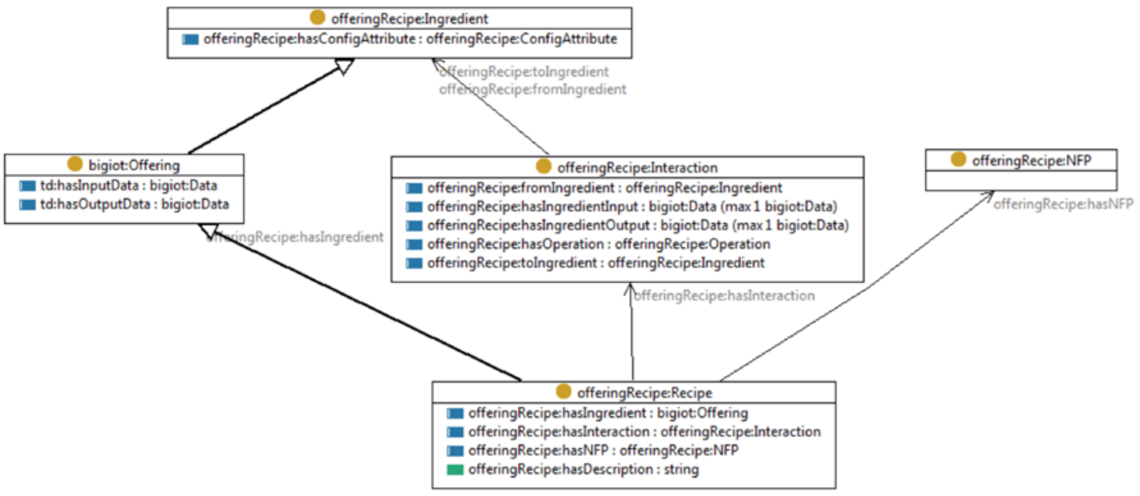
offeringRecipe:Ingredient
offeringRecipe:hasConfigAttribute : offeringRecipe:ConfigAttribute
offeringRecipe:toIngredient
offeringRecipe:fromIngredient
bigiot:Offering
td:hasInputData : bigiot:Data
td:hasOutputData : bigiot:Data
offeringRecipe:hasIngredient
offeringRecipe:Interaction
offeringRecipe:fromIngredient : offeringRecipe:Ingredient
offeringRecipe:hasIngredientInput : bigiot:Data (max 1 bigiot:Data)
offeringRecipe:hasIngredientOutput : bigiot:Data (max 1 bigiot:Data)
offeringRecipe:hasOperation : offeringRecipe:Operation
offeringRecipe:toIngredient : offeringRecipe:Ingredient
offeringRecipe:NFP
offeringRecipe:hasNFP
offeringRecipe:hasInteraction
offeringRecipe:Recipe
offeringRecipe:hasIngredient : bigiot:Offering
offeringRecipe:hasInteraction : offeringRecipe:Interaction
offeringRecipe:hasNFP : offeringRecipe:NFP
offeringRecipe:hasDescription : string

Figure 4, Recipe Model

The Recipe Model is built upon the BIG IoT Semantic Core Model (see Deliverable D3.2a). Figure 5 shows how Offering with its input and output Data (defined in the BIG IoT Semantic Core Model with prefix “bigiot”) is related to Ingredient (defined in the Recipe Model with prefix “offeringRecipe”).

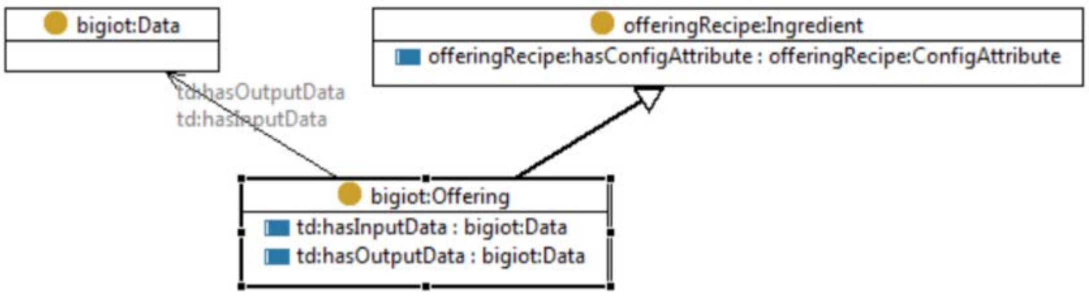
bigiot:Data
offeringRecipe:Ingredient
offeringRecipe:hasConfigAttribute : offeringRecipe:ConfigAttribute
td:hasOutputData
td:hasInputData
bigiot:Offering
td:hasInputData : bigiot:Data
td:hasOutputData : bigiot:Data

Figure 5, Offering in Recipe Model

Recipe further may specify configuration attributes “ConfigAttributes” for each Ingredient. Thus, for each Recipe, a user knows what configuration options that can be set (see Figure 6). Configuration attributes are the parameters used for the selection of Offerings. For example, configuration attributes for FreeBikeLocations type of Offerings can be specified as price, location, etc. The configuration attributes can be filled either by the Service Developer or a User while instantiating a Recipe. The configuration not only parameterizes an Offering or a Recipe, but also helps to filter out the Offerings based on configuration attributes.

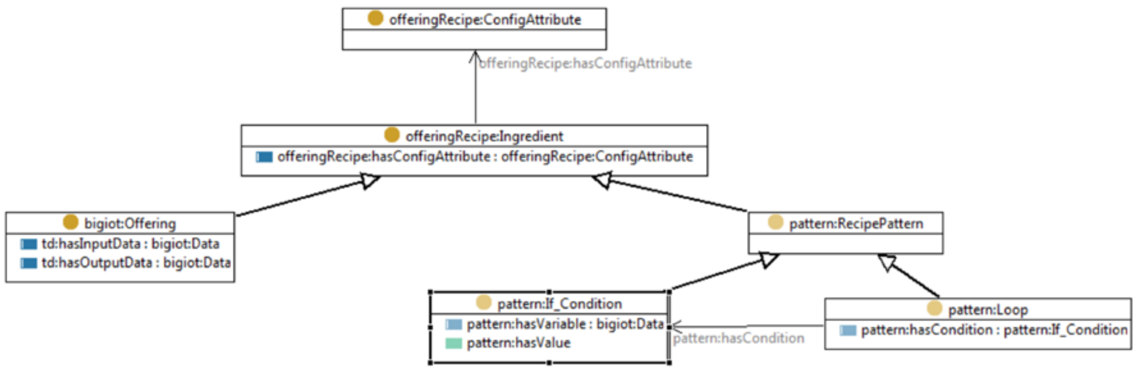
offeringRecipe:ConfigAttribute
offeringRecipe:hasConfigAttribute
offeringRecipe:Ingredient
offeringRecipe:hasConfigAttribute : offeringRecipe:ConfigAttribute
bigiot:Offering
td:hasInputData : bigiot:Data
td:hasOutputData : bigiot:Data
pattern:RecipePattern
pattern:If_Condition
pattern:hasVariable : bigiot:Data
pattern:hasValue
pattern:hasCondition
pattern:Loop
pattern:hasCondition : pattern:If_Condition

Figure 6, Ingredient in Recipe Model

The complete Recipe Model in RDF can be found in Annex 2.

4.3 Composition of Offerings with Recipes Patterns

A Recipe may specify a composition of Offerings. A Composition specifies how Ingredients (Offerings) are put together in order to provide an added value Service (Offering). In addition, Recipe defines constraints on the composition, i.e., under which conditions Ingredients (Offerings) may be put together. Therefore a Recipe may be seen as a specification of connections between Ingredients (Offerings) with certain conditions. For certain cases these conditions can be represented with so called Recipe Patterns. A Recipe Pattern is also an Ingredient (see Figure 5). Figure 5 shows only two types of Recipe Pattern for the sake of simplicity (i.e., if condition and loop). But in the following list we show a few additional examples of Recipe Patterns. Note, however that the list may be revisited after getting some experience from the modeling of Ingredients and Offerings based on BIG IoT use cases.

Recipes Patterns: If-Then, If-Then-Else, Sequence, Conjunction, Disjunction, Negation, Iterate, Repeat-until, Repeat-while, Split, Unordered list, Choice, Split+Join.

The list of Recipe Patterns is not exhaustive and is subject of change.

4.4 Benefits of Semantic-Based Recipes

A Recipe is a semantic template for BIG IoT Applications that can be easily created, discovered, reused, extended, and composed. Semantics play the key role in the concept of Recipes with the following benefits:

- Semantic Discovery: Semantic model for the Recipes enable discovery of several equivalent Offerings (possibly from different BIG IoT Platform Providers), which can be used to implement Recipes.
- Re-usability: Modeling an application as a composition of Offerings enables the re-usability of Offerings, as they can be used in many Recipes.
- Modularity of Offerings: For an Offering to be reusable, it needs to be modular. Thus, Recipes enforce the modular description of Offerings.
- Replacability of Offerings: In case, an Offering is not available anymore, the semantic description of a Recipe, where the Offering is instantiated, can be used to replace the Offering with an equivalent one.
- Interoperability of Recipes: Recipes are built out of standardized and semantically described ingredients. Thus they enhance interoperability.
- Efficiency: As the use of Offerings in a Recipe is not fixed (hard-coded), the end-user may choose the cheapest and highest quality available Offerings in order to run the Recipe.
- Higher revenue: Usage of Offerings in multiple Recipes enhances the revenue for a BIG IoT Platform Provider.
- Competition among BIG IoT Platform Providers: The Recipes invoke competition among the BIG IoT Platform Providers to provide more attractive offerings required for the existing or new Recipes.
- Seamless access to a variety of complex applications to end-users: Semantic modeling of the Recipes provides seamless access to complex applications by enabling semantic based discovery and connectivity to Offerings.

- Machine interpretability of Recipes: Recipes are fully interpretable by machines. Thus intelligent machines as consumers and implementers of Recipes are enabled.
- Long tail market: Recipes enable quicker and cheaper implementations of BIG IoT applications. This may open the log tail market for applications that so far were cost ineffective.

4.5 Use Cases of Offering Orchestration Recipe

In this section, we introduce a Recipe example "ReserveFreeParkingSpotRecipe" to demonstrate how to create a Recipe and Recipe instance. To create this example, we use the Recipe model in Annex 2 and the BIG IoT Application domain model in D4.2, part of T4.2, which provides domain-dependent vocabulary for the "Mobility" domain.

In the following we describe the steps for implementing this Recipe example.

1. Creating a Recipe
2. Discover a Recipe
3. Discover the Offerings required by a Recipe
4. Implement a Recipe

Step 1: Create a Recipe

The "ReserveFreeParkingSpace Recipe" is a composition of two offerings "Get parking spot information offering" and "Reserve parking space offering", see Figure 7. It uses the "If-Then" pattern to check the status of the parking space and reserve the parking space, if it is available. This Recipe once created can be instantiated in multiple platforms with the matching offerings.

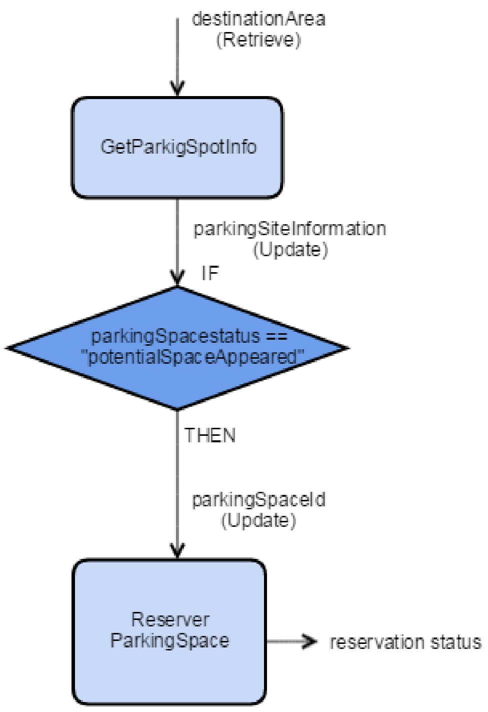

Figure 7, Workflow of ReserveFreeParkingSpotRecipe

The example Recipe "ReserveFreeParkingSpace" has three ingredients, "GetParkingSpotInformation", "ReserverParkingSpace" and "IfParkingSpaceAvailable". The first two ingredients correspond to the offerings from the BIG IoT Marketplace. The third ingredient is a Recipe pattern.

The category, input and output data of the ingredients are described in the Recipe using the BIG IoT application domain vocabulary. The offerings matching to the Recipe can be discovered from a triple store (where the Offering Descriptions are stored) based on the offering category, input and output types described for the ingredients in the Recipe. The Recipe also describes the configuration attributes for its ingredients. In this example, the "GetParkingSpaceInformation" ingredient has the configuration attribute named "vehicleType". This can be used to configure the Recipe during instantiation, e.g., by selecting the vehicle type such as "Car", "Bicycle" or "MotorBike" e.t.c for which the developer wants to get the parking information.

The interactions between the ingredients are described by the “Interactions” class of the Offering Recipe model. In this example, two interactions between the ingredients “GetParkingSpotInformation”, “IfParkingSpaceAvailable” and between “IfParkingSpaceAvailable”, “ReserverParkingSpace” are described. The interactions also describe the flow of data between the ingredients of the Recipe.

As explained in the above sections, a Recipe can also have Non-Functional properties such as “Price”, “Location” etc. In this example, the Recipe has “price” as the non-functional property which specifies the price for the Recipe. The property “hasDescription” is used to provide a human readable description about the recipe. The complete example can be found in Annex 5.

Step 2: Discover a Recipe

The created recipes are stored in the triple store where the Offering Descriptions and BIG IoT application domain model are also stored. A marketplace developer can discover and instantiate a recipe from the triple store by executing a SPARQL query on the SPARQL endpoint of the triple store. Query_1 in Annex 1 shows a sample SPARQL query that can be used to discover the “ReserveParkingSpace Recipe” based on the offering categories of the ingredients present in the recipe.

A Recipe can also be discovered by its category or based on the category and input, output types of its ingredients. For example, a developer can discover a recipe that belongs to “parking” category.

Step 3: Discover the Offerings required by a Recipe

After discovering the desired Recipe from a triple store, the next step is to discover the offerings that match to the ingredients of the Recipe. As mentioned above, the ingredients in a Recipe are the offerings and recipe patterns. The Offerings in a Recipe are described using the BIG IoT Offering domain model and the category, inputData and outputData of the ingredients are described using the BIG IoT Application domain model. The same models are used to describe the offerings from the providers of different platforms. This enables the semantic interoperability between Recipes and Offering Descriptions.

Thus, a set of SPARQL queries can be generated from a Recipe which when executed retrieves suitable offerings from the triple store. Query_2 in Annex 1 shows a sample SPARQL query created from the example Recipe. If there are any Offering Descriptions (OD) in the triple store that match the query, then it retrieves those Offering Descriptions, see Annex 1. In our case, the query retrieves the ODs shown in the Annex 3 and Annex 4.

The developer can choose the offerings among the discovered offerings and can instantiate and implement the Recipe with the selected offerings. The Recipe instance is a semantic description, which has ingredient IDs that are replaced by the URIs of the selected offerings. The Recipe Instance for the current example is shown in the Annex 6. To the recipe instance can be created and stored in the triple store, if the developer wants to offer the recipe instance as an Offering in the BIG IoT Marketplace.

Step 4: Implement a Recipe

To implement a Recipe means to generate an application code which can access the running resources from the IoT platforms (offerings) by using the BIG IoT lib. A Recipe can be easily implemented by using the semantic description of the Recipe Instance. The semantic description of a Recipe Instance can generate a code skeleton based on the URIs of the Offerings used in the Recipe, interactions described in the Recipe and the Recipe pattern. The developer can easily extend this code based on his needs.

5 Conclusions and Future Work

This deliverable presents the results of the 1st iteration of the work related to the automated discovery and orchestration of offerings in the BIG IoT ecosystem, which has been carried out from M5 to M12 of the project. The work was carried out by Task 4.3, but also involved interactions and coordination with representatives from other project activities, in particular Task 3.2 (Semantic interoperability for smart object platforms and services), Task 4.2 (Semantic models for the application domain), and Task 2.3 (Requirements analysis and specification).

In BIG IoT, added-value services are created based on composition of existing offerings on the Marketplace. So we presented the work around the specification of one of the key concepts of the BIG IoT Marketplace: the semantic offering composition templates called *Recipe*. Thereby, a Recipe provides descriptions for the composition of offerings. It is a specification of requirements of an added value service, and hence it represents a template that can be fulfilled by multiple offerings. Recipes will bring value to the BIG IoT Marketplace by providing attractive added-value services to end users. On the other hand, they enhance competition on the marketplace as offerings in recipes may come from different (Offering) Providers. The 2nd and 3rd evolution of these concepts will be reported in next deliverables D4.3.b and D4.3.c.

In the future, It will be possible to create and store all the recipes in the main triple store and have the possibility to expose the collection of recipes (templates and instances) in the marketplace, this way the users will have the possibility to reuse existing resources. It will be possible to support the registration of new offerings provided by recipes created by users/providers that will be available by any consumer.

We are going to exploit the full potential of semantic querying and reasoning for an optimized offering discovery and composition. With an automated dynamic reacting system for discovery of offering and adaptation of instances of recipes to maximize productivity (changes in offerings availability, best price choice, etc).

Further, we are planning to implement for the Marketplace portal a graphical user interface that will allow users to create dynamically recipes in a drag and drop style. It will be possible to discover, visualize and filter recipes, offerings and functional patterns and arrange them as blocks in the interface to create your own recipe that fulfill your needs.

6 References

[1] D2.1, Analysis of technology readiness, BIG IoT, 2016.

[2] Chinnici, Roberto, et al. "Web services description language (WSDL) version 2.0 part 1: Core language," W3C recommendation 26 (2007): 19.

[3] UDDI, http://www.uddi.org/pubs/uddi_v3.htm

[4] WS-BPEL, https://www.oasis-open.org

[5] http://www.wsmo.org/

[6] https://www.w3.org/Submission/OWL-S

[7] H.Klus, Dissertaiton, „Anwendungsarchitektur-konforme Konfiguration selbstorganisierender Softwaresysteme,“, 2013.

[8] IFTTT, https://ifttt.com/

7 Annex

Annex 1: SPARQL Query

```
Query_1# Search for  a Recipe based on the
Categories of its ingredients.

SELECT ?recipe
WHERE {
    ?recipe offeringRecipe:hasIngredient ?ingredient_1 .
    ?ingredient_1 bigiot:category ?category_1 .
    ?recipe offeringRecipe:hasIngredient ?ingredient_2 .
    ?ingredient_2 bigiot:category ?category_2 .
    FILTER (?category_1 = mobility:parkingSiteMgmtCategory &&
            ?category_2 = mobility:parkingSpaceMgmtCategory)
}

Query_2# Search for Offerings
Matching to the Recipe

PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX bigiot: <http://w3c.github.io/bigiot/bigiotDomainModel#>
PREFIX mobility: <http://w3c.github.io/bigiot/mobility#>
PREFIX td: <http://www.w3c.org/wot/td#>
PREFIX parkingRecipe: <http://w3c.github.io/bigiot/parkingRecipe#>
SELECT ?offering_1 ?offering_2
WHERE {
    ?offering_1 bigiot:category mobility:parkingSiteMgmtCategory .
    ?offering_1 td:hasInputData ?input_1 .
    ?input_1 bigiot:rdfType "geo:SpatialThing" .
    ?offering_1 td:hasOutputData ?output_1 .
    ?output_1 bigiot:rdfType "mobility:parkingSpaceOrGroupIdentifier" .
    ?offering_1 td:hasOutputData ?output_2 .
    ?output_2 bigiot:rdfType "mobility:ParkingSpaceStatus" .
    ?offering_1 td:hasOutputData ?output_3 .

    ?output_3 bigiot:rdfType "mobility:parkingStatusTime" .
    ?offering_2 bigiot:category mobility:parkingSpaceMgmtCategory .
    ?offering_2 td:hasInputData ?input_21 .
    ?input_21 bigiot:rdfType "mobility:parkingSpaceOrGroupIdentifier"
}
```

Annex 2: Recipe RDF Model

```
     # baseURI: http://w3c.github.io/bigiot/offeringRecipeModel
     # imports: http://w3c.github.io/bigiot/RecipePatternModel
     # imports: http://www.w3c.org/wot/td
     # prefix: offeringRecipe

     @prefix bigiot: <http://w3c.github.io/bigiot/bigiotDomainModel#> .
     @prefix mobility: <http://w3c.github.io/bigiot/mobility#> .
     @prefix offeringRecipe:
<http://w3c.github.io/bigiot/offeringRecipeModel#> .
     @prefix owl: <http://www.w3.org/2002/07/owl#> .
     @prefix pattern: <http://w3c.github.io/bigiot/RecipePatternModel#> .
     @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
     @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
     @prefix td: <http://www.w3c.org/wot/td#> .
     @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

     pattern:RecipePattern
       rdfs:subClassOf offeringRecipe:Ingredient ;
     .
     bigiot:Data
       rdf:type owl:Class ;
       rdfs:subClassOf owl:Thing ;
     .
     bigiot:Offering
       rdf:type owl:Class ;
       rdfs:subClassOf offeringRecipe:Ingredient ;
     .
     <http://w3c.github.io/bigiot/offeringRecipeModel>
       rdf:type owl:Ontology ;
       owl:imports <http://w3c.github.io/bigiot/RecipePatternModel> ;
       owl:imports <http://www.w3c.org/wot/td> ;
       owl:versionInfo "Created with TopBraid Composer" ;
     .
     offeringRecipe:ConfigAttribute
       rdf:type owl:Class ;
       rdfs:subClassOf owl:Thing ;
     .
     offeringRecipe:Create
       rdf:type offeringRecipe:Operation ;
     .
     offeringRecipe:Delete
       rdf:type offeringRecipe:Operation ;
     .
     offeringRecipe:Ingredient
       rdf:type owl:Class ;
       rdfs:subClassOf owl:Thing ;
     .
     offeringRecipe:Interaction
       rdf:type owl:Class ;
       rdfs:subClassOf owl:Thing ;
       rdfs:subClassOf [
           rdf:type owl:Restriction ;
```

```
            owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
            owl:onClass bigiot:Data ;
            owl:onProperty offeringRecipe:hasIngredientInput ;
          ] ;
        rdfs:subClassOf [
            rdf:type owl:Restriction ;
            owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
            owl:onClass bigiot:Data ;
            owl:onProperty offeringRecipe:hasIngredientOutput ;
          ] ;
        rdfs:subClassOf [
            rdf:type owl:Restriction ;
            owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
            owl:onClass offeringRecipe:Ingredient ;
            owl:onProperty offeringRecipe:hasIngredientFrom ;
          ] ;
        rdfs:subClassOf [
            rdf:type owl:Restriction ;
            owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
            owl:onClass offeringRecipe:Ingredient ;
            owl:onProperty offeringRecipe:hasIngredientTo ;
          ] ;
      .
      offeringRecipe:NFP
        rdf:type owl:Class ;
        rdfs:subClassOf owl:Thing ;
      .
      offeringRecipe:Notify
        rdf:type offeringRecipe:Operation ;
      .
      offeringRecipe:Observe
        rdf:type offeringRecipe:Operation ;
      .
      offeringRecipe:Operation
        rdf:type owl:Class ;
        rdfs:subClassOf owl:Thing ;
        owl:equivalentClass [
            rdf:type owl:Class ;
            owl:oneOf (
                offeringRecipe:Create
                offeringRecipe:Delete
                offeringRecipe:Notify
                offeringRecipe:Observe
                offeringRecipe:Retrieve
                offeringRecipe:Update
              ) ;
          ] ;
      .
      offeringRecipe:Recipe
        rdf:type owl:Class ;
        offeringRecipe:hasConfigAttribute offeringRecipe:ConfigAttribute ;
        rdfs:subClassOf bigiot:Offering ;
        rdfs:subClassOf owl:Thing ;
      .
```

```
offeringRecipe:Retrieve
  rdf:type offeringRecipe:Operation ;
.
offeringRecipe:Update
  rdf:type offeringRecipe:Operation ;
.
offeringRecipe:hasConfigAttribute
  rdf:type owl:ObjectProperty ;
  rdfs:domain offeringRecipe:Ingredient ;
  rdfs:range offeringRecipe:ConfigAttribute ;
.
offeringRecipe:hasDescription
  rdf:type owl:DatatypeProperty ;
  rdfs:domain offeringRecipe:Recipe ;
  rdfs:range xsd:string ;
.
offeringRecipe:hasIngredient
  rdf:type owl:ObjectProperty ;
  rdfs:domain offeringRecipe:Recipe ;
  rdfs:range offeringRecipe:Ingredient ;
.
offeringRecipe:hasIngredientFrom
  rdf:type owl:ObjectProperty ;
.
offeringRecipe:hasIngredientInput
  rdf:type owl:ObjectProperty ;
  rdfs:domain offeringRecipe:Interaction ;
  rdfs:range bigiot:Data ;
.
offeringRecipe:hasIngredientOutput
  rdf:type owl:ObjectProperty ;
  rdfs:domain offeringRecipe:Interaction ;
  rdfs:range bigiot:Data ;
.
offeringRecipe:hasIngredientTo
  rdf:type owl:ObjectProperty ;
.
offeringRecipe:hasInteraction
  rdf:type owl:ObjectProperty ;
  rdfs:domain offeringRecipe:Recipe ;
  rdfs:range offeringRecipe:Interaction ;
.
offeringRecipe:hasNFP
  rdf:type owl:ObjectProperty ;
  rdfs:domain offeringRecipe:Recipe ;
  rdfs:range offeringRecipe:NFP ;
.
offeringRecipe:hasOperation
  rdf:type owl:ObjectProperty ;
  rdfs:domain offeringRecipe:Interaction ;
  rdfs:range offeringRecipe:Operation ;
.
offeringRecipe:hasRecipeURI
  rdf:type owl:DatatypeProperty ;
```

```
        rdfs:range xsd:anyURI ;
      .
      td:hasInputData
        rdf:type owl:ObjectProperty ;
        rdfs:domain bigiot:Offering ;
        rdfs:range bigiot:Data ;
      .
      td:hasOutputData
        rdf:type owl:ObjectProperty ;
        rdfs:domain bigiot:Offering ;
        rdfs:range bigiot:Data ;
      .
```

Annex 3: GetParkingSpotInfo Offering Description

```
     {
       "@context": ["http://big-iot.eu/ctx",
       {"mobility" : "http://big-iot.eu/mobility#"}],

     "name": "Parking Spot Availability Service",
       "@type": "offering",
       "category": "mobility:parkingSiteMgmtCategory",
       "providerId": "1452fwr",
       "inputData": [
         { "name": "areaSpecification", "rdfType": "schema:GeoCircle",
           "members": [{"name": "geoCoordinates", "rdfType": "sche-
ma:GeoCoordinates",
            "members": [{ "name": "latitude", "rdfType": "schema:latitude",
"valueType" : "number"},
                        { "name": "longitude", "rdfType": "sche-
ma:longitude", "valueType" : "number"}]},
                      { "name": "radius", "valueType": "num-
ber","rdfType":"schema:geoRadius" }]}
       ],
       "outputData": [
         {"name": "geoCoordinates", "rdfType": "schema:GeoCoordinates",
           "members":[{ "name": "latitude", "rdfType": "schema:latitude",
"valueType" : "number"},
                      { "name": "longitude", "rdfType": "schema:longitude",
"valueType" : "number"}]},
         { "name": "parkingSpotID", "valueType": "string",
"rdfType":"mobility:parkingSpaceOrGroupIdentifier" },
         { "name": "timestamp", "valueType": "xsd:dateTime",
"rdfType":"mobility:parkingSpaceStatusTimeStamp"},
         { "name": "status", "valueType": "string",
"rdfType":"mobility:ParkingSpaceStatus" }
       ],
       "endpoint": {
         "url": "http://example.org/pspot",
         "endpointType": "HTTP_GET",
         "accessInterfaceType": "BIGIOT_LIB"
       },
       "license": { "type": "OPEN_DATA_LICENSE" },
       "price": {
         "amount": 0.002,
         "currency": "EUR",
         "accounting": "PER_ACCESS"
       },
       "region": "http://sws.geonames.org/3128760/"
     }
```

Annex 4: ReserveParkingSpace Offering Description

```
     {
       "@context": ["http://big-iot.eu/ctx",
       {"mobility" : "http://big-iot.eu/mobility#"}],

     "name": "Parking Spot Reservation Service",
       "@type": "offering",
       "category": "mobility:parkingSiteMgmtCategory",
       "providerId": "1452fwr",
       "inputData": [ { "name": "parkingSpaceId", "rdfType": "mobili-
ty:hasParkingIdentifier", "valueType" : "string"},
                      { "name": "paymentMode", "rdfType": "mobili-
ty:PaymentMethodForParking", "valueType" : "string"}],
       "outputData":[ { "name": "parkingReservationStatus",
"rdfType":"mobility:ReservationStatus", "valueType": "string" }],
       "endpoint": {
         "url": "http://example.org/pspot",
         "endpointType": "HTTP_PUT",
         "accessInterfaceType": "BIGIOT_LIB"
       },
       "license": { "type": "OPEN_DATA_LICENSE" },
       "price": {
         "amount": 0.002,
         "currency": "EUR",
         "accounting": "PER_ACCESS"
       },
       "region": "http://sws.geonames.org/3128760/"
     }
```

Annex 5: ReserveFreeParkingSpace Recipe

```
{
  "@context": {
    "bigiot": "http://big-iot.eu/core#",
    "offeringRecipe":
"http://w3c.github.io/bigiot/offeringRecipeModel#",
    "owl": "http://www.w3.org/2002/07/owl#",
    "parkingRecipe": "http://w3c.github.io/bigiot/parkingRecipe#",
    "pattern": "http://w3c.github.io/bigiot/RecipePatternModel#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "schema": "http://schema.org/",
    "td": "http://www.w3c.org/wot/td#",
    "xsd": "http://www.w3.org/2001/XMLSchema#"
  },
  "@graph": [
    {
      "@id": "parkingRecipe:IfParkingSpaceAvailable",
      "@type": "pattern:If_Condition",
      "pattern:hasRelationalOperator": {
        "@id": "pattern:equalTo"
      },
      "pattern:hasValue": "mobility:potentialSpaceAppeared",
      "pattern:hasVariable": {
        "@id": "parkingRecipe:checkParkingSpaceStatus"
      },
      "td:hasOutputData": {
        "@id": "parkingRecipe:checkParkingSpaceId"
      }
    },
    {
      "@id": "http://big-iot.eu/mobility#DataValue",
      "rdfs:subClassOf": {
        "@id": "bigiot:Data"
      }
```

```
        },
        {
          "@id": "parkingRecipe:checkParkingSpaceId",
          "@type": "bigiot:Data"
        },
        {
          "@id": "parkingRecipe:parkingSpaceId",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "mobility:parkingSpaceOrGroupIdentifier"
          }
        },
        {
          "@id": "parkingRecipe:reserveFreeParkingSpaceRecipe",
          "@type": "bigiot:Offering",
          "offeringRecipe:hasIngredient": [
            {
              "@id": "parkingRecipe:GetParkingSpotInformation"
            },
            {
              "@id": "parkingRecipe:ReserverParkingSpace"
            },
            {
              "@id": "parkingRecipe:IfParkingSpaceAvailable"
            }
          ],
          "offeringRecipe:hasInteraction": [
            {
              "@id": "parkingRecipe:Interaction_1"
            },
            {
              "@id": "parkingRecipe:Interaction_2"
            }
          ],
          "td:hasInputData": {
            "@id": "parkingRecipe:areaSpecification"
          },
          "td:hasOutputData": {
            "@id": "parkingRecipe:reservationStatus"
          }
        },
        {
          "@id": "offeringRecipe:Recipe",
          "@type": "owl:Class",
          "rdfs:subClassOf": {
            "@id": "bigiot:Offering"
          }
        },
        {
          "@id": "parkingRecipe:ReserverParkingSpace",
          "@type": "bigiot:Offering",
          "schema:category": {
            "@id": "http://big-iot.eu/mobility#parkingSiteMgmtCategory"
          },
          "td:hasInputData": {
            "@id": "parkingRecipe:reserveparkingSpaceId"
          },
          "td:hasOutputData": {
            "@id": "parkingRecipe:reservationStatus"
          }
        },
        {
          "@id": "parkingRecipe:parkingSpaceStatus",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "mobility:ParkingSpaceStatus"
          }
        },
        {
          "@id": "parkingRecipe:reserveparkingSpaceId",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "mobility:parkingSpaceOrGroupIdentifier"
          }
        },
        {
          "@id": "parkingRecipe:Interaction_2",
          "@type": "offeringRecipe:Interaction",
          "offeringRecipe:hasIngredientFrom": {
            "@id": "parkingRecipe:IfParkingSpaceAvailable"
          },
          "offeringRecipe:hasIngredientInput": {
            "@id": "parkingRecipe:reserveparkingSpaceId"
          },
          "offeringRecipe:hasIngredientOutput": {
            "@id": "parkingRecipe:checkParkingSpaceId"
          },
          "offeringRecipe:hasIngredientTo": {
            "@id": "parkingRecipe:ReserverParkingSpace"
          },
          "offeringRecipe:hasOperation": {
            "@id": "offeringRecipe:Update"
          }
        },
        {
          "@id": "http://big-iot.eu/mobility#Data",
          "rdfs:subClassOf": {
            "@id": "bigiot:Data"
          }
        },
        {
          "@id": "parkingRecipe:reservationStatus",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "mobility:ReservationStatus"
          }
        },
        {
          "@id": "parkingRecipe:Interaction_1",
          "@type": "offeringRecipe:Interaction",
          "offeringRecipe:hasIngredientFrom": {
            "@id": "parkingRecipe:GetParkingSpotInformation"
          },
          "offeringRecipe:hasIngredientInput": {
            "@id": "parkingRecipe:checkParkingSpaceStatus"
          },
          "offeringRecipe:hasIngredientOutput": {
            "@id": "parkingRecipe:parkingSpaceStatus"
          },
          "offeringRecipe:hasIngredientTo": {
            "@id": "parkingRecipe:IfParkingSpaceAvailable"
          },
          "offeringRecipe:hasOperation": {
            "@id": "offeringRecipe:Create"
          }
        },
        {
          "@id": "parkingRecipe:timestamp",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "mobility:parkingSpaceStatusTimeStamp"
          }
        },
        {
          "@id": "parkingRecipe:areaSpecification",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "schema:GeoCircle"
          }
        },
        {
          "@id": "parkingRecipe:parkingSpaceCoordinates",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "schema:Geocoordinates"
          }
        },
        {
          "@id": "http://w3c.github.io/bigiot/parkingRecipe",
          "@type": "owl:Ontology",
          "owl:imports": [
            {
              "@id": "http://w3c.github.io/bigiot/offeringRecipeModel"
            },
            {
              "@id": "bigiot:"
            },
            {
              "@id": "http://w3c.github.io/bigiot/RecipePatternModel"
            },
            {
              "@id": "http://big-iot.eu/mobility"
            }
          ],
          "owl:versionInfo": "Created with TopBraid Composer"
        },
        {
          "@id": "parkingRecipe:GetParkingSpotInformation",
          "@type": "bigiot:Offering",
          "schema:category": {
            "@id": "http://big-iot.eu/mobility#parkingSiteMgmtCategory"
          },
          "td:hasInputData": {
            "@id": "parkingRecipe:areaSpecification"
          },
          "td:hasOutputData": [
            {
              "@id": "parkingRecipe:timestamp"
            },
            {
              "@id": "parkingRecipe:parkingSpaceId"
            },
            {
              "@id": "parkingRecipe:parkingSpaceStatus"
            },
            {
              "@id": "parkingRecipe:parkingSpaceCoordinates"
            }
          ]
        },
        {
          "@id": "parkingRecipe:checkParkingSpaceStatus",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "mobility:ParkingSpaceStatus"
          }
        }
      ]
    }
```

Annex 6: ReserveFreeParkingSpace Recipe Instance

```
    {
      "@context": {
          "bigiot": "http://big-iot.eu/core#",
          "offeringRecipe":
"http://w3c.github.io/bigiot/offeringRecipeModel#",
          "owl": "http://www.w3.org/2002/07/owl#",
          "parkingRecipe": "http://w3c.github.io/bigiot/parkingRecipe#",
          "pattern": "http://w3c.github.io/bigiot/RecipePatternModel#",
          "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
          "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
          "schema": "http://schema.org/",
          "td": "http://www.w3c.org/wot/td#",
          "xsd": "http://www.w3.org/2001/XMLSchema#"
        },
        "@graph": [
          {
            "@id": "parkingRecipe:IfParkingSpaceAvailable",
            "@type": "pattern:If_Condition",
            "pattern:hasRelationalOperator": {
              "@id": "pattern:equalTo"
            },
            "pattern:hasValue": "mobility:potentialSpaceAppeared",
            "pattern:hasVariable": {
              "@id": "parkingRecipe:checkParkingSpaceStatus"
            },
            "td:hasOutputData": {
              "@id": "parkingRecipe:checkParkingSpaceId"
            }
          },
          {
            "@id": "http://big-iot.eu/mobility#DataValue",
            "rdfs:subClassOf": {
              "@id": "bigiot:Data"
            }
          },
          {
            "@id": "parkingRecipe:checkParkingSpaceId",
            "@type": "bigiot:Data"
          },
          {
            "@id": "parkingRecipe:parkingSpaceId",
            "@type": "bigiot:Data",
            "bigiot:rdfType": {
              "@type": "rdf:PlainLiteral",
              "@value": "mobility:parkingSpaceOrGroupIdentifier"
            }
          },
          {
            "@id": "parkingRecipe:reserveFreeParkingSpaceRecipe",
            "@type": "bigiot:Offering",
            "offeringRecipe:hasIngredient": [
              {
                "@id": "http://localhost:8080/bigiot/NG-
01/GetParkingSpotInfoOffering"
              },
              {
                "@id": "http://localhost:8080/bigiot/NG-
 01/ParkingSpaceReservationOffering"
              },
              {
                "@id": "parkingRecipe:IfParkingSpaceAvailable"
              }
            ],
            "offeringRecipe:hasInteraction": [
              {
                "@id": "parkingRecipe:Interaction_1"
              },
              {
                "@id": "parkingRecipe:Interaction_2"
              }
            ],
            "td:hasInputData": {
              "@id": "parkingRecipe:areaSpecification"
            },
            "td:hasOutputData": {
              "@id": "parkingRecipe:reservationStatus"
            }
          },
          {
            "@id": "offeringRecipe:Recipe",
            "@type": "owl:Class",
            "rdfs:subClassOf": {
              "@id": "bigiot:Offering"
            }
          },
          {
            "@id": "http://localhost:8080/bigiot/NG-
 01/ParkingSpaceReservationOffering",
            "@type": "bigiot:Offering",
            "schema:category": {
              "@id": "http://big-iot.eu/mobility#parkingSiteMgmtCategory"
            },
            "td:hasInputData": {
              "@id": "parkingRecipe:reserveparkingSpaceId"
            },
            "td:hasOutputData": {
              "@id": "parkingRecipe:reservationStatus"
            }
          },
          {
            "@id": "parkingRecipe:parkingSpaceStatus",
            "@type": "bigiot:Data",
            "bigiot:rdfType": {
              "@type": "rdf:PlainLiteral",
              "@value": "mobility:ParkingSpaceStatus"
            }
          },
          {
            "@id": "parkingRecipe:reserveparkingSpaceId",
            "@type": "bigiot:Data",
            "bigiot:rdfType": {
              "@type": "rdf:PlainLiteral",
              "@value": "mobility:parkingSpaceOrGroupIdentifier"
            }
          },
          {
            "@id": "parkingRecipe:Interaction_2",
            "@type": "offeringRecipe:Interaction",
            "offeringRecipe:hasIngredientFrom": {
              "@id": "parkingRecipe:IfParkingSpaceAvailable"
            },
            "offeringRecipe:hasIngredientInput": {
              "@id": "parkingRecipe:reserveparkingSpaceId"
            },
            "offeringRecipe:hasIngredientOutput": {
              "@id": "parkingRecipe:checkParkingSpaceId"
            },
            "offeringRecipe:hasIngredientTo": {
              "@id": "http://localhost:8080/bigiot/NG-
 01/ParkingSpaceReservationOffering"
            },
            "offeringRecipe:hasOperation": {
              "@id": "offeringRecipe:Update"
            }
          },
          {
            "@id": "http://big-iot.eu/mobility#Data",
            "rdfs:subClassOf": {
              "@id": "bigiot:Data"
            }
          },
          {
            "@id": "parkingRecipe:reservationStatus",
            "@type": "bigiot:Data",
            "bigiot:rdfType": {
              "@type": "rdf:PlainLiteral",
              "@value": "mobility:ReservationStatus"
            }
          },
          {
            "@id": "parkingRecipe:Interaction_1",
            "@type": "offeringRecipe:Interaction",
            "offeringRecipe:hasIngredientFrom": {
              "@id": "http://localhost:8080/bigiot/NG-
 01/GetParkingSpotInfoOffering"
            },
            "offeringRecipe:hasIngredientInput": {
              "@id": "parkingRecipe:checkParkingSpaceStatus"
            },
            "offeringRecipe:hasIngredientOutput": {
              "@id": "parkingRecipe:parkingSpaceStatus"
            },
            "offeringRecipe:hasIngredientTo": {
              "@id": "parkingRecipe:IfParkingSpaceAvailable"
```

```
          },
          "offeringRecipe:hasOperation": {
            "@id": "offeringRecipe:Create"
          }
        },
        {
          "@id": "parkingRecipe:timestamp",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "mobility:parkingSpaceStatusTimeStamp"
          }
        },
        {
          "@id": "parkingRecipe:areaSpecification",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "schema:GeoCircle"
          }
        },
        {
          "@id": "parkingRecipe:parkingSpaceCoordinates",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "schema:Geocoordinates"
          }
        },
        {
          "@id": "http://w3c.github.io/bigiot/parkingRecipe",
          "@type": "owl:Ontology",
          "owl:imports": [
            {
              "@id": "http://w3c.github.io/bigiot/offeringRecipeModel"
            },
            {
              "@id": "bigiot:"
            },
            {
              "@id": "http://w3c.github.io/bigiot/RecipePatternModel"
            },
            {
              "@id": "http://big-iot.eu/mobility"
            }
          ],
          "owl:versionInfo": "Created with TopBraid Composer"
        },
        {
          "@id": "http://localhost:8080/bigiot/NG-
01/GetParkingSpotInfoOffering",
          "@type": "bigiot:Offering",
          "schema:category": {
            "@id": "http://big-iot.eu/mobility#parkingSiteMgmtCategory"
          },
          "td:hasInputData": {
            "@id": "parkingRecipe:areaSpecification"
          },
          "td:hasOutputData": [
            {
              "@id": "parkingRecipe:timestamp"
            },
            {
              "@id": "parkingRecipe:parkingSpaceId"
            },
            {
              "@id": "parkingRecipe:parkingSpaceStatus"
            },
            {
              "@id": "parkingRecipe:parkingSpaceCoordinates"
            }
          ]
        },
        {
          "@id": "parkingRecipe:checkParkingSpaceStatus",
          "@type": "bigiot:Data",
          "bigiot:rdfType": {
            "@type": "rdf:PlainLiteral",
            "@value": "mobility:ParkingSpaceStatus"
          }
        }
      ]
    }
```