# Exploiting Aggregated Open Data from Smart Cities for the Future Internet Society

# D4.1: Design Methodologies Report

| | |
|---|---|
| Authors | Stefan Nastic, Ehsan Valizadeh |
| Institution lead | TU Wien (TUW) |
| Version | Final |
| Reviewers | Omer Ozdemir (ATOS) |
| Work package | WP4 |
| Task | 4.1 |
| Due date | 30/6/2017 |
| Submission date | 15/7/2017 |
| Distribution level: | Public (PU) |
| Abstract | This document outlines the **architecture of Micro Data Analytics Services (MiDAS)** for implemented as part of SMART-FI platform and describes novel SMART-FI **programming model for real time Big Data Analytics** |
| Keywords | Data analytics services, Stream programming model, Microservices, Real time Data analytics, Stream processing, Architecture |
| License information | This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/ |

# Document Description

## Document Revision History

| *Version* | *Date* | *Modifications Introduced* | |
|---|---|---|---|
| | | *Description of change* | *Modified by* |
| v0.1 | 28/03/17 | First draft version and TOC | Stefan Nastic (TUW) |
| v0.2 | 11/04/17 | Initial contributions | Javier Cubo (UMA) |
| v0.3 | 15/07/17 | Sent for revision | Stefan Nastic (TUW) |
| V0.4 | 15/09/17 | Contribution | Ehsan Valizadeh (SAMPAS) |
| V0.5 | 24/10/17 | Contribution | Ehsan Valizadeh (SAMPAS) |
| V0.6 | 25/11/2017 | Internal review | Omer Ozdemir (ATOS) |

# Table of Contents

# Table of Figures

# Terms and abbreviations

| | |
|---|---|
| WP | Work Package |
| MiDAS | Micro Data Analytic Services |
| UDF | User Defined Function |
| SOA | Service Oriented Architecture |
| RDF | Resource Description Framework |
| SPARQL | Spark Query Language |
| FaaS | Function as a Service |
| SLA | Service Level Agreement |
| QOS | Quality of Service |

# Executive Summary

In this deliverable we address the first objective of WP4: "developing advanced models of generic, elastic data analytic services for Big Data, in order to analyze the aggregated data for predictions and recommendations". In particular, we tackle the problem of defining the advanced models of generic, elastic data analytic services for Big Data. To this end we present the Micro Data Analytic Services (MiDAS) architecture and data analytics model. Moreover, we propose a solution to simplify the development of value-added data analytic services in order to facilitate exploiting the exposed data. To achieve this we developed a novel programming model for MiDAS, specifically focusing on the real-time data processing and streaming data.

# 1 Introduction

The present document describes the deliverable D4.1 of Task 4.1 "Data analytics techniques" in WP4 "Models of data analytic services". Generally, the main objective of WP4 is twofold: First, we will develop advanced models for programming generic, elastic data analytic services for Big Data, to analyze the aggregated data for predictions and recommendations. Second, we will implement tooling support for developing such data analytic services in order to facilitate exploiting the normalized Smart City data and turning it into disruptive innovation building blocks, based on micro services.

The purpose of Task 4.1 is identification, development and integration of Big Data Analytics techniques, which can be used to develop and execute value-added data analytics services and derive meaningful information from sensory data. Examples of such services could include predictive capabilities and recommendations in the context of the project pilots.

In general, deliverable D4.1 focuses on introducing the relevant concepts and architecture for SMART-FI data processing facilities. More details are given in the following section.

## 1.1 About this deliverable

The main objective of this deliverable is to provide design methodologies and models for implementing unified Micro Data Analytic Services (MiDAS) (e.g., batch and real time processing functions), based on the state-of-art data processing architectures, frameworks and models, in order to enable generic, flexible interoperation with different data processing and analytics frameworks. Moreover, the focus of this deliverable is on providing real time stream data analytics programming models, which are an integral part of MiDAS model and SMART-FI platform.

## 1.2 Document structure

Section 1 introduces the document; Section 2 gives a short overview of the deliverable and its context in the SMART-FI platform; Section 3 presents relevant state-of-the0-art work; Section 4 outlines MiDAS architecture; Section 5 gives a detailed overview of MiDAS real time data analytics programming model; Finally, Section 6 concludes the deliverable.

# 2 Deliverable Overview and Context

## 2.1 Relation to the main work package objectives

The main objective of WP4 is twofold: First, to develop advanced models of generic, elastic data analytic services for Big Data, in order to analyze the aggregated data for predictions and recommendations. Second, to implement tooling for developing the value-added data analytic services in order to facilitate exploiting the exposed data and turning it into disruptive innovation building blocks, based on micro services.

In this deliverable we address the first objective of WP4. In particular, we tackle the problem of defining the advanced models of generic, elastic data analytic services for Big Data. To this end we present the MiDAS architecture and data analytics model. Moreover, we propose a solution to simplify the development of value-added data analytic services in order to facilitate exploiting the exposed data. To achieve this we developed a novel programming model for micro data analytics services, specifically focusing on the real-time data processing and streaming data.

## 2.2 Relation to other work packages

Figure 1 illustrates the main dependencies of WP4. We will use the homogenized, aggregated data, developed in WP3, as the baseline for the models and techniques that will be developed as part of the WP4. The main outcomes of WP4 will be models and tools for developing elastic micro data analytic services, with a special focus on runtime service elasticity and governance concerns. In addition WP4 will provide example data analytic services (demonstrating the proposed model and tools) as well as an integration methodology for integrating micro data analytic services with service interoperability mechanisms (WP5). The outcomes of WP4 will be provided as inputs for the activities in WP5.

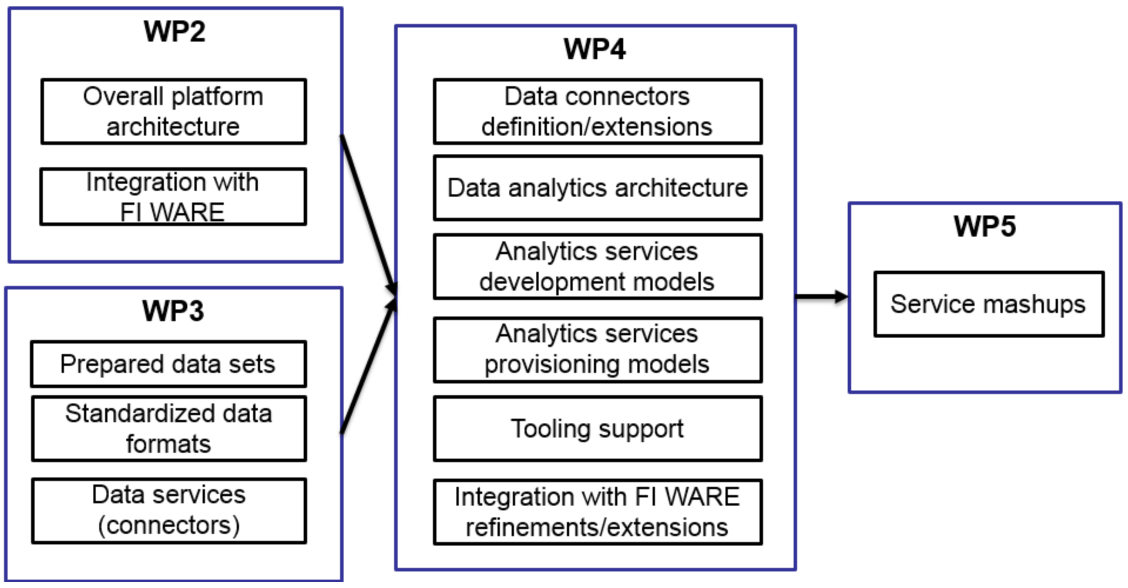
WP2
Overall platform architecture
Integration with FI WARE
WP3
Prepared data sets
Standardized data formats
Data services (connectors)
WP4
Data connectors definition/extensions
Data analytics architecture
Analytics services development models
Analytics services provisioning models
Tooling support
Integration with FI WARE refinements/extensions
WP5
Service mashups

**Figure 1 Overview of main action points, dependencies and outcomes of WP4**

### 2.2.1 Main action points

In this deliverable we mainly focus on the following action points defined within the Task 4.1:

- MiDAS architecture
- Data connectors
- MiDAS real-time analytics programming model

We discuss these action points in more detail in Section 4 and Section 5.

# 3 State-of-The-Art in Big Data Processing and Analytics

In this section, we consider related work containing research efforts and projects, at
different levels, as regards the main functionality provided by SMART-FI: i) Data normalization, ii) Data analytics, and iii) Service orchestration. In particular, in this deliverable we present the state-of-the-art in the area of data analytics and processing.

With respect to the data analytics, various underlying data processing frameworks, especially for streaming and batch analytics exist. Among the seminal papers, Agrawal et al. [1] and Cuzzocrea et al. [2] explored the possibility of running database management systems (DBMS) in a Cloud environment, concluding that there is not a one-size-fits-all solution, due to the trade-off between scalability and query expressiveness. The availability of huge amount of daily produced Smart City data fosters the exploration of new data analytics approaches which should simplify the ingestion, transformation, and consumption of data by possibly neglecting (or hiding) the complexity of managing a scalable and distributed processing system. Supported by concrete case studies, Hashem et al. [3] highlight the tight relationship that exists between Cloud computing and Big Data. The former provides the underlying engine that enables several classes of distributed data-processing platforms (e.g., batch processing, stream processing), whereas the latter might utilize distributed and fault-tolerant storage technologies based on Cloud resources in order to simplify the management and processing of data. Moreover, some research efforts envision and conceive a conceptual architecture that tightly combines the requirement of data analytics and the potentialities of Cloud computing. It results
a model named Cloud-based Analytics as a Service or Data Analytics as a Service. For example, Domenico Talia [4] discussed the complexity and variety of data types and processing power to perform analysis on large datasets. Therefore, he proposed three Cloud-based service models that support their execution: data analytics software as a service, where an analytic application or task is offered as a service, data analytics platform as a service, where analytic suites or frameworks are offered hiding the cloud infrastructure, and data analytics infrastructure as a service, where virtualized resources enable the storage and processing of Big Data. This idea is exploited also in other research papers, e.g., [5, 6], among which the one by Zulkernine et al. [6] presents a conceptual architecture for Cloud-based Analytics-aaS, which however includes only a preliminary implementation, lacking the details of how to process the massive dataset.

Other research contributions focus more on the scalable execution of user-defined functions (UDF) among a large and distributed set of computing nodes, which can be acquired or released as needed, encompassing the on-demand resource principle of Cloud computing. Nowadays, two opposite approaches are commonly adopted:

batch processing and stream processing [7]. The former stores all the data, usually on a distributed file system, and then operates on them on the basis of different programming models, among which the well-known MapReduce. The latter processes all the data on-the-fly, i.e., without storing them, so it can produce results in a near real-time fashion. A plethora of frameworks is available to process the data following one or the other approach: examples of batch processing frameworks are Apache Hadoop[1] (an open-source implementation of MapReduce) and Apache Tez[2]. Examples of stream processing frameworks are Apache Storm [8], Apache Flink[3], IBM Infosphere [9], and Amazon Kinesis[4].

Finally, there have been several initiatives in European Union under 7th Framework Programme and Horizon 2020. The SUPERSEDE[5] project proposes a feedback-driven approach to the life cycle management of software services and applications, with the ultimate purpose of improving users' quality of experience. Decisions on software evolution and runtime adaptation will be made upon analysis of end-user feedback and large amount of data monitored from the context. An integrated platform will articulate the methods and tools produced in the project. MARKOS[6] project uses data analytics techniques for the analysis of software sources, and the support to consume these data, migrated to a RDF/S repository and accessing through SPARQL Endpoints.

Most of the presented solutions could be used to execute data analytics processes. However, there is a lack of tools to generate/accelerate elastic data analytics services that utilize these frameworks to handle large-scale data to offer new analytics under micro services models. Most of the time, the developer has to write all analytics functions, service interfaces and complex configurations for the runtime concerns.

# 4 Architecture of Micro Data Analytics Services

In this section, we provide an architectural overview of Micro Data Analytics Services (MiDAS). Moreover we discuss its main components and how MiDAS fits into the overall SMART-FI platform design.

## 4.1 MiDAS Architecture and main design principles

Generally, the main purpose of Smart City data analytics services is enabling transforming the city data into disruptive innovation building blocks for the Smart Cities of the future, based on micro services technologies. To this end, MiDAS facilities (which are an integral part the SMART-Fi platform) provide models and components that allow for developing and managing value-added data analytic services in Smart Cities. MiDAS purpose is twofold: First, it provides advanced models for programming generic, elastic data analytic services, in order to facilitate analyzing the aggregated data for predictions and recommendations. Second, it implements tooling support for developing and managing such micro data analytic

[1] http://hadoop.apache.org/

[2] https://tez.apache.org/

[3] https://flink.apache.org/

[4] https://aws.amazon.com/kinesis/

[5] https://www.supersede.eu/

[6] http://www.markosproject.eu

services (which is being developed in the context of Task 4.2). In the remainder of this section, we mainly focus on the first aforementioned objective.

Figure 2, shows the architecture overview of the micro data analytics services that are capable to support both online and offline Smart City data analytics in a uniform way. The overall data analytics architecture of SMART-FI platform is based on the Lambda architecture[7]. It comprises three main layers: *Real-time Data Analytic Layer, Batch Analytics Layer and Fusion and Serving Layer*. The most important components of the data analytics are the micro data analytics services (depicted as shaded boxes in Figure 2 include:

- **Batch view function**, used to precompute static (slow-changing) partial aggregate views.
- **Stream transformation function**, used to compute real-time window deltas (real-time delta views).
- **Fusion function**, used to combine partial aggregate with real-time delta views and serve the results proactively or on-demand, enabling push or pull based interaction.

Subsequently, we describe these components in more details, mainly focusing on the *Real-time Data Analytic Layer*. The components depicted as dotted-line boxes in Figure 2 represent the computed data views. They are not directly exposed to users and serve as inputs to the Fusion functions. We tackle the problem of realizing the *Fusion functions* in the context of WP5.

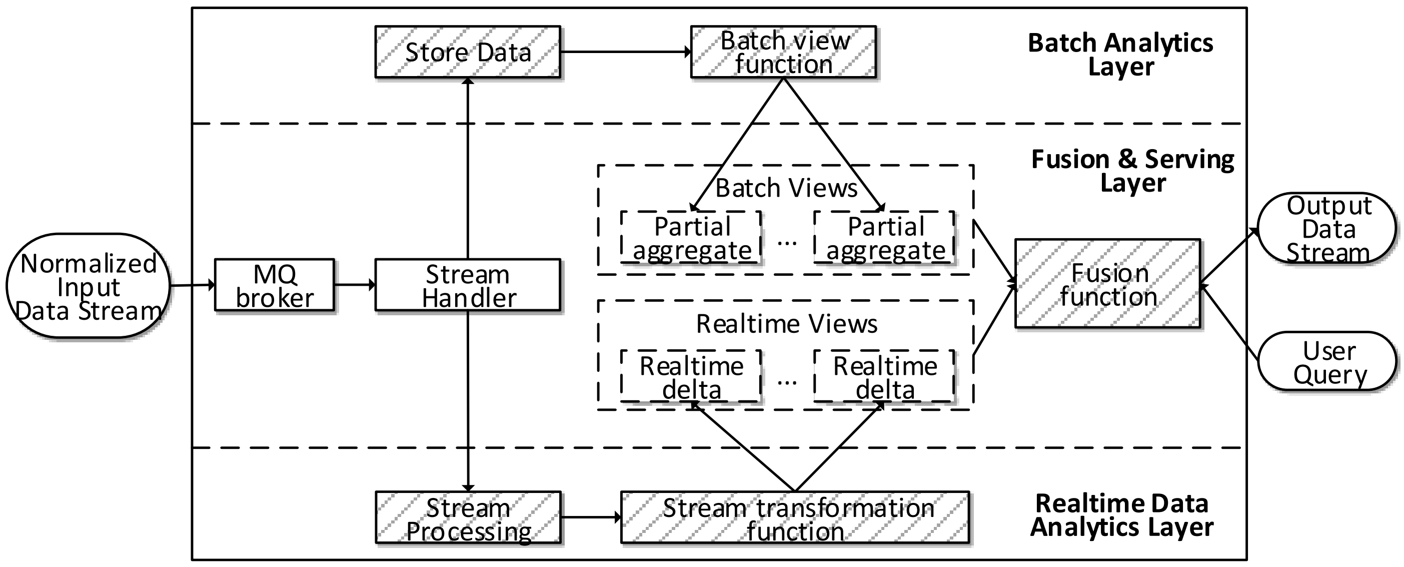

**Figure 2 MiDAS Services Architecture Overview.**

In our platform, we provide a novel model for real-time data analytics, which treats the data streams as first class citizens. In general, there is one-to-one mapping between MiDAS and data streams. From architectural point of view an instance of MiDAS is a logical entity identified by an *ID* (or URI) and its model is characterized by the following main three elements:

[7] http://lambda-architecture.net/

- **Stream data**: the sequence of events that constitutes the stream. Every new event is handled by the *Stream Processing* component that triggers the update of the downstream MiDAS, i.e., the streams in a dependent relationship with the current one. The events in a stream can be volatile or temporary stored.

- **Stream Transformation function**: this is a stateless function defined by the user, which transforms the incoming events in new events, according to the contract definition. The transformation function is automatically managed by the execution environment to support elastic scaling, runtime governance and QoS.

- **MiDAS contract**: Generally, the contract defines the type of the stream and encapsulates its most important properties, such as operational mode (i.e., window-based, partition-based mode), side effects and SLAs. Therefore, MiDAS contract can be seen as complex data type in a type system, which is related to the data transformation function.

In section 5, we discuss these concepts in more detail and give implementation overview. We specifically focus on the programming model perspective and how end users (e.g., developers) benefit from MiDAS model in practice.

## 4.2 MiDAS in the context of SMART-FI platform

### 4.2.1 SMART-FI platform overview

In this section, we show how the MiDAS architecture and models fit into SMART-FI platform architecture. In addition, we clarify the architectural and functional connections/dependencies of MiDAS and the rest of the SMART-FI platform (see Section 4.2.3).

The SMART-FI platform is the central component of the SMART-FI ecosystem that serves as one of the main building blocks and a cornerstone for developing a sustainable SMART-FI ecosystem. It enables developing, managing and interoperating Smart City data analytics services, in order to facilitate exploiting Smart City open data and optimizing various city sectors, such as transportation, governance services and urban energy. The main objective of the SMART-FI platform is to allow horizontal integration of open city data and data analytics services by providing a set of generic component and mechanism that will enable development of higher-level Smart city applications and services (see Figure 3 top).

Figure 3 (bottom) depicts how SMART-FI platform is based on the FIWARE and it utilizes its several components. However, it goes one step beyond by developing generic components and mechanisms based on microservices technologies, in such a way that other Smart City platforms could seamlessly adapt and incorporate SMART-FI components to suit their needs.

### 4.2.2 Core SMARAT-FI platform facilities

At the SMART-FI platform facilities level (Figure 3), three layers represent the main components, as well as a Governance and Service Market Place. Each facility is capable to perform a set of processes to get its main goal. Here we describe briefly the main data flow between these components. Heterogeneous data sets are managed in the *Data normalization facility*. It generates data sets with access rules and a normalized schema, based on urban ontologies, which are stored in semantic data store.

These normalized data streams serve as the main input for and are used by the MiDAS. As described previously in Section 3, the MiDAS analytic model and services enable the development and management of data analytic services in Smart Cities, providing elastic data analytic services to analyses the aggregated data for predictions and recommendations.

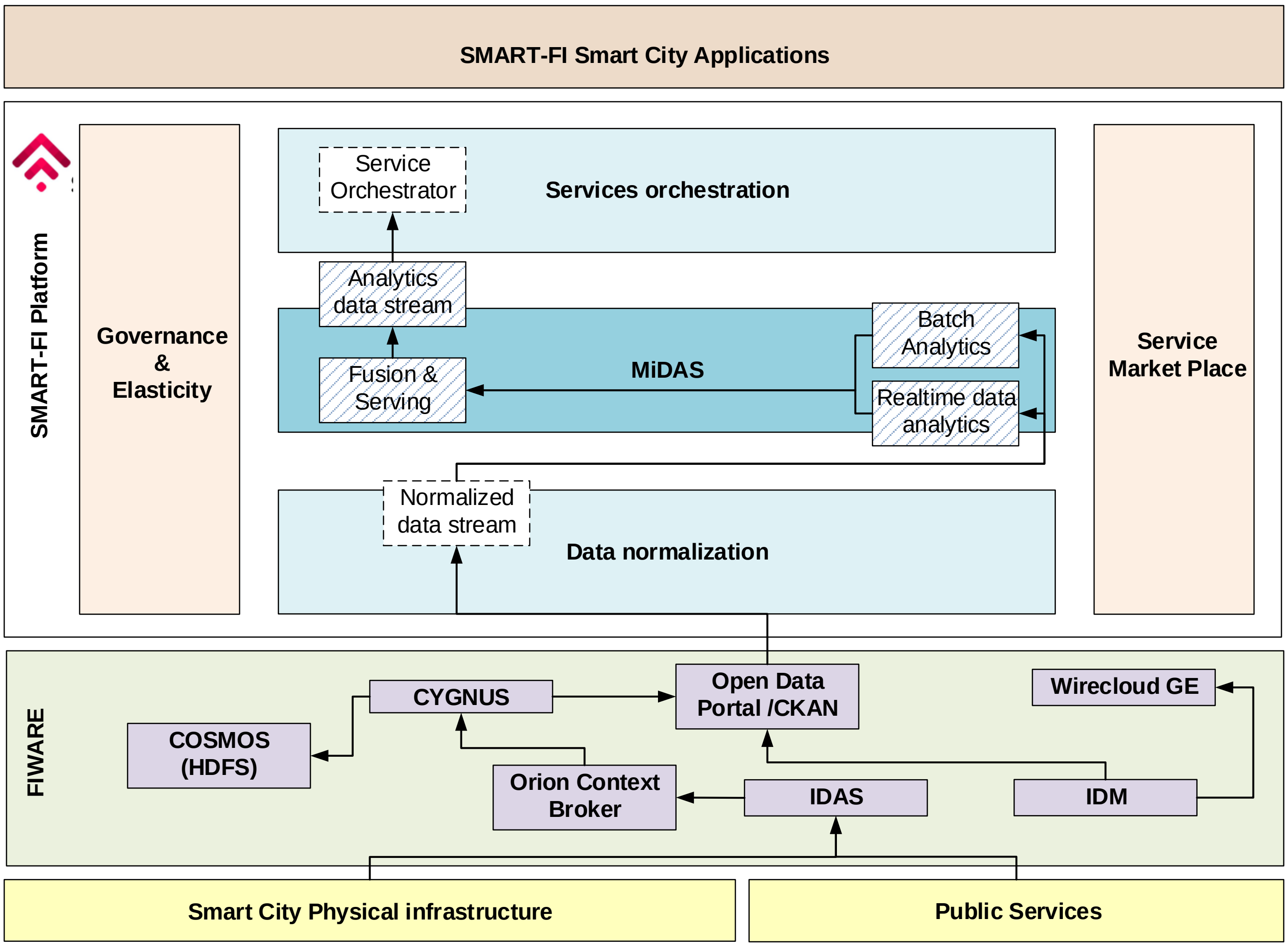

**Figure 3 SMART-FI Platform Architecture Overview.**

Finally, MiDAS facility provides a set of functions and analytics data streams, which are consumed and orchestrated by *Service orchestration* facility. With the Service orchestration facility, mechanisms for deploying and integrating existing or new services and applications will be provided, obtaining applications (mashups of services) and creating a marketplace that considers the FIWARE Lab and third-party applications.

More details on the SMART-FI platform and its main facilities can be found in our book chapter[8].In the following section, we describe the key components of SMART-FI platform that represent main dependencies of MiDAS facility.

[8] Stefan Nastic, Javier Cubo, Malena Donato, Schahram Dustdar, Orjan Guthu. Mats Jonsson, Omer Ozdemir, Ernesto Pimentel, M. Serdar Yumlu. „Exploiting Aggregated Open IoT Data from Smart Cities in the Future Internet Society".Springer Series, Internet of Things Technology, Communication and Computing, 2017.

### 4.2.3 Data Connectors & MiDAS functional dependencies

Implementing a Smart Application requires gathering and managing context information, referring to values of attributes characterizing relevant entities. Systems dealing with management of city services or third-party apps (subject to access control policies) can consume and produce context info. Overall city governance can rely on context information available (real-time and historic) to monitor KPIs and run Big Data analysis.

**Context Management in FIWARE**

The Publish/Subscribe Context Broker is a GE of the FIWARE platform that exposes the (standard) interfaces for retrieval of the context information, events and other data from the Context or Data/Event Producers to the Context or Data/Event Consumers. The consumer doesn't need to know where the data are located and what the native protocol for their retrieval is. It will just communicate to the Publish/Subscribe Context Broker GE through a well-defined interface specifying the data it needed in a defined way: on request or on subscription basis. The Publish/Subscribe Context Broker GE will provide the data back to the consumer when queried, in case of "on-request", or when available, in case of "on-subscription" communication mode.

The following figure shows a logical architecture of the Publish/Subscribe Context Broker GE in FIWARE with its main components and interactions with other actors.

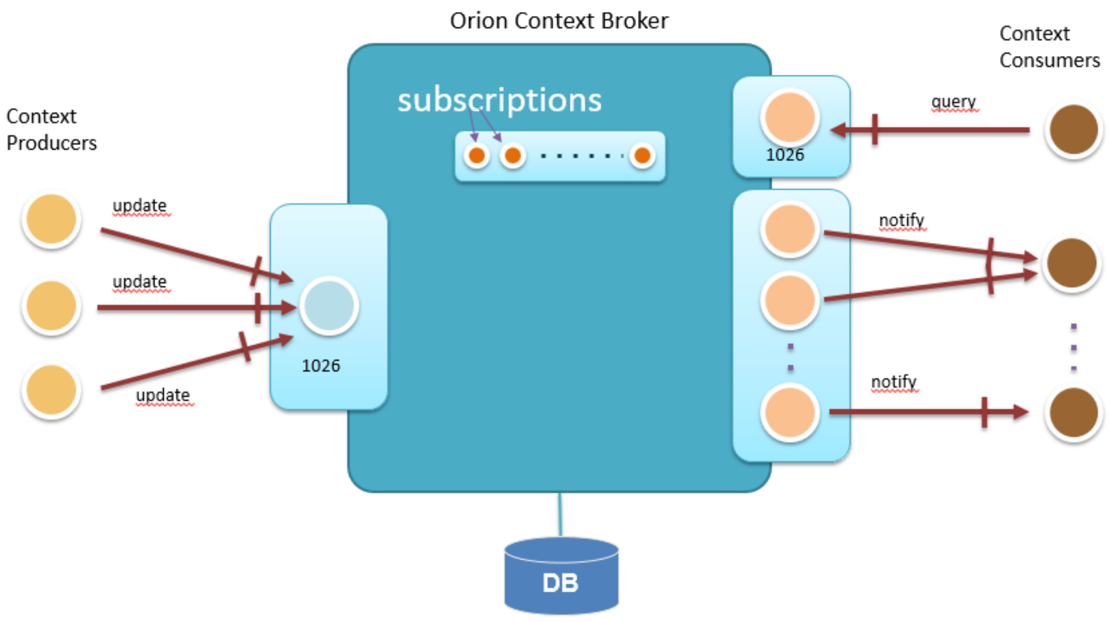

**Figure 4 Logical architecture of the Publish/Subscribe Context Broker GE**

The FIWARE Context Broker GE implements the OMA NGSI- 9/10 API: a simple yet powerful standard API for managing Context information complying with the requirements of a smart city.

Orion Context Broker allows to manage all the whole lifecycle of context information including updates, queries, registrations and subscriptions. Using the Orion Context Broker, it is possible to register context elements and manage them through updates and queries, in addition, subscribe to context information so when some condition occurs (e.g. a context element has changed) consumer receive a notification. These usage scenarios and the Orion Context Broker features

Apart from Orion Context Broker, there are other related components that you may find useful, such as Cygnus or Steelskin PEP. Cygnus implements a connector for context data coming from Orion Context Broker and aimed to be stored in a specific persistent storage, such as HDFS, CKAN or MySQL. Steelskin PEP is a proxy meant to secure Orion Context Broker, by intercepting every request sent to the Orion, validating it against the Access Control component.

**Context Elements**

Aligned with the standard OMA NGSI specification, Context Information in FIWARE is represented through generic data structures referred to as Context Elements. A Context Element refers to information that is produced, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. It has associated a value defined as consisting of a sequence of one or more <name, type, value> triplets referred to as context element attributes. FIWARE will support a set of built-in basic data types as well as the possibility to define structured data types: vectors and key-maps (which elements can be other vectors, key-maps or simple data types).

A Context Element typically provides information relevant to a particular entity, being it a physical thing or part of an application. As an example, a context element may contain values of the “last measured temperature”, “square meters” and “wall color” attributes associated to a room in a building. That's why they typically contain an EntityId and an EntityType uniquely identifying the entity. Finally, there may be meta-data (also referred to as semantic data) linked to attributes in a context element. However, the existence of meta-data linked to a context element attribute is optional.

In summary, context information in OMA NGSI is represented through data structures called context elements, which have associated:

- An EntityId and EntityType, uniquely identifying the entity to which context data refers.
- A sequence of one or more data element attributes (<name, type, value> triplets)
- Optional meta-data linked to attributes (also <name, type, value> triplets)

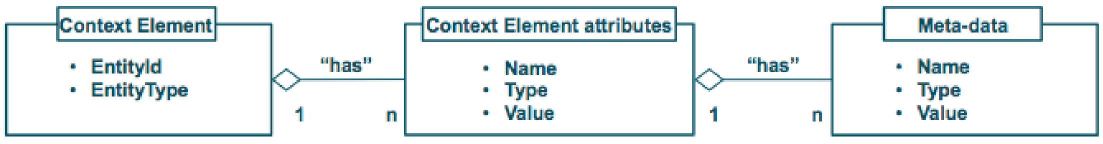

**Figure 5 Context Elements**

**Interactions related to forwarded updates to Context Consumers**

Publish/Subscribe Context Broker is able to forward context update to Context Consumers able to process that operation. This is the typical case for actuation functionality based on update side-effects as Context Consumer

- Context Producer can send an updateContext operation to publish/subscribe Context Broker

- Publish/Subscribe Context Broker forwards the updateContext to the appropriated Context Consumer and returns the result (either success or error) to the originating Context Provider

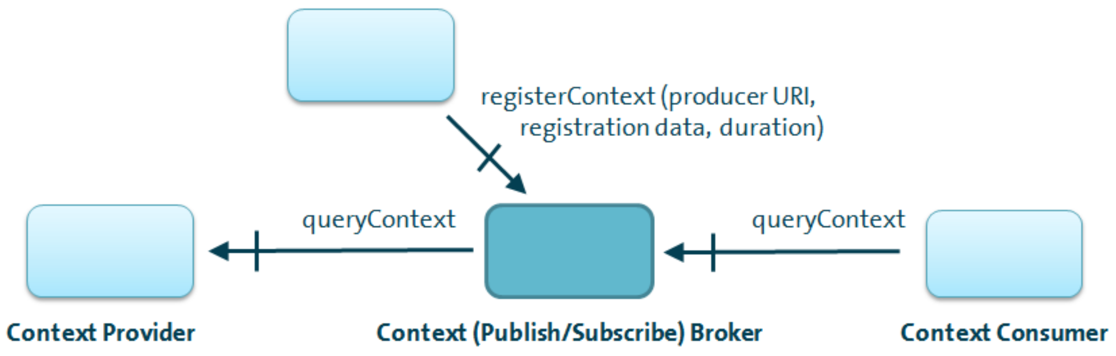
registerContext (producer URI, registration data, duration)
queryContext
queryContext
Context Provider
Context (Publish/Subscribe) Broker
Context Consumer

**Figure 6 Interactions related to forwarded updates to Context Consumers**

SMART-FI platform uses Orion Context Broker in order to manage context and context availability, it is HTTP and REST-based and used for creating and pulling data, pushing data/notifications and standard operations. Context information may come from many sources: existing systems, users, through mobile apps and sensor networks (Internet of Things). Capturing data from, or acting upon, IoT devices become as easy as to read/change the value of attributes linked to context entities using a Context Broker. [10]

The FIWARE NGSI API is Restful: any web/backend programmer gets quickly used to it.

**JSON payload features**

- Flexible to accommodate different data flavors
- Terse (only the essential is provided)
- Simple and ready for front-end data consumers
- It can represent any dataset
    - As a collection of entities with the same type

**API Overview**

- **Simple operations (RESTful)**
    - Useful for app developers (Northbound interfaces)
    - One "transaction" per request
- **Bulk operations (RPC Style → HTTP POST)**
    - Allow to perform many simple operations at the same time
    - Mostly useful for more complex backend processes
    - South / Eastbound interfaces
    - Original OMA-NGSI with simplified payload binding

- **Operation Groups**
  - Query Operations
    - Northbound interfaces
  - Update Operations
    - For creating new data available on the system
  - Subscribe operations
    - For subscription to change in data
  - Register operations
    - For associating external providers to data items
- **Entities**
  - GET /v2/entities
    - Retrieve all entities
  - POST /v2/entities
    - Creates an entity
  - GET /v2/entities/{entityID}
    - Retrieves an entity
  - [PUT|PATCH|POST] /v2/entities/{entityID}
    - Updates an entity (different "flavors")
  - DELETE /v2/entities/{entityID}
    - Deletes an entity
- **Attributes**
  - GET /v2/entities/{entityID}/attrs/{attrName}
    - Retrieves an attribute's data
  - PUT /v2/entities/{entityID}/attrs/{attrName}
    - Updates an attribute's data
  - DELETE /v2/entities/{entityID}/attrs/{attrName}
    - Deletes an attribute
  - GET /v2/entities/{entityID}/attrs/{attrName}/value
    - Retrieves an attribute's value
  - PUT /v2/entities/{entityID}/attrs/{attrName}/value
    - Updates an attribute's value

- **Query operations**

  - Tell me rooms on which temperature is less than 23 degrees and have at least 10 seats
    - **GET** /v2/entities?**type**=Room&**q**=temperature<23;seatNumber>=10
  - What are the vehicles currently at a radius of 10 kms with center Gangnam-Gu?
    - GET /v2/entities?type=Vehicle&geometry=point&coords=37.496667,127.0275&georel=near;maxDistance=10000
  - What are the Mercedes Vehicles currently at a radius of 10 kms with center Gangnam-Gu (Seoul)?
    - GET /v2/entities?type=Vehicle&coords=37.496667,127.0275&geometry=point;&georel=near;maxDistance=10000&q=manufacturer:'Mercedes Benz'
  - Tell me vehicle faults which happened today
    - GET /v2/entities?type=VehicleFault&q=startDate>=2015-07-17T00:00:00

- **Query Payload Examples**

```
[
        {
          "id": "123-456-789",
          "type": "Vehicle",
          "model": "C200",
          "brand": "Mercedes Benz",
          "buildYear": "2010"
        },
        {
        "id": "000-987-654",
          "type": "Vehicle",
          "model": "Astra",
          "brand": "Opel",
          "buildYear": "2003"
        }
]
```

  - Tell me the temperature at the Chrisantemum room
    - GET /v2/entities/r786543/attrs/temperature/value
      Result:
      23.5
  - Tell me the known car brands
    - GET /v2/entities?type=CarBrand&attrs=name&options=values
      Result:
      ["Ford", "Mercedes Benz", "Hyundai"]

- **Context Broker operations: create & pull data**

  - **Context Producers** publish data/context elements by invoking the **update** operations on a Context Broker.
  - **Context Consumers** can retrieve data/context elements by invoking the **query** operations on a Context Broker

**Data Connectors**

Data Connectors are custom solutions implemented as wrappers, in its simplest form, involves data retrieval from Orion Context Broker and other data sources, apply single or multi-step transformations if required and send/push it in the MiDAS. In case this data needs to be transformed or translated, a spark stream or similar technologies can be leveraged for stream processing. Each use case in SMART-FI platform has its own implementation of data connectors in different technology stacks.

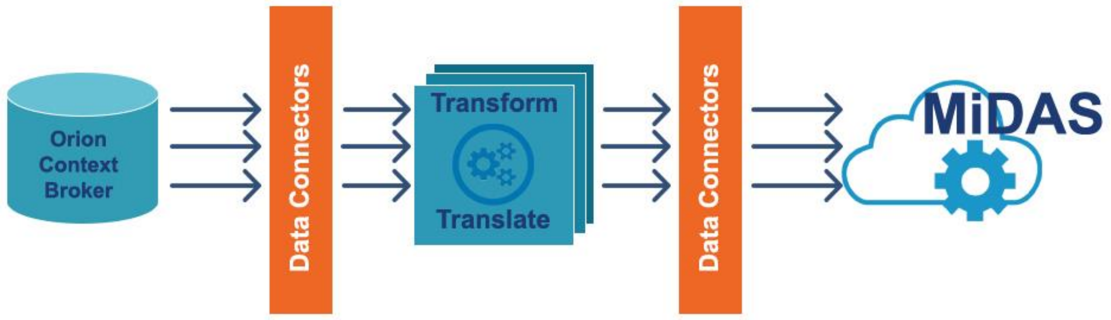

**Figure 7 Data Connectors**

**Data Formats**

As explained in Section 5 of deliverable 3.1 document.

# 5 Real Time Data Analytics Programming Model

## 5.1 MiDAS Osmotic flow

Although so far several smart Cities release their open-data as a static collection of data, it is clear that the rise of Edge computing pushes towards a more dynamic solutions regarding the Smart City platforms such as SMART-FI, where smart cities ever stronger share their data in real time as continuous open-streams.

To overcome some of the limitations of current approaches w.r.t. real time data analytics (Section 2), we devise *MiDAS Osmotic Flow*, a new solution for Cloud-supported Edge data analytics that enables to smoothly run data analytics applications on Edge and Cloud resources (i.e. Smart City platforms), which are seamlessly integrated following the Osmotic Computing principles [9]. To this end, Osmotic Flow comprises a novel stream analytics model which promotes Streams as First Class Citizens, and is designed to exploit the presence of scattered resources in a simple manner. The novel stream model enables simplicity, flexibility, and scalability of application development, hence supporting Smart City application designers.

The main motivation for our programming model is **efficient composition of data transformation functions**. This means the *MiDAS* Osmotic Flow model should support the composition of data transformations, so to easily realize complex applications. Differently from most of the existing approaches, the composition of transformation functions should try to minimize as much as possible the impact on the network. To this end, Osmotic Flow should consider by design the possibility of composing data streams coming from multiple, public IoT devices and applications, thus promoting the principle of sharing and reusability.

## 5.2 MiDAS programming model

We propose a new approach for realizing real-time data analytics applications that considers *stream as a first class citizen*[9]. Our stream model allows the user to easily define new streams, which extract high value information from raw data, without worrying about low level concerns related to their runtime execution, such as resource allocation, streams deployment, elasticity, and governance. These operations are managed by the underlying framework, which takes care of the application execution.

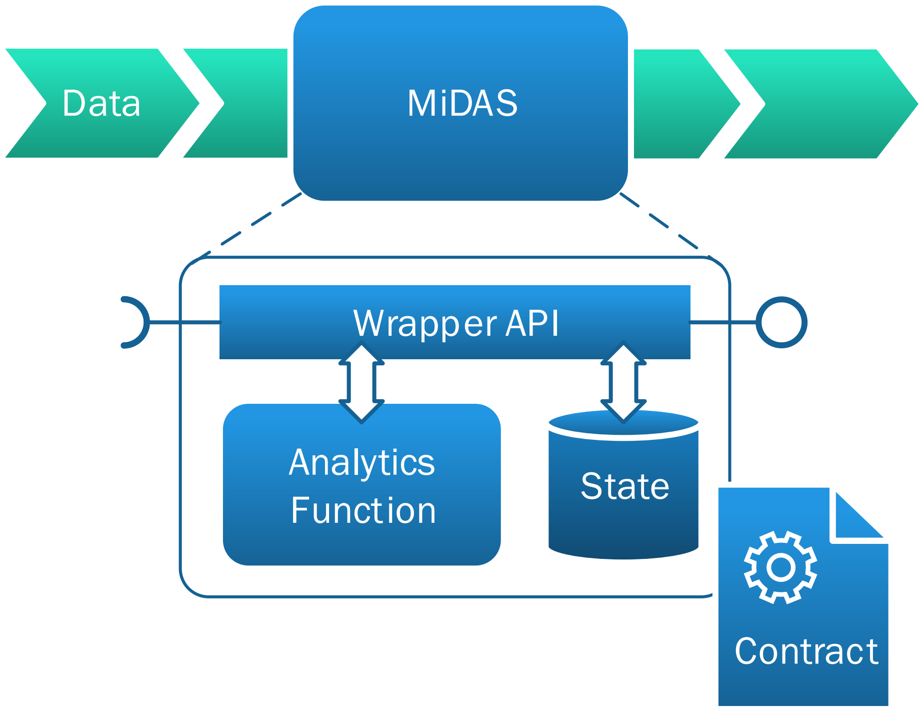

**Figure 8 Entities composing a stream in the Osmotic Flow model**

In MiDAS Osmotic Flow, as depicted in Figure 4, a stream is a complex entity that shapes and moves an unbounded sequence of data between two endpoints. Specifically, an input endpoint receives data from an external data source or from another stream, whereas an output endpoint is used to emit data towards sinks or other streams. A stream is characterized by the following elements:

- one or more data sources: a data source is an entity, potentially external to the system, that continuously generates events or data. For example, a data source can be an IoT device emitting temperature measurements or a service publishing notifications;
- one or more sinks: a sink is a final information consumer, that is interested in receiving high-level information extracted from raw data. For example, a sink can be a dashboard for the application stakeholders;
- a transformation function: it encapsulates the user-defined logic which manipulates (e.g., combines, filters, splits) incoming data so to produce new outgoing data. As such, it defines the high-level information carried by the stream. These data are transferred to other application components (i.e., sinks, streams) through the output endpoint of the stream;

[9] In this section we use terms Stream and „an instance of MiDAS“interchangeably.

- a contract: it is a high-level configuration descriptor of the stream that, on the one hand, enables the framework to automatically manage the stream execution and, on the other hand, allows the user to customize the stream runtime in a simple yet effective manner.

For the execution, the MiDAS runtime framework will need to encapsulate the stream in a self-contained entity, i.e., a MiDAS, which is then transparently managed and executed on the computing infrastructure (this is however subject of Task 4.2).

### 5.2.1 Transformation Function

Transformation functions are the only piece of code that has to be defined by the user, which encapsulate the data analytics logic. For an efficient execution, the stream model requires transformation functions to be as scalable as possible, therefore the latter are either stateless or provide an explicit definition of their state. We distinguish between two kind of transformations: simple and batch.

- A simple transformation is applied to every incoming data in parallel and produces zero, one, or more outgoing data. For example, a simple transformation realizes a streaming version of the map function of MapReduce: each incoming event is transformed in a key-value pair.
- A batch transformation allows to collect a group of data before applying a transformation, which can produce zero, one, or more outgoing data. The group of data fully determines the function state, which can then be manipulated by the transformation. A batch of data can be determined according to two modes: window and window-and-key. A window-based batch transformation creates a time-based or count-based window of events, that have to be collected before running the transformation on the entire batch. For example, a batch transformation can computes statistics on temperature measurements of the latest 30 seconds. A window-and-key batch transformation is a special case of windowed transformation that has a finer granularity in selecting the data composing the batch. A classic example for a window-and-key transformation is the implementation of the word counter, which computes some statistics on elements with the same key (i.e., word).

Simple transformations are stateless operations, whereas batch transformations provide an explicit definition of their internal state. This definition allows transformations to be side-effect free, i.e., they produce a deterministic outcome for a well-defined set of input data. This property enables the realization of highly scalable and distributable data analytics applications, because transformations can be managed with a very limited footprint.

It is worth observing that, although the stream model encourages the definition of lightweight transformation functions, it does not prevent the user from developing a sophisticated data analytics logic. Indeed, stateful transformations can be developed decoupling and externalizing the operation state from the transformation function by leveraging on external storage or caching systems.

### 5.2.2 Contract

The execution framework is responsible for executing the streams and, as such, it automatically performs management operations, such as elastically scaling streams so to handle varying workloads. Without limiting resources that can be acquired as needed, under a pay-per-use model, the user may incur in high expenses when multiple resources are needed for a suitable stream execution. Leveraging on the contract, the user can customize and limit the way management operations are performed. The contract provides a high-level description of the stream configuration. It is conceptually divided in sections, each of which focuses on specific aspects of the stream runtime. We identify the following sections:

- Placement: this section enables to customize the stream deployment over the distributed infrastructure. If no restrictions are provided, the execution framework can deploy a stream everywhere, thus the framework tries to maximize the utilization of Edge resources. Nevertheless, some streams might require different policies that, e.g., maximize the utilization of nearby resources or exploit centralized Cloud resources. These preferences might follow from non-functional requirements (e.g., execution cost) or from user-defined choices.
- Elasticity: at runtime, the execution framework elastically adapts the stream deployment so to guarantee the stream responsiveness in face of unexpected workload variations. Nevertheless, unexpected loads can lead to high execution costs, which might be acceptable or not according to the application scenario. This section of the contract enables to personalize the way the framework elastically scales the streams, e.g., by indicating scaling strategies, scaling limitations, and resource limitations.
- Governance rules: together with the previous sections, the governance rules enable to specify further restrictions regarding the stream deployment and adaptation. These restrictions are often related to security, privacy, or law concerns. For example, a governance rule can exclude every edge resource belonging to a specific geographical region or can require to encrypt the exchanged data, so to meet stringent law restriction.
- QoS requirements: this section expresses non-functional properties that the framework should meet during the stream execution, so to obtain a desired quality level. For example, requirements can restrict the maximum stream latency or minimum stream throughput.

### 5.2.3 Public and Ephemeral Streams

A stream can be ephemeral or public. An ephemeral stream is a special kind of stream that exists only if a sink is (directly or indirectly) interested to information coming from this stream. An indirect interest is manifest when one or more streams lie in between the stream and the final information consumer (i.e., sink). Being ephemeral, the existence of the stream depends on the presence of (direct or indirect) interested sinks and its scope is restricted within the same application, i.e., it can be used only by user-defined transformations running within the same application that contains the stream. A public stream is a globally available stream and, as such, can be used by any user. Examples are events generated by public data sources, such as weather or traffic monitoring stations. Public streams exists independently from the presence of interested consumers.

### 5.2.4 Streaming MiDAS Application

In the stream model, the transformation function is the only segment of code defined by the user. To enable its management, the transformation function is wrapped together with the contract so to create a MiDAS instance. It is then automatically executed by the underlying runtime framework.

In this model, a data analytics application results from the composition of multiple streams, which collaborate to extract valuable information from raw data, emitted by possibly distributed data sources. In an application, multiple streams can be arranged in pipelines or in parallel branches that, ultimately, reach final information consumers (i.e., sinks). A pipeline is a sequence of streams, where each stream builds on the output of the previous one, e.g., to filter or enhance the informative value of data. Parallel branches are exploited to concurrently perform multiple task on the same data (e.g., multiple aggregations of measurements). The user expresses the composition of streams by bonding together the endpoints of multiple streams. Then, the underlying framework deals with the coordination and the concurrent execution of the streams as well as the propagation of data between their transformation functions.

More details on the presented programming model can be found in our journal publication[10].

# 6 Conclusion

This document provides basic information on the current progress in Task 4.1 of WP4. It introduced Micro Data Analytics Services (MiDAS) model and MiDAS preliminary architecture. Presented MiDAS facilities represent an integral part of the SMART-Fi platform and provide models and components that allow for developing and managing value-added data analytic services in Smart Cities. In this deliverable we mainly focused on MiDAS' support for programming generic, elastic data analytic services, in order to facilitate analyzing the aggregated data for predictions and recommendations. Furthermore, we presented MiDAS real-time analytics programming model intended for online stream analytics. Among other things, our programming model enables efficient composition of data transformation functions. Hence it facilitates development of complex Smart City applications, e.g., for predictions and recommendations, by reliving the Smart City application developers from much of the currently faced development burden, such as explicate state management, operator placement and runtime concerns. Finally, we have outlined a coherent picture of MIDAS facilities in the context of larger SMART-FI platform, by elaborating on its main dependencies on other SMATRT-FI components, as well as its main outputs that underpin higher-level components of the SMART-FI platform.

[10] Matteo Nardelli, Stefan Nastic, Schahram Dustdar, Massimo Villari, and Rajiv Ranjan. "Osmotic Flow: Osmotic Computing+ IoT Workflow." IEEE Cloud Computing 4, no. 2 (2017): 68-75.

# 7 References

[1] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. Big data and cloud computing: Currentstate and future opportunities. In Proceedings of the 14th International Conference on Extending Database Technology, EDBT/ICDT '11, pages 530–533, New York, NY, USA, 2011. ACM.

[2] Alfredo Cuzzocrea, Il-Yeol Song, and Karen C. Davis. Analytics over large-scale multidimensional data: The big data revolution! In Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP, DOLAP '11, pages 101–104, New York, NY, USA, 2011. ACM.

[3] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of big data on cloud computing: Review and open research issues. Information Systems, 47:98 – 115, 2015.

[4] Domenico Talia. Clouds for scalable big data analytics. Computer, 46(5):98–101, May 2013.

[5] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of big data on cloud computing: Review and open research issues. Information Systems, 47:98 – 115, 2015.

[6] F. Zulkernine, P. Martin, Y. Zou, M. Bauer, F. Gwadry-Sridhar, and A. Aboulnaga. Towards cloud-based analytics-as-a-service (claaas) for big data analytics in the cloud. In 2013 IEEE International Congress on Big Data, pages 62–69, June 2013.

[7] Nathan Marz and JamesWarren. Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2015

[8] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. Storm@twitter. In Proceedings of the 2014 ACM SIGMOD

[9] M. Villari, M. Fazio, S. Dustdar, O. Rana and R. Ranjan, "Osmotic Computing: A New Paradigm for Edge/Cloud Integration," in IEEE Cloud Computing, vol. 3, no. 6, pp. 76-83, Nov.-Dec.2016. doi: 10.1109/MCC.2016.124

[10] FIWARE Catalogue:Data/Context Management
https://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker