Horizon 2020 Program (2014-2020)

Big data PPP

Research addressing main technology challenges of the data economy

Industrial-Driven Big Data as a Self-Service Solution

D2.4: Universal Messaging Bus (Interim Version)[†]

Abstract: This deliverable reports on the I-BiDaaS data ingestion and integration solution for collecting and distributing real-time streaming raw data (structured, unstructured, noisy and possibly incomplete).

Contractual Date of Delivery	30/06/2019
Actual Date of Delivery	30/06/2019
Deliverable Security Class	Public
Editor	*Dr. Gerald Ristow (SAG)*
Contributors	UNSPMF, IBM, BSC, ITML
Quality Assurance	*Omer Boehm (IBM)* *Ioannis Arapakis (TID)* *Kostas Lampropoulos (FORTH)*

[†] *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780787.*

The *I-BiDaaS* Consortium

Foundation for Research and Technology – Hellas (FORTH)	Coordinator	Greece
Barcelona Supercomputing Center (BSC)	Principal Contractor	Spain
IBM Israel – Science and Technology LTD (IBM)	Principal Contractor	Israel
Centro Ricerche FIAT (FCA/CRF)	Principal Contractor	Italy
Software AG (SAG)	Principal Contractor	Germany
Caixabank S.A. (CAIXA)	Principal Contractor	Spain
University of Manchester (UNIMAN)	Principal Contractor	United Kingdom
Ecole Nationale des Ponts et Chaussees (ENPC)	Principal Contractor	France
ATOS Spain S.A. (ATOS)	Principal Contractor	Spain
Aegis IT Research LTD (AEGIS)	Principal Contractor	United Kingdom
Information Technology for Market Leadership (ITML)	Principal Contractor	Greece
University of Novi Sad Faculty of Sciences (UNSPMF)	Principal Contractor	Serbia
Telefonica Investigation y Desarrollo S.A. (TID)	Principal Contractor	Spain

Document Revisions & Quality Assurance

Internal Reviewers

1. *Omer Boehm (IBM)*
2. *Ioannis Arapakis (TID)*
3. *Kostas Lampropoulos (FORTH)*

Revisions

Version	Date	By	Overview
0.9	30/06/2019	Dr. Gerald Ristow	Incorporated review comments from FORTH
0.8	14/06/2019	Dr. Gerald Ristow	Incorporated reviewers' comments and feedback, ready for final review
0.7	24/05/2019	Dr. Gerald Ristow	Document completed and ready for review by TID and IBM
0.6	22/05/2019	Dr. Gerald Ristow	Completed the introduction and most sections
0.5	22/05/2019	Omer Boehm	Added section on TDF as message publisher
0.4	21/05/2019	Dr. Gerald Ristow	Completed some of the SAG sections
0.3	17/05/2019	Dr. Gerald Ristow	Made changes to TOC according to reviewer's comments
0.2	22/04/2019	Dr. Gerald Ristow	ToC - Second draft.
0.1	12/03/2019	Dr. Gerald Ristow	ToC - First draft.

Table of Contents

List of Abbreviations

AMQP – Advanced Message Queuing Protocol

CEP – Complex Event Processing

CSP – Constraint Satisfaction Problem

DFP – Data Fabrication Platform

EPL – Event Processing Language

ESB – Enterprise Service Bus

ETL - Extract, Transform and Load

GPU – Graphics Processing Unit

JSON – JavaScript Object Notation

M2M - Machine to Machine

MoM – Message-oriented-Middleware

MQTT – Message Queuing Telemetry Transport

MVP – Minimum Viable Product

SAG – Software AG

TDF - Test Data Fabrication, formerly known as DFP

UM – Universal Messaging

UMEM - Universal Messaging Enterprise Manager

List of Figures

Executive Summary

This document describes the chosen end-to-end solution for collecting, aggregating, pre-processing and transforming real-time streaming raw data (structured, unstructured, noisy and possibly incomplete) into structured messages of a common, unified, format.

The main component used for this is a message broker which uses a publish-subscribe mechanism for distributing the data to all components that need them.

The data exchange uses open standards like MQTT [2] to allow for a flexible and easily extensible component architecture.

In the MVP, delivered with M12, we have shown that this approach works well and has the desired flexibility. IBM's TDF tool can send data to the message broker which can then be processed either by the batch or the streaming layer or by both of the I-BiDaaS platform. The data from one source can also be combined with data from other sources in order to lead to a better insight of the data.

We also sketch how the pre-processing can be done and that the solution is scalable.

1 Introduction

The I-BiDaaS platform consists of many different components that need to communicate with each other. A common way to do this is to exchange structured messages between the components via a publish-subscribe mechanism using a message broker leading to a message-oriented middleware (MoM) solution [3]. This approach is sometimes called Enterprise Service Bus (ESB). This can be achieved by many software solutions, either open source or commercial. They mostly differ in the number of message-passing standards they support and if data persistence or high-availability or failover support is needed.

For the I-BiDaaS platform, we are using the MQTT protocol to exchange messages which is an ISO standard. This guarantees that new components can be connected easily to the platform to read or write data to it.

For a Big Data platform, it is essential that the platform and in our case, especially the message broker can handle large volumes and high velocity inputs which is achieved by a scalable architecture.

The data formats and the data fabrication process is described in other deliverables of WP2. The visualization component is also described there, especially in D2.3. The batch processing that will analyse and process the historical data is detailed in WP3. WP4 described the real time complex event processing engine which uses the analysed historical data as input and which can use GPUs for fast and efficient analysis.

2 Message-oriented-Middleware Solutions

2.1 Universal Messaging as chosen MoM Solution for I-BiDaaS

For this project, we have chosen the commercial product Universal Messaging by SAG [4] since it supports many messaging standards like AMQP, JMS and MQTT. It also has interfaces to many programming languages like Java and C. In addition, it supports many features suitable for enterprises like clustering, scalability and durable subscriptions to name just a few.

At the network level Universal Messaging will run on an TCP/IP enabled network supporting normal TCP/IP based sockets, SSL enabled TCP/IP sockets, HTTP and HTTPS, providing multiple communications protocols.

One commonly used protocol for messaging is MQTT which we will use in the project.

2.2 Possible Open Source Alternatives

Even though UM provides many enterprise features that makes it a stable and scalable product, it might be desirable for various reasons to consider alternatives for UM. Especially if the costs for the software stack become an issue, open source alternatives may be preferable. In order to be usable in the I-BiDaaS platform, they need to support the MQTT standard. Some indicative examples are

1. Apache ActiveMQ [5]
2. Mosquitto [6]
3. RabbitMQ with the MQTT plug-in [7]

3 Role of UM in the I-BiDaaS Architecture

The role of the Universal Messaging (UM) component becomes clear, when looking at the overall I-BiDaaS architecture shown in **Figure 1**. UM is shown in the left part of the figure and mainly serves as a data ingestion component for the I-BiDaaS platform.

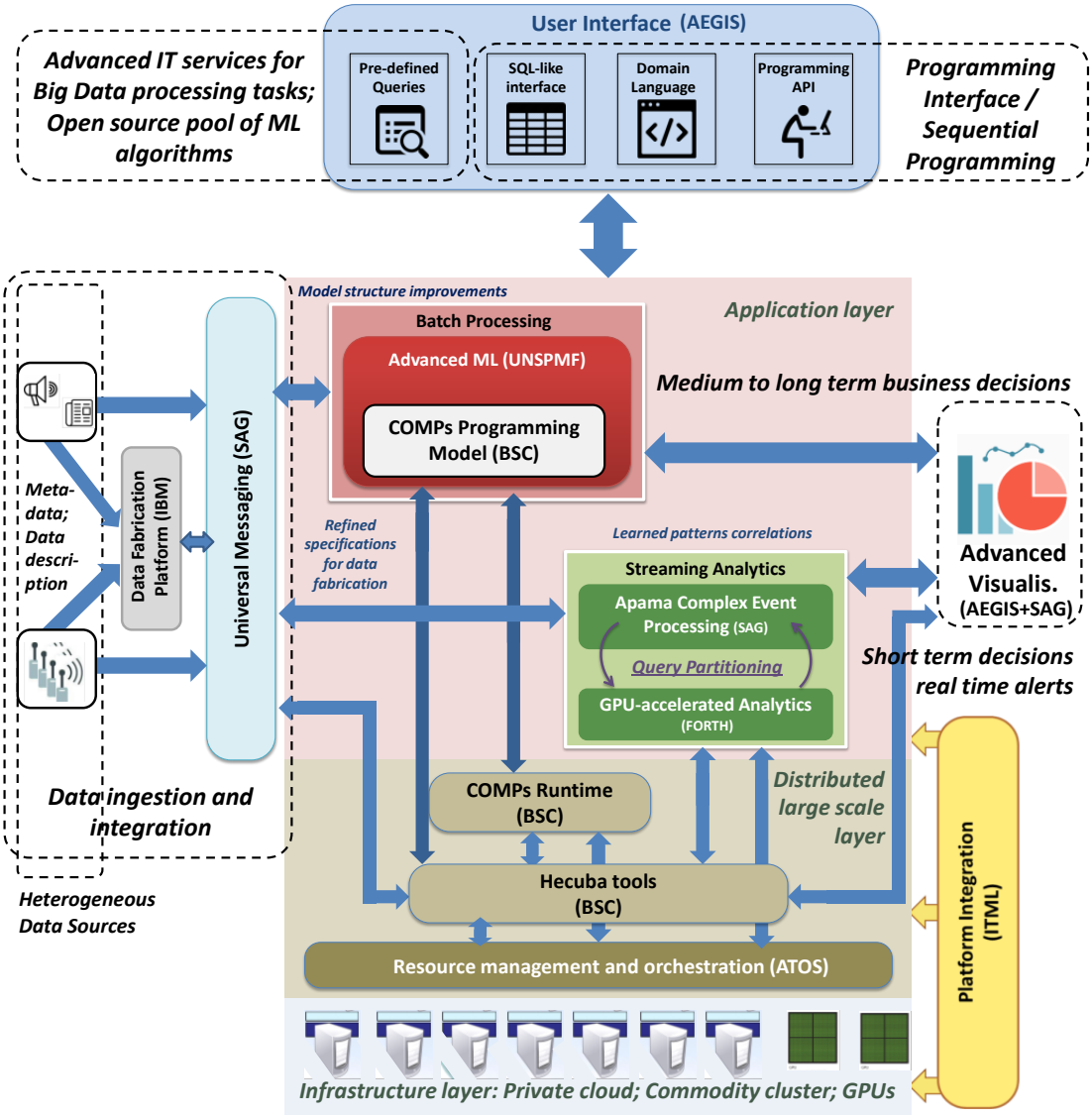

Figure 1: I-BiDaaS Platform

The messages published to UM consist of metadata and a message body, the latter can be in any format, however, for this project we favour JSON [1] so that the messages can be read easily by humans and the corresponding key-value pairs can be identified easily. It also helps in finding inconsistencies in the communication channel.

The Hecuba tools also connect to UM in order to persist the data that is needed for historical data analysis in the central data store, a distributed Cassandra DB.

3.1 Message Standards

UM supports many industry standards on message-passing, e.g.

- MQTT, an ISO standard used for machine to machine (M2M) communication
- Java Message Service (JMS)
- AMQP v1.x

For the I-BiDaaS project, we have chosen the MQTT protocol as a communication standard. The reason is that MQTT is widely used, a rather light protocol and supported by many message brokers. It is preferred over other protocols and traditional client/server exchanges due to its minimal packet overhead and that it does not require standard formatting for data such as temperatures or time stamps. In addition, one can find many examples for the different programming languages [8] which facilitate an easier interaction and integration with the other components.

3.2 Message Publishers

MQTT applications can publish events to channels. If the specified channels do not already exist in the Universal Messaging realm, they will be automatically created by the server as MIXED type with a JMS engine.

3.2.1 TDF

The TDF tool supports the generation of data into databases and into various standard file formats. For databases, the tool generates a CSP problem for each table row (or 'wide row' in case where referential integrity constraints exist), where for files, the tool generates a single CSP problem for the entire file. The tool can import the data structure from the database metadata or, in the case of files, enable the user to define the structure of the data manually using predefined, C-like, elements. In addition to the structure, modelling of the data and the relationships in it using various types of constraint rules is required. Both the data structure description and the model are translated into an abstract constraint satisfaction problem (CSP) and sent to the integrated CSP solver. The solver returns a solution, which contains a desired data point that satisfies all the constraints. This solution is then translated and written into the desired output database tables or files.

TDF was extended to support streaming data output using the MQTT protocol mentioned above. The CSP solution provided by the solver is converted from the TDF internal representation into the JSON format and sent according to the provided configurations options, i.e., broker, port, topic, quality of service, authentication method, secure communications, etc.

The format that was selected for databases is the JSON tabular data, which is essentially a JSON array of arrays where the first array contains the columns names and the remaining arrays contain the table rows.

The format that was selected for the file projects was to recursively iterate the file hierarchy, and convert each C-Like element into the proper JSON object.

3.2.2 Others

Publishing messages using the MQTT protocol can be done from many different sources and systems since helper libraries are available in nearly any programming language. One method

is to use a trigger, e.g., for Facebook or Twitter that will scan for posts or tweets of interest and if they occur, publish them to a well-defined UM channel.

Normally, one would use different channels for different data sources so that the origin is clear and subscribers can better pick the information they want.

Since MQTT is an ISO standard for M2M communication, many sensors support it directly so that the desired information, e.g., temperature or pressure can be sent directly from the sensor to UM. This is done in many Industry 4.0 projects and scenarios and can be used to enrich the date from other sources.

3.3 Message Subscribers

MQTT applications can subscribe to channels. If the specified channels do not already exist in the Universal Messaging realm, they will be automatically created by the server as MIXED type with a JMS engine.

3.3.1 Hecuba

The Hecuba tools used in the I-BiDaaS project use mainly Python for their tasks. For Python, MQTT libraries are available as well; one example is the widely used Paho implementation [9]. Even though the use cases considered for the MVP did not require the Hecuba tools to be connected to UM, we do not expect this to be a problem when using the Paho library.

3.3.2 Apama

Apama is a complex event processing engine that is used in the I-BiDaaS platform for streaming analytics purposes. More details about the architecture and the technical background are given in D4.1. In there, the interaction of Apama with UM is also described and an example from the MVP is given. In general, Apama used the so-called connectivity plug-ins to connect to external components. For UM, one can use the proprietary UM plug-in or the MQTT plug-in. The latter can also be used to connect to other MQTT message brokers like Mosquitto [6] and ActiveMQ [5].

The data taken from UM can be passed on to GPUs if the specific streaming analytics use case requires this.

3.3.3 Others

Subscribing to messages on UM using the MQTT protocol can be done from many different sources and systems since helper libraries are available in nearly any programming language. UM keeps track of the current subscribers to a specific message channel and if a new message arrives, it is automatically sent to all subscribers. If a subscriber is offline or not reachable without a proper subscription cancelation from UM, UM will retain the message till the message is sent to all known subscribers.

3.4 Message Formats and Payloads

Events published via MQTT only contain a byte payload and are tagged MQTT. They are fully interoperable with any UM subscriber on any client platform supported and can be snooped using the UMEM.

3.5 The MVP as a concrete Use Case

We have verified that data exchange via MQTT messages works in the MVP. This is described in detail in sections 3.2 and 3.3 in D5.2 which was submitted in M12.

3.6 Possibilities of Data Processing and Pre-Processing

Since UM is a passive component, we need other components that send data to UM, called publishing, or that read data from UM, called subscribing. One way to do data processing and pre-processing is to use a component that can read from UM the MQTT messages. A widely used product is Nifi [10] which has a nice user interface and which could be made to fit nicely in the I-BiDaaS infrastructure. The data can be processed or transformed and then put back onto UM for the other I-BiDaaS components to use them. One can also use Nifi to process the data and place it into Cassandra DB since Nifi has adapters to read from and write to Cassandra DB and other databases.

3.7 Scalability and Load Balancing

Scalability in terms of messaging middleware typically means *connection scalability*, which is the ability to support large numbers of concurrent connections; this is something UM does out of the box. However, in defining truly global enterprise applications, a single system often needs to scale across more than one processing core, often in more than one geographic location.

UM servers can be configured in a variety of ways to suit scalability (and resilience) requirements. Multiple Universal Messaging servers can exist in a single federated name space. This means that although specific resources can be put onto specific UM realm servers, any number of resources can be managed and access centrally from a single entry point into UM's federated namespace. In addition to high availability and resilience features, UM clusters also offer a convenient way to replicate data and resources among a number of realm servers.

4 Conclusions and Outlook

This document described the chosen end-to-end solution for collecting, aggregating, pre-processing and transforming real-time streaming raw data (structured, unstructured, noisy and possibly incomplete) into structured messages of a common, unified, format.

It consists of a message broker, where we have chosen SAG's Universal Messaging product and an ETL tool, where we have chosen Apache Nifi. In the message broker, we use the ISO standard MQTT for the messages, which is supported by many tools, including Nifi. Nifi has many adapters for different systems and standards so that is a very flexible tool for doing the needed processing, pre-processing and transformation steps for the different use cases.

Some of the proposed components were not yet fully tested in the I-BiDaaS platform. The full analysis will be given in D2.6 (Universal Messaging - final version), due M30.

5 References

[1] See https://json.org/
[2] See mqtt.org/
[3] See https://docs.oracle.com/cd/E19340-01/820-6424/aeraq/index.html
[4] See https://www.softwareag.com/corporate/products/az/universal_messaging/
[5] See https://activemq.apache.org/
[6] See http://mosquitto.org/
[7] See https://www.rabbitmq.com/
[8] See http://www.steves-internet-guide.com/mqtt/
[9] See https://pypi.org/project/paho-mqtt/
[10] See https://nifi.apache.org/