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

D2.6: Universal Messaging Bus (Final 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). It focusses on data pre-processing and securing Universal Messaging.

Contractual Date of Delivery	30/06/2020
Actual Date of Delivery	30/06/2020
Deliverable Security Class	Public
Editor	*Dr. Gerald Ristow (SAG)*
Contributors	IBM, BSC, CAIXA, CRF, TID, FORTH
Quality Assurance	*Omer Boehm (IBM)* *Dr. Ioannis Arapakis (TID)* *Dr. 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. *Dr. Ioannis Arapakis (TID)*
3. *Dr. Kostas Lampropoulos (FORTH)*

Revisions

Version	Date	By	Overview
1.6	29/06/2020	Dr. Gerald Ristow	Incorporated feedback and suggestions from FORTH, ready for submission to the PO
1.5	18/06/2020	Dr. Gerald Ristow	Figure 3 was updated by TID. Ready to be sent into final review.
1.4	10/06/2020	Dr. Gerald Ristow	Incorporated and replied to the suggestions from IBM
1.3	10/06/2020	Dr. Gerald Ristow	Incorporated and replied to the suggestions from TID
1.2	02/06/2020	Dr. Gerald Ristow	Ready for review
1.1	29/05/2020	Ioannis Arapakis	Completed section 2.5 (TID Use Cases)
1.0	28/05/2020	Jordi Luque Serrano	Added CC Use Case to section 2.5 (TID Use Cases)
0.9	27/05/2020	Cesare Cugnasco	Added section 2.3 (BSC batch tools)
0.8	27/05/2020	Dr. Gerald Ristow	Added the summary and the first part of section 2
0.7	26/05/2020	Ramon Martín de Pozuelo	Added section 2.3 (CAIXA Use Case)
0.6	26/05/2020	Dr. Gerald Ristow	Added section 3 (Nifi)
0.5	24/05/2020	Danilo Spennacchio	Added section 2.4 (CRF Use Case)
0.4	22/05/2020	Dr. Gerald Ristow	Added section 4 (UM Configuration)
0.3	18/05/2020	Omer Boehm	Added section 2.1 (IBM TDF)
0.2	04/05/2020	Dr. Gerald Ristow	Adopted TOC to the feedback from UNSPMF
0.1	14/04/2020	Dr. Gerald Ristow	ToC - First draft.

Table of Contents

List of Abbreviations

CC – Call Center

CSP – Constraint Satisfaction Problem

DFP – Data Fabrication Platform

ETL - Extract, Transform and Load

GPU – Graphics Processing Unit

JSON – JavaScript Object Notation

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 how the message broker Universal Messaging is used in the use cases of the data providers where the details are given in other deliverables. It starts with the necessary data ingestion and batch processing part and then explains the UM usage in the use cases in detail. In addition, it shows how data pre-processing can be done by using Apache Nifi to read, enhance and write back messages that are published to UM by other clients. Since some of the data from the data providers contain sensitive information and needs to be protected, we discuss in detail how UM can be made more secure and how access can be restricted.

1 Introduction

The software component Universal Messaging (UM) plays a central role in the I-BiDaaS platform as a flexible and reliable source of data ingestion. Since it supports many industry standards (see D2.4 for a short list or [1] for more details), it is rather simple to publish messages or subscribe to them. How UM interacts with the other I-BiDaaS components is described at the beginning of section 2.

The first step we took in the project was to make IBM's TDF [2] tool write directly to UM instead of going via a database or the file system. This allows for real-time use cases, which become more common in many industrial applications. Details are given in section 2.1.

We used UM in many of the use cases of the data providers. For the CAIXA use cases, this is discussed in section 2.3, for the CRF use cases in section 2.4 and for the TID use cases in section 2.5, respectively.

Since data can come in different flavours, one often needs to transform the data before it becomes usable. This can be done with many tools and we favour Apache Nifi [7] and describe a simple use case that uses UM in section 3.

Since UM is a central component in the I-BiDaaS platform, which is used by many other components and different partners, we need to protect and regulate its usage. This is described in detail in section 4, where we first show how the whole UM realm can be disabled for anonymous access and then go into details on how each channel can be protected. In the final section 4.4, we show how UM can be configured to use only encrypted communication, either with or without client verification.

2 Role of UM in the I-BiDaaS Architecture

The role of the UM component [1] 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.

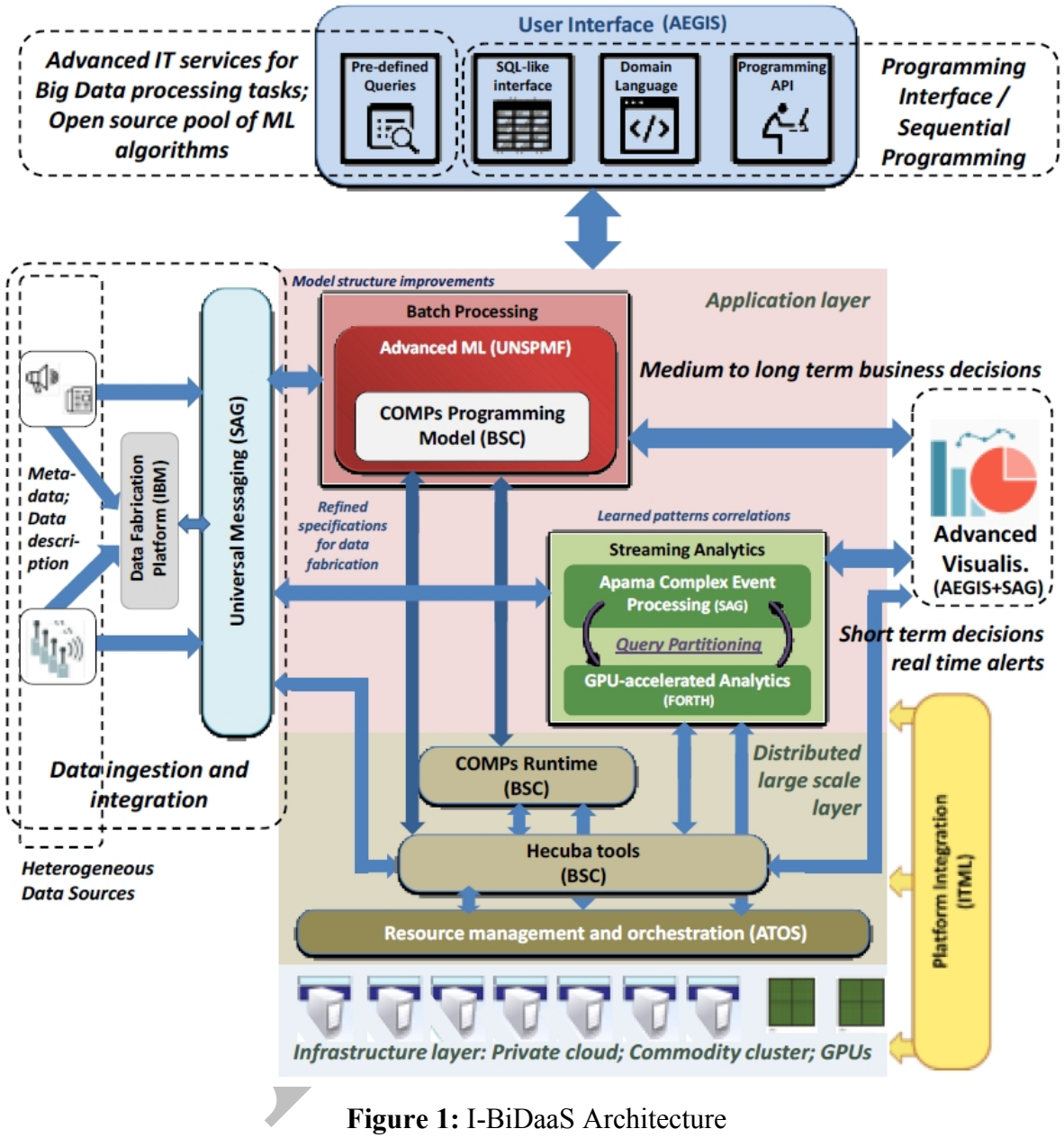

Figure 1: I-BiDaaS Architecture

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 [3] 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 messages are sent using the OASIS MQTT standard [4].

For many of the I-BiDaaS use cases, we have used the IBM Test Data Fabrication [2] to generate synthetic data and ingest it into the platform for further processing. This is shown in

Figure 1 to the left by the box labelled Data Fabrication Platform (IBM). The next section, 2.1, will give more details on the software and its operation.

The Hecuba tools [5] also connect to UM via Python in order to persist the data that is needed for historical data analysis in the central data store, a distributed Cassandra DB [6]. Details are given in subsection 2.2.

The last three subsections of this section briefly summarize how UM is used in the data provider use cases.

2.1 Data ingestion by the IBM Test Data Fabrication Platform

The IBM Test Data Fabrication (see D2.5) can be used to generate synthetic data using user defined constraints and data inferred constraints (see D3.2, section 4.2.3). Fabricated data can be written into standard relational databases, as well as into various standard file formats. TDF employs a CSP solver to solve the problem generated by the provided constraints described above. TDF can generate a CSP per each table row (or 'wide row' in case where referential integrity constraints exist), which is a more scalable option, or a single CSP to generate the entire data solution at once, which enables satisfying global constraints, e.g. inter row dependencies ('sum Of', 'numOf'), distributions, uniqueness etc.

Users can configure TDF to stream the fabricated data using the MQTT protocol (see D2.4), in addition to writing the data into a database or a file (for persistency). TDF creates a JSON message and sends it to the preconfigured broker (See D2.4, section 3.2.1). The integration of the universal messaging bus and TDF is described in D2.5, section 5.1.

2.2 Data consumption by the Hecuba batch tools

Many applications require to integrate both the streaming and batch layer. In these cases, data enters the system in the form of events published into a queue system like the UM. Each event/message is then consumed from both the streaming layer and the batch one. In such a scenario, the incoming data is organized as a sequence of messages that follow a pre-agreed format. These messages can either be generated from IBM's TDF or produced by any third-party data source, such as IoT sensors, transactions, user activity logs, etc. Subsequentially, both the streaming and batch layer reads the incoming information and put it at use. For instance, from one side, the streaming could apply on real-time an ML model, while on the other hand, the batch layer could use the historical data to increase the accuracy of the model over time. Using as example CAIXA's *Enhanced control on online banking* use case, every message contains information about money transfer requests. In this case, the batch layer stores all the history of each transaction and uses it to update the fraud prediction model. Once the model is improved, the batch model sends the updated parameters of the model via the UM to the streaming layer, which uses such information to mark suspicious transactions.

The Hecuba MQTT provides simple templates that simplify the development of the code required to set up a process that is listening to the UM, that parse and validates the incoming messages and consequentially store them into Cassandra. These templates propose good practises about how to define schema and format of messages, and how to store them into Cassandra. Following these templates, developers can integrate the streaming and batch module in a few minutes, using the JSON standard [3] to define the message format and the Hecuba interface to determine the persistent schema.

For more information about the Hecuba MQTT module, please refer to the deliverable D4.3, chapter 2.

2.3 Use of UM in the CAIXA use cases

CAIXA proposed three different use cases to be tested in I-BiDaaS:

1. Analysis of relationships through IP addresses
2. Advanced analysis of bank transfer payment in financial terminal
3. Enhanced control on online banking

"Analysis of relationships through IP addresses" use case was used to test the I-BiDaaS MVP, which was divided into two subcases: batch & stream processes. First, a batch process is applied in order to find relationships between customers (taking into account that they connect from the same IP addresses to CAIXA online banking). It results in a dataset of customer relationships. After that, UM is used for deploying the stream processing subcase, in which bank transfers between customers are analyzed. In this second part of the use case, a real-time analysis of those bank transfers is done, looking for transfers between senders and receivers that are already related in the previous batch processing analysis in order to find those transfers between a sender and a receiver. The whole deployment of this use case was already used and tested in the MVP.

For the second and third use cases of CAIXA, stream processing analysis is considered in order to analyze new coming entries (bank transfers from the bank offices in the "Advanced analysis of bank transfer payment in financial terminal" use case, and mobile-to-mobile bank transfers in "Enhanced control on online banking" use case) aiming at scoring in real-time the anomaly grade or the level of risk of those transactions in real-time, taking into account the clustering and anomaly detection models previously created in a batch processing analysis.

2.4 Use of UM in the CRF use cases

CRF selected two use cases, namely *Production process of aluminium die-casting* and *Maintenance and monitoring of production assets* and provided Big Data to I-BiDaaS technologists and analysts in order to improve the quality of the manufacturing processes and do predictive maintenance. Data retrieved from the manufacturing industry are often unstructured, noisy and incomplete. Therefore, they must be collected, aggregated, pre-processed and transformed into structured messages of a common, unified format for the analyses.

SAG's Universal Messaging has been used as a message broker that supports the MQTT protocol (see D5.1) for Data Ingestion and Integration, as shown in Figure 1. Synthetic and real Data, provided by CRF, are modelled with the IBM's Test Data Fabrication platform (see D2.5). After pre-processing, data are ingested in the system through UM and sent to the advanced Machine Learning model and COMPSSs model (see D3.1 and D3.2) in the Batch Processing Layer for batch analytics and also to Apama (see D4.1 and D4.2) for stream analytics details. Analytics results, as well as additional metadata, flow through the UM and are used for assisting the automation process of the data modeling (see D2.2 and D3.3). Through the UM, the module delivers to the data fabrication platform analytics results and meta-data that are used for semi-automatic and automatic data fabrication (D3.1).

The description of the configuration and monitoring of the UM servers for the *Production process of aluminium die-casting* use case is given in D5.4.

For the testing phase, data will be extracted from the CRF server and will be used to train a Random Forest model from a Python program. Once the model was trained, sample data from the CRF server was sent to the Universal Messaging, picked up by Apama to be passed on to

the pre-trained model for classification. This simulates a real-time event processing where the machine data is analysed before it is stored in the CRF server (see D 4.2).

The results are made available to the end user via the AEGIS's Advanced Visualization module (see D4.1).

2.5 Use of UM in the TID use cases

The TID use cases consist of three main cornerstones for any telco company:

- Quality of service in Call Centers
- Accurate location prediction with high traffic and visibility
- Optimization of placement of telecommunication equipment

In all three uses cases, anonymised Big Data has been provided to I-BiDaaS technologists and analysts in order to assess the quality of the call center services, e.g. by providing transcripts of customer service phone calls or aggregated and anonymised mobility and antenna logs, in order to perform predictions on user movements.

2.5.1 Quality of service in call centers

With respect to the case of CC analytics, we have defined and revised, with the involved consortium partners, the necessary operational experiments to carry out. These include the analysis of 6 months of call center interactions performed by customers of the company, as well as the computation of an aggregated sentiment score by both time window and set of call centers. This analysis was performed in a batchmode manner, aiming at highlighting specifically those word tokens that appear the most along the time window and significativally contribute to the sentiment scoring. The experiments include, but are not limited to, the following technical developments:

- Storage of automatic speech transcripts and phone call metadata in a Terracotta Database.
- Developing of a sentiment analysis tool based in natural language models in Spanish, by FORTH (D3.3).
- Adaptation and computation in GPU hardware of a positive/negative score and aggregation in time – at different time scales – and location (D3.3).
- Integration with SAG's UM (see D5.1), for Data Ingestion and Integration as shown in Figure 1.
- Integration of the results, for the end user, via the AEGIS's Advanced Visualization module (see D4.1).

We note that previous experiments are still on-going and are carried out in TID premises, due to the nature and privacy risks involved. To this end, the necessary access and hardware have been provisioned to I-BiDaaS technologists and analysts. Furthermore, to ensure GPDR compliance, several pre-processing steps are applied. It aims to obfuscate, by token replacement, personal or product information found in the automatic speech transcriptions for both customer and agent representative, together with company products and processes. Some of the anonymization steps performed by TID (D2.2) include:

- Replacement of personal names, places and locations by corresponding general tag, e.g. <name>.

- Switching of real-time timestamps of the call interaction, as stored by TID recorders, between phone calls.
- Switching of the ordering of word token sequences within the same phone call and removal of speaker (customer/agent) information.

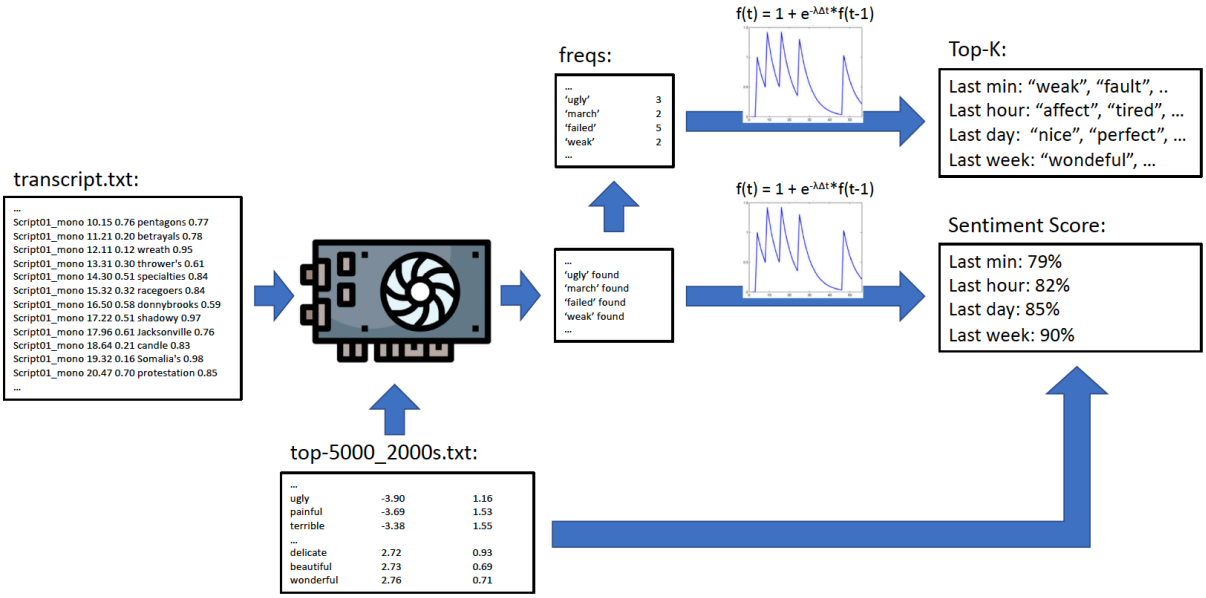

Figure 2: Example of CC transcripts flow within the FORTH sentiment analysis GPU

The ASR transcripts are obtained by one of TID prototypes (D2.2) and stored in anonymised ctm format in a Terracota Data Base for further processing by the FORTH's sentiment tool (D3.3). The final sentiment score is obtained by aggregation of calls in a time window, which values can be set to one day, one hour or one minute. Those calls falling in the same time window are aggregated together and contribute to the final sentiment score for the corresponding time window. The SAG's UM is used to collect sentiment analysis results and carry them out to the visualization tool. Sentiment analytics results flow through the UM and together with necessary call metadata (D2.2 and D3.2) are delivered to the AEGIS's Advanced Visualization module (D2.3). This previous step aims to simulate a real-time event processing in which the call center interactions data is analysed before it is finally stored in the TID servers. A sample execution is shown in Figure 3. In the top left window, we show the output of the sentiment analysis tool with sentiment score by word (English version), e.g. for the words beautifully, annoy and awkward. The top right window depicts the status of the TerracottaDB ingestion. The bottom left window shows the status of GPU processing and sending the updates to UM. The bottom right window gives some code snippets from the sentiment score I-BiDaaS tool.

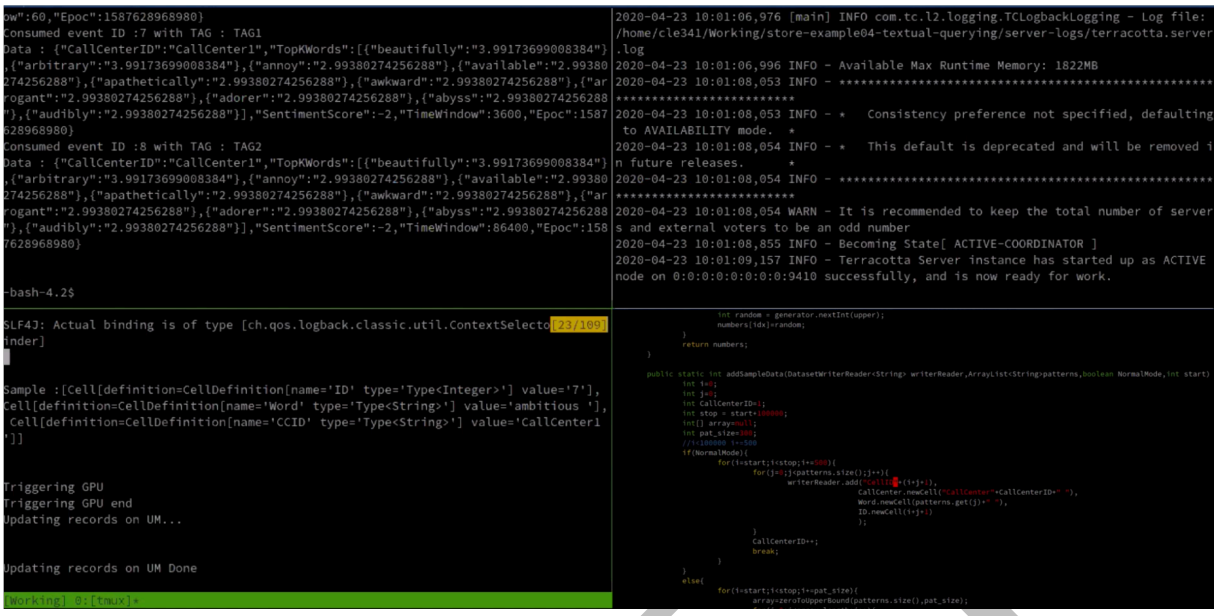

Figure 3: Example execution in TID server of the sentiment analysis tool.

Final benchmarks of the CC analysis tool solution include the throughput measured as the number of phone calls analysed per time unit and the retrieval of those specific word terms that contribute the most to sentiment score within a time window in order to improve CC operations by the real-time analytics insight. The former is a specific KPI defined in D3.1.

As mentioned above, the final results are made available to the end user via the AEGIS's Advanced Visualization module (see D4.1). It includes the aggregation of the scores within three "virtual" call centers and by time window. Visualization tool allows to expand sentiment score information by popping up a list of the most common token words per call center and time window, that is, providing and easy tool for end-user inspection and with valuable insights about real-time operations of the CC.

2.5.2 Accurate Location Prediction with High Traffic and Visibility

One of the aims in all networks is the optimization of the network operations by providing caches and optimal antenna locations. While some optimization is already in place, the requirement is to further analyze the data and provide new insights into how the network could be further optimized, given the provided data from customer usage. Based on the data retrieved by the customers, their usage, and the location, we have designed predictive models and algorithms that can support the decision-making process regarding routing and placement of telecommunication equipment. In other words, we aim to test the I-BiDaaS solution efficiency with respect to the prediction of places with high traffic and congestion events in order to optimise their resource distribution.

Two important tasks that stem from these challenges are to: 1) interpolate missing events to recover plausible event trajectories and 2) forecast immediately next events to anticipate movements at scale. To this end, we have applied the following experimental protocol:

- Data selection
- Data preparation
    - Aggregation of antenna KPI data from a major European telecommunications company, covering a city and a large number of cell sites
- Data analysis

    - Experimentation with various ML models (SVN, RF, XGBoost) and DL models to predict changes in the number of connected mobile phone users per sector
- Data visualization

Regarding the data selection process, we have created a dataset of mobility data. The data was collected on a 4-hour window (i.e. 6 samples per 24 hours window) over 30 days and consists of anonymous traces collected from a large European cellular network provider with tens of millions of subscribers (cross-sectorial). More specifically, it consists of 21M traces by 1.2M people usage statistics, from 40k sectors. Each such trace is a time series of mobile events and these events contain the encrypted user identifier, a timestamp, and the location of the associated base station used to deliver service to the user. The base stations have varied coverage (between ~100 m to tens of km) depending on deployment density and radio propagation characteristics like obstacles, hills, or mountains. The expected user displacement in urban areas is smaller than in rural areas and can reach as low as 70 m.

The end goal is to predict movement from one sector to another by estimating the delta (%) in the connected users per sector *N* hours in advance.

The developed ML models are provided by UNSPMF as Python programs both for training and inference purposes. It is important to note how these models communicate with other parts of the platform, and especially with the UM component. Python scripts used for training and inference accept a number of parameters either through input files (in CSV, JSON or other non-binary format) or through arguments. In the second case, the robust and versatile Python module "argparse" is used. Regarding output results, they are given in JSON format either to standard output of the process or a specific file. The control of running the models, passing the parameters and collecting the results is given to the I-BiDaaS platform, more specifically to the container management system (Docker). With streaming use cases where quick responses are important, it is also possible to provide inference scripts that load the models into memory and expose them as RESTful Web Services.

3 Data pre-processing with Apache NiFi

Not all data is already available in a format so that it can be processed automatically by the next tools in the processing chain. One needs to transform or enrich the data so that it can be further processed. A tool with a user-friendly web-based user interface, which can do such tasks easily, is Apache Nifi [7]. On Linux and Mac OS X systems, Nifi can be configured as a service to start automatically. Once started, the user designs dataflows in the UI by placing and connecting so-called Processors on the canvas. Many types of Processors are already available, e.g. to read and write to different types of databases and message brokers. Nifi can also be used as ETL tool for loading data into the I-BiDaaS platform.

Let us consider the use case depicted in Figure 4. As in many I-BiDaaS use cases, data is published to the message broker UM from the data sources, e.g. a JSON object given a specific *firstname*. Nifi will subscribe to the corresponding channel, pick up the data and add a *lastname* to the message and publishes it back to UM but to another channel.

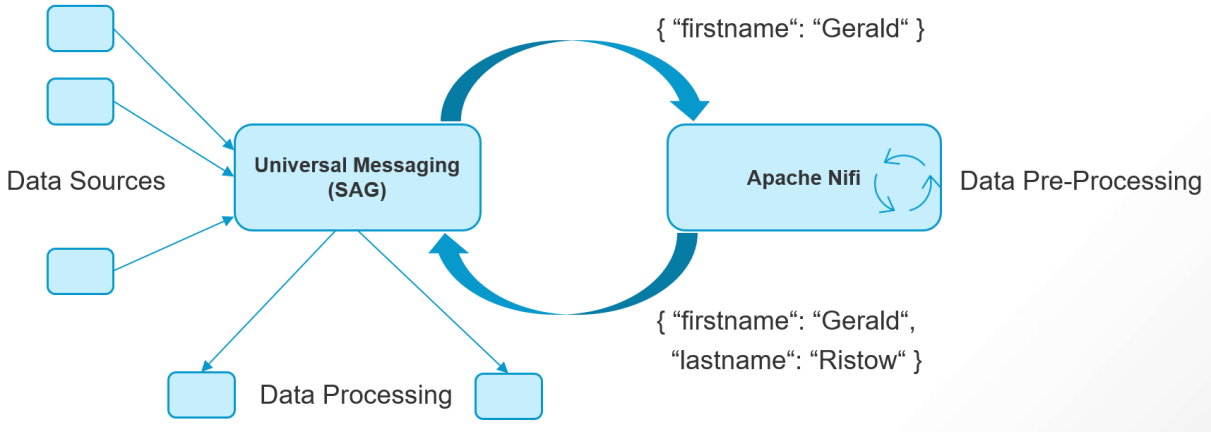

Figure 4: Simple Nifi Use Case

Further processing can be done easily by subscribing to the corresponding channel, either the original one to get to the shorter JSON object or the channel with the pre-processed and enriched JSON message, which was done by Nifi.

The whole dataflow of how it looks in the Nifi UI is shown in Figure 5. The main task consists of just three Processors:

1) ConsumeMQTT – which reads the message from a specific channel from UM
2) ReplaceText – which adds the *lastname* to the message
3) PublishMQTT – which writes back the modified message to another UM channel

We have added the two Processors PutFile and LogMessage in order to save a copy of the enriched message to a file and to allow for logging, respectively.

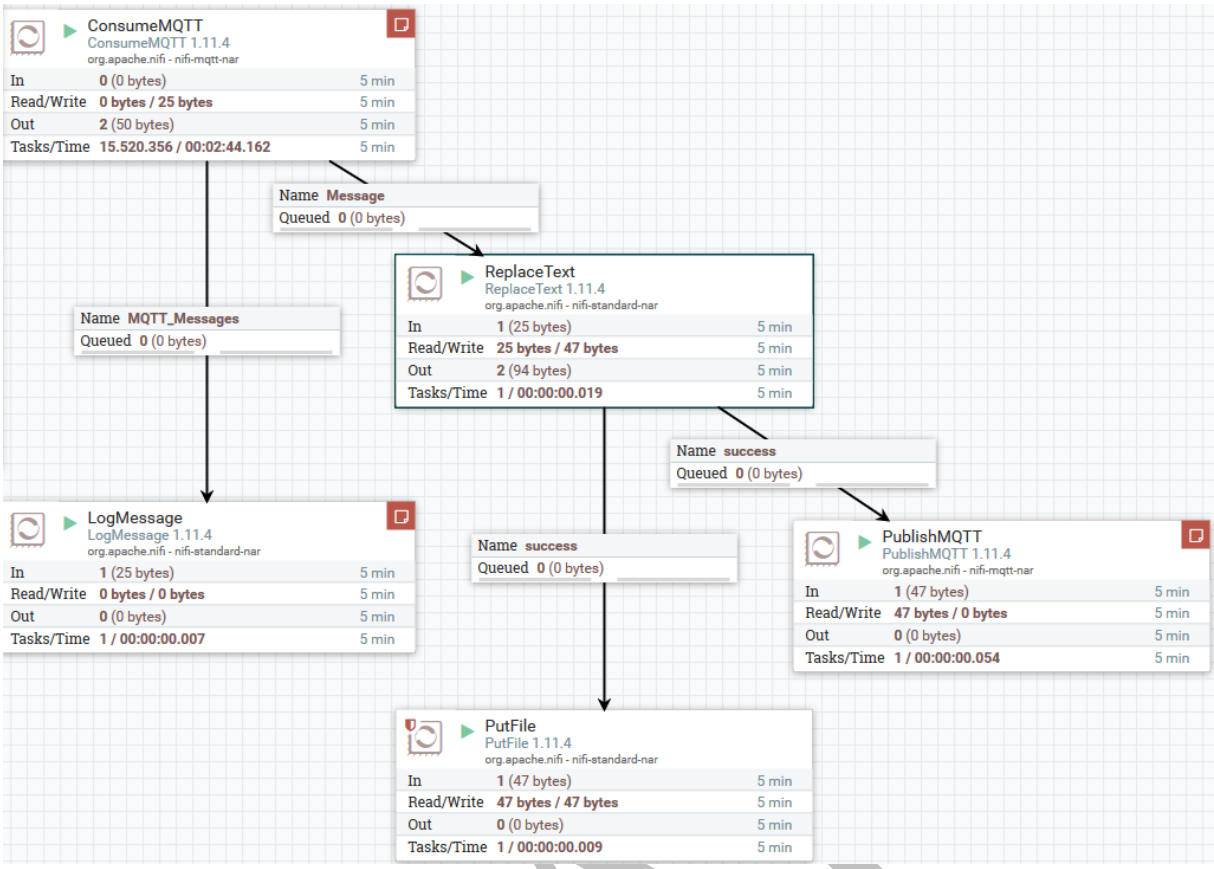

Figure 5: Nifi dataflow to enrich an MQTT message

Apache Nifi can be used by the end users either before they bring their data to the I-BiDaaS platform, or it can be built into the I-BiDaaS platform as a separate component to allow for data manipulation during the data processing on the platform.

4 Securing Universal Messaging

4.1 How to protect the UM realm

After the installation, the UM server's default configuration is such that everyone can connect freely to the server and change the configuration. One can use the supplied Enterprise Manager (UMEM) to do this in the following way.

That everyone can connect to the server can be seen by inspecting the Security tab of the realm server, which contains an entry for the Everyone group and for a user specification of *@*, which means that all users from all locations can connect. The green check marks indicate the permissions, here all is allowed, see Figure 6.

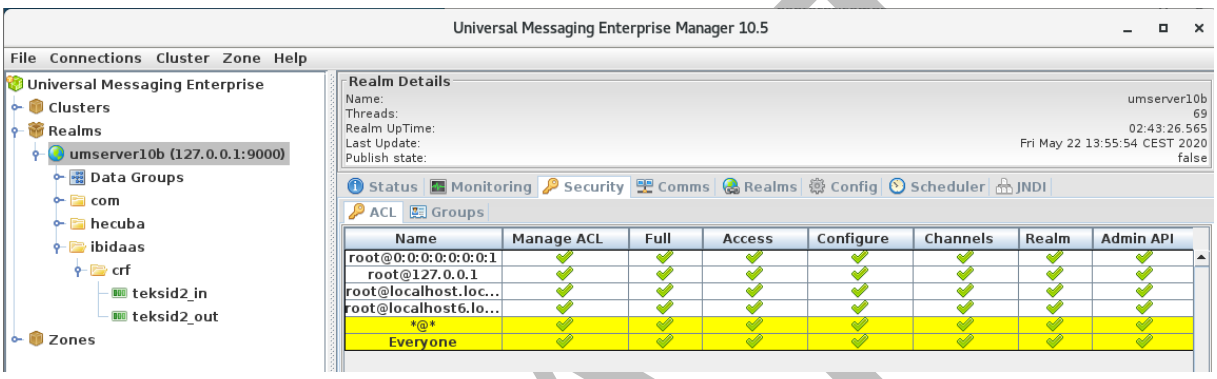

Figure 6: Default permissions of the UM server, shown in UMEM

One should delete these two lines. If one wants another user than root to have access to the server, one specifies the user and host in the following way <username>@<hostname>, e.g. ibidaas@localhost.localdomain and gives the desired permissions.

4.2 How to protect publishing and subscribing to specific topics

In the default installation, everyone can publish messages to any channel (topic) and also subscribe to any channel to receive messages. If this is unwanted, the client needs to support the provision of a username when connecting to UM. One can then e.g. specify two users, one who can publish messages and one who can receive messages, in the following way:

1) First, you need to give the two users the access right for the UM server

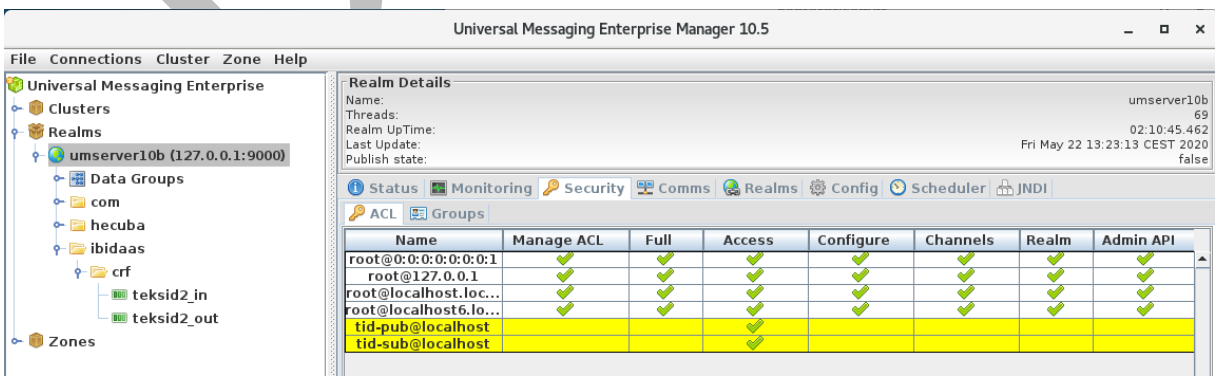

Figure 7: Setting access rights in UM

2) And then the specific publish or subscribe permission on the channel

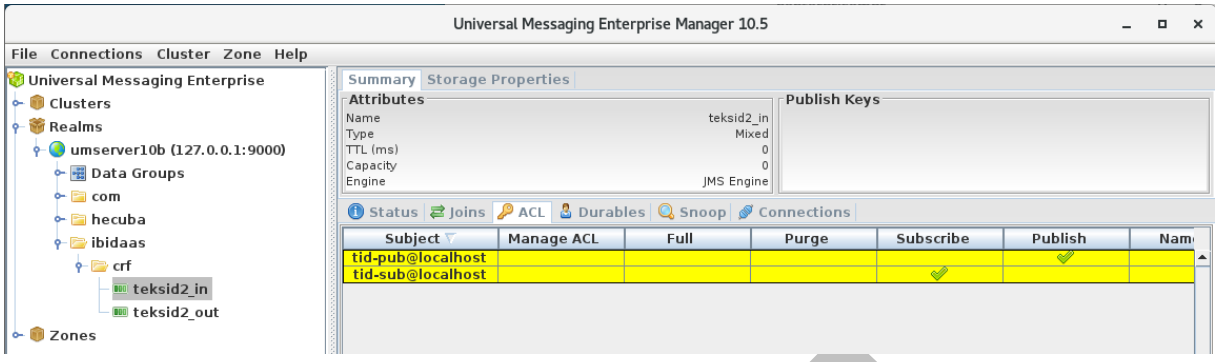

Figure 8: Setting subscription and publishing rights in UM

For trouble shooting one, should look under the Monitoring tab of the realm for messages that start with the string "UserManager:".

4.3 How to disable automatic MQTT channel creation

The default is that a channel is automatically created when an MQTT client tries to publish or subscribe to it. In order to disable that behaviour one looks under the Config tab of the realm for the MQTT Config entries and sets the parameter "EnableAutoCreateTopics" to false:

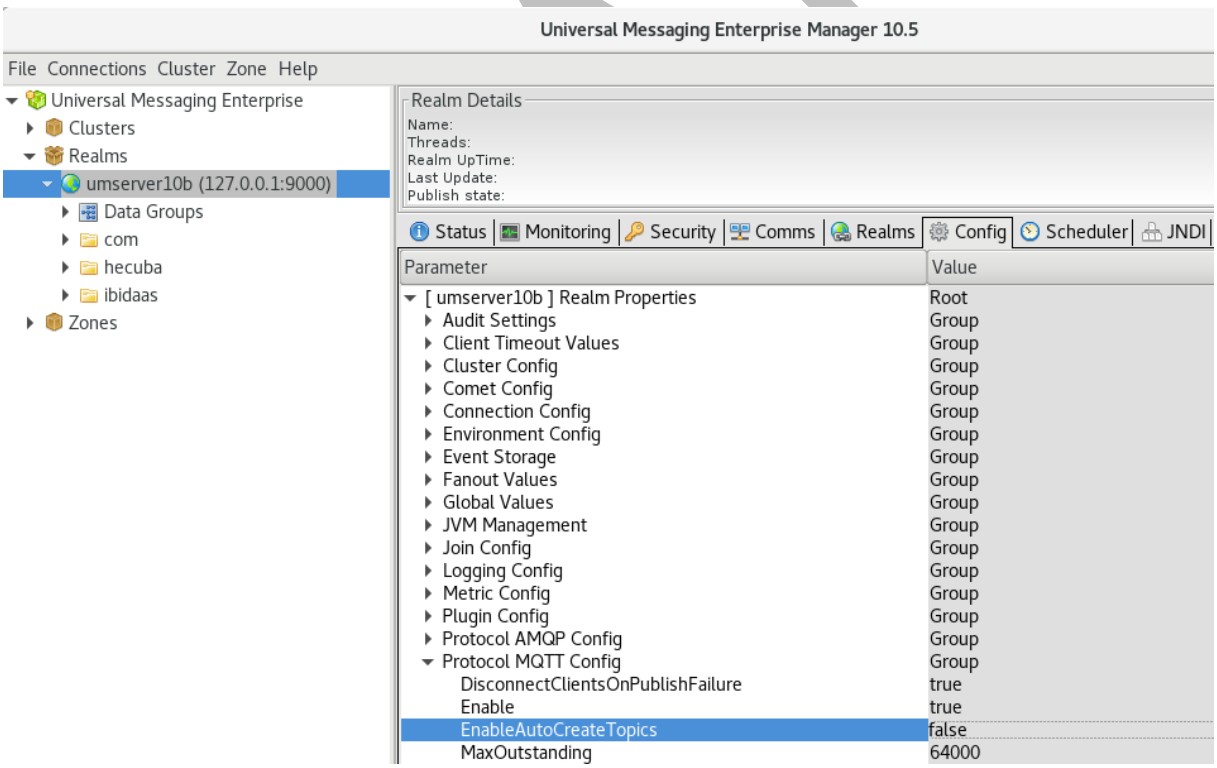

Figure 9: MQTT options in UM

4.4 How to configure UM for encrypted communication via SSL

In order to allow for encrypted communication with the UM server, one has to define an interface that supports SSL, e.g. using the NSPS protocol on the default HTTPS MQTT port of 8883:

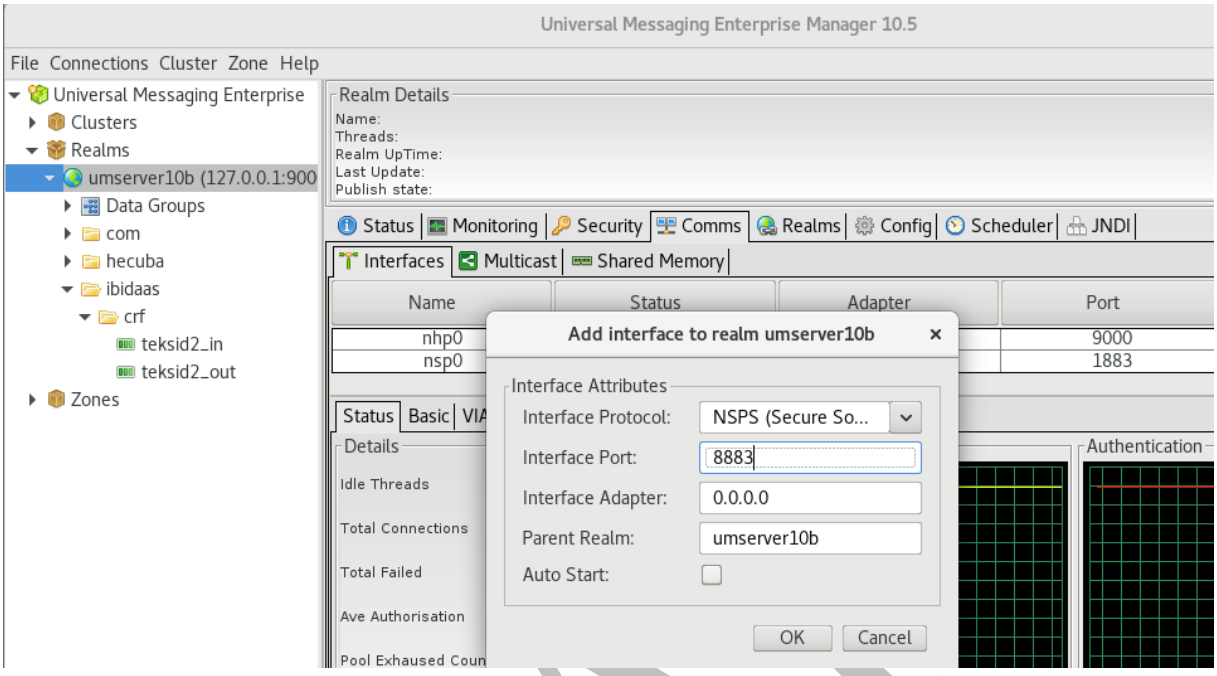

Figure 10: Configuring UM for SSL communication

One can specify that the interface starts automatically when the server starts by ticking the "Auto Start" button. Once an SSL-enabled is available, the Certificates tabs for this interface can be selected where one specifies the path to the trust and key stores, the certificate alias and the passwords. One can also enable client cert validation, which means that the certificates of the clients that use the SSL-enabled interface need to be imported into the trust store. Configuration examples are shown in Figure 11.

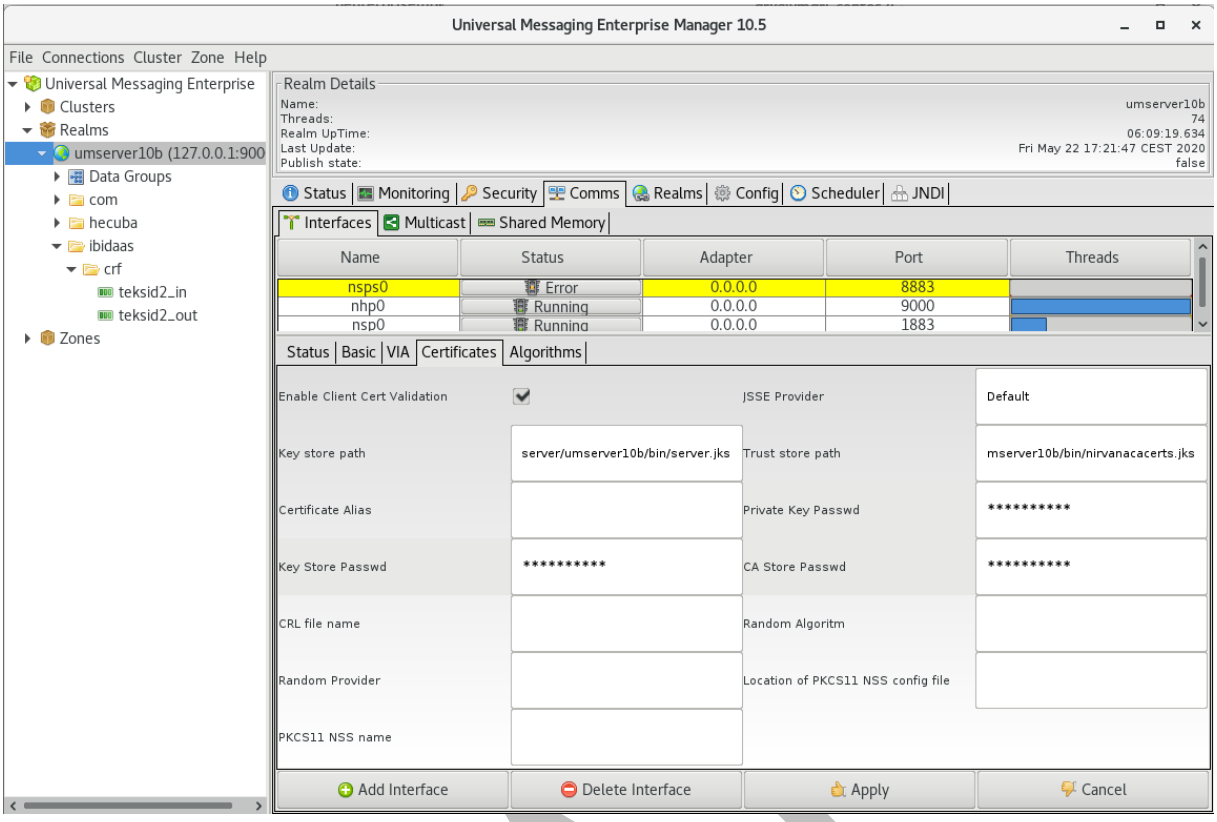

Figure 11: UM security configuration for an SSL-enabled interface, shown in UMEM

5 Conclusions and Outlook

This document described how the UM Bus is used in the I-BiDaaS platform. It started with the data ingestion via IBM's TDF tool and explained how data could be picked up from UM by the Hecuba batch tools. The main part of the document focussed on the role and usage of UM in the use cases from the three data providers CAIXA, CRF and TID. In order to allow a flexible loading of data, Apache Nifi can be used as ETL tool and also to pick up and modify data from UM. Security aspects are very important to cloud platforms and we give guidelines on how UM should be configured in order to restrict and control data access and visibility.

6 References

[1] See https://www.softwareag.com/corporate/products/az/universal_messaging/
[2] See https://www.ibm.com/products/infosphere-optim-test-data-fabrication
[3] See https://json.org/
[4] See mqtt.org/
[5] See https://www.bsc.es/research-and-development/software-and-apps/software-list/hecuba
[6] See https://cassandra.apache.org/
[7] See https://nifi.apache.org/