Data Management Component for Virtual Factories Systems

Artem A. Nazarenko¹, Joao Giao¹, Joao Sarraipa¹, Oscar J. Saiz², Oscar Garcia³ and Ricardo Jardim-Gonçalves⁴

¹ CTS, UNINOVA, DEE, FCT, Universidade Nova de Lisboa, Caparica, Portugal
a.nazarenko@campus.fct.unl.pt, {jgs, jfss, rg}@uninova.pt
² IK4-Ikerlan Technology Research Centre, Arrasate-Mondragón, Spain
osaiz@ikerlan.es
³ Information Catalyst SL, Valencia, Spain
oscar.garcia@informationcatalyst.com

Abstract. The vf-OS project aims to develop a middleware platform for the virtual factory of the future. The vf-OS is composed of several modules providing interoperability mechanisms for systems to exchange real-time data. Moreover, it enables the managing of data flows, transforming data, providing open API’s to ease integration process. The proposed Data Management Component (DMC) intends to cover the issues related to data handling, pre-processing, extracting, and data flows management for Virtual Factories. It integrates four subcomponents: a data infrastructure middleware element that handles the data communications; a data storage mechanism able to work with high loads and triples; a semantic and mapping function to establish model information integration; and an analytical module composed by machine learning and prediction mechanisms to enable knowledge extraction from the vast amount of data generated by the associated virtual factory system.

Keywords: vf-OS, Message Oriented Middleware, Virtual Factory, Internet of Things, Data Infrastructure Middleware, Data Harmonisation, Data Analytics.

1 Introduction

Modern Internet of Things (IoT) and Cyber-Physical Systems developed to cover needs of manufacturing domain are facing with difficulties related not only to amounts data generated which need to be properly stored, transmitted and delivered on demand, but also with challenges of structuring and analysing data in order to create additional value. The vf-OS project tightly related to concepts of IoT and CPS aiming at provision of an open platform for Virtual Factory (VF) and collaborative environment for different entities on different levels of manufacturing process. vf-OS ecosystem will also inherit the problem of handling enormous ascending and descending data flows. Some of these challenges are covered by other vf-OS platform components, where one of them is responsible for data connection (I/O functionality) and data movement which comes from its corresponding middleware module. However,
before data are absorbed or consumed by applications, they need to be mapped from input to output type – this will be facilitated via semantic harmonisation and transformation techniques. Finally, for data consumption analytics interfaces provided by the infrastructure must scale to extract relevant data subsets according to user’s expectations in the simplest possible way. To access the data, the proposed component also includes all the elements required for data stream management: reading, cleaning, storing, indexing, enrichment, search & retrieval, fusion, maintenance, and correspondence of open APIs. This data infrastructure will serve analytic and decision-making services to the Virtual Factory Open Operating System.

2 Virtual Factory Data and Connect

The focus of the proposed DMC (Fig 1) is on providing a set of semi-independent but related services taking the inputs of a variety data at large scale, with different characteristics such as transmission speed, etc., and providing set of nontrivial analytic operators. It is composed by four subcomponents: the Data Infrastructure Middleware; the Data Storage; the Data Harmonisation; and the Data Analytics. The Data Infrastructure Middleware provides the core access that utilise Data Storage functions for permanent and cross application access. Data Harmonisation provides transformation services, based on semantics, and is particularly connected to the first subcomponent for data connectivity. All these four will be supported by the Data Analytics.

Fig. 1 – DMC architecture

2.1 Data Infrastructure Middleware

This component specifies and implements a data bus that will support the other subcomponents and the overall vf-OS application for data storage, transformation, and analytic operations. The data infrastructure will contain adapters in order to aggregate data from various enterprise information sources including: machines, hardware sensors (which might include high-precision camera data, accelerometers, vibration, and temperature sensors), software sensors from Enterprise Resource Planner (ERP) systems and external business context data, etc. and take use of other vf-OS activities. Since sensorial data typically generates large amounts of micro measurements, the supporting data infrastructure needs to provide a high throughput technology pipeline for acquisition, pre-processing and aggregation of collected data [1].
At the early stage two main concepts for middleware were considered REST and message-oriented (AMQP). Talking about advantages and disadvantages, first to be mentioned is that REST has synchronous nature and AMQP broker implies rather asynchronous processing, for instance, message might be requested right after it has been published in the queue or later, as it is kept until being consumed. REST has an advantage that the web-kind behaviour can be transmitted to its resources. In some cases, when direct communication client-server is not possible, REST approach will be completely useless, while AMQP allows ignoring mutual awareness between exchange parties. Another clear advantage considering IoT domain is that AMQP supports both point-to-point as well as publish/subscribe patterns, while REST implies only point-to-point connection. Considering two ways of implementation with broker or broker-less, the first one was chosen due its characteristics extremely relevant for vf-OS: history logging, traceability, message queuing and reordering.

Fig. 2 – Messaging model for vf-OS components using RabbitMQ broker

To implement the abovementioned functionality message-oriented approach or middleware was selected: the AMQP protocol implemented by RabbitMQ message broker. According to the definition provided by designer of AMQP architecture [2] it is: “a binary wire protocol and well-defined set of behaviours for transmitting application messages between systems using a combination of store-and-forward, publish-and-subscribe, and other techniques.” It is based on four basic definitions: (i) **Message** – basic unit of transmitted data, the main part of which is not interpreted by the server (only headers are processed). (ii) **Exchange point** – delivery service for messages, all messages are sent into this point to be then distributed into different queues (see Fig. 2). There are three main types of exchange points: **Direct Exchange** – sending a message directly at a single queue, **Topic Exchange** – producing the copies of the message and sending to all clients, **Headers Exchange** – checking all headers on matching with query predicates from interested clients. (iii) **Queue** – here the messages are being kept, until the client request. (iv) **Bindings** – deliver necessary information about routing, in different types of exchange it is provided by different entities.

Main components represented on Fig. 2 are as follows message broker (RabbitMQ) deployed on the server side implementing the AMQP protocol for message exchange and on the client side both message producers as well as message consumers. The client side is only responsible for creating or requiring messages and also adding some headers with relevant information about type of exchange and target node/nodes the rest is being performed by broker. A Publish-subscribe middleware will help to
implement Event Driven Architecture (EDA) and is fully compatible with major industry standards (i.e. JBI, SCA, BPEL or WSDL).

2.2 Data Storage

This component implements the storage services for vf-OS assets. It supports three major dimensions of “Big Data” when dealing with intensive streaming data, namely: Volume (scale of data being processed), Velocity (data transmission speed and optimized reaction time), and Variety (supporting heterogeneous types of data). It implements a scalable data storage system, capable of handling real-time sensor data and events, based on an underlying infrastructure that transparently absorbs very large amounts of data, as well as other types of non-real-time heterogeneous data. The vf-OS Platform has different needs of Data Storage, e.g., sensor data, events, alarms, time series, maps, models, data files, log files and structured data. Each of them has its own requirements in terms of velocity of storage and querying, volume and updateability of the data, consistency, availability, partition tolerance, and other dimensions. So, it is not possible to think of a solution based on a single storage system. Here is where polyglot persistence comes into play.

In 2006, Neal Ford coined the term “polyglot programming”\(^1\), to express the idea that applications should be written in a mix of languages to take advantage of the fact that different languages are suitable for tackling different problems. The same concept can be applied to databases; thus applications utilize different databases using each for what they are best at to achieve an end goal. This is called polyglot persistence\(^2\). vf-OS storage is based on this polyglot persistence paradigm. So, following this paradigm, four different storage services are provided: 1) Relational data - to store relational data from vApps as well as other relational information of the vf-OS Platform itself; 2) Time Series data - to allow storage and querying time series data 3) Document-Oriented data - to store, retrieve and manage document-oriented information, also known as semi-structured data; 4) RDF data - to store and query subject-predicate-object triples.

2.3 Data Harmonisation

Data Harmonisation aims at (i) extracting more information from incoming data and (ii) preparing the data in a form/schema suitable for other vf-OS components and vf-OS Assets. It will enable semantic enrichment with background knowledge and data mining on real-time streams, received through the Messaging component. The data enrichment objective is to generate from the observed data additional derived attributes/features, either using external background knowledge or internal relationships within data. This could include the use of external ontologies, statistical properties, (models) of data and/or temporal characteristics of data. The key aim is to encode functional transformations of data to help the data (pre-)processing as long as the

---

\(^1\) [http://nealford.com/memeagora/2006/12/05/Polyglot_Programming.html](http://nealford.com/memeagora/2006/12/05/Polyglot_Programming.html)

\(^2\) [https://martinfowler.com/bliki/PolyglotPersistence.html](https://martinfowler.com/bliki/PolyglotPersistence.html)
vApp needs it; as an example, by carrying out analytic techniques of T5.4 to detect non-linear and other nontrivial patterns within the data (otherwise non-detectable by traditional analytic techniques).

Issues related to data extraction from heterogeneous sources with further transformation of them are solved by so called Extract-Transform-Load (ETL) Systems. The main goal of ETL systems is to make data more useful for further processing and analytics. This might be achieved through linking data and creating triples or semantic triples, which can be described as a statement of the subject-predicate-object form. ETL workflow consists of three main phases, as it follows from the definition: extraction, transformation and load. According to [3], data extraction aims at getting data from the source, which can be presented in form of flat file or data base source. Transformation is a process of “cleaning” data or bringing them to the form corresponding to the scheme, including some actions such as: normalization, filtering, etc. And the last stage is loading them to the data storage (Section 2.2).

Since vf-OS is oriented towards the software developers wanting to develop vApps for the Manufacturing Industry, it is expected that the ETLs would be part of the execution of a vApp. However, vf-OS follows a different approach: instead of programming the ETLs directly in the source code of the vApp, the ETLs will be programmed as independent services and, thus, deployed as standalone services that could be reused by other vApps and launch several instances of the same service if deemed necessary. As such, the Data Harmonisation is divided in two components: one for design-time, the Data Mapping, and another for run-time, the Data Transformation. Thus, the ETLs will be “programmed” under the Data Mapping component and deployed as these standalone services and executed within the vApp as part of the Data Transformation component.

2.4 Data Analytics

This sub-component covers the creation of building blocks for off-line analytical processing of inputs coming from the Manufacturing environment. This will include machine learning algorithms supporting supervised and unsupervised scenarios. The core of the analytic algorithms will be based on the combination of the modern statistical-machine-learning linear-algebra based algorithms (e.g. SVM, CRF, LDA, Mixture-Models) and traditional data-mining algorithms (e.g. decision trees and rules, k-means, association rules). This will cover typical classification and segmentation scenarios for enriched representations coming from semantic based ontological descriptions to capture nonlinearities in data. The key research innovation will be provided by using “multi-level” analysis on the top of more traditional machine learning algorithms simultaneously observing the data on multiple aggregation levels [1].

In a factory there will be two different types of analytics tasks to be performed: those ones depending on historical data sources, e.g. readings of sensors for a given period of time in the past, and those ones that need to be calculated on real-time as they will be involved in triggering alarms conducting to time (or spend) critical actions. As such, two different modules will be integrated into the Data Analytics component
focusing each of them in these two, although complementary, areas. Similarly, to the Data Harmonisation, the Data Analytics will be composed of analytics libraries so that the vf-OS user, i.e. the developer of vApps, would be able to re-use the existing analytics libraries that will be provided by vf-OS. This means that there will be a set of libraries each of which covering complementary functionality. There will be libraries for the various analytics approaches as for modelling and training like Random Forest predictors, which will be built on top of a SOTA analytics tool [4] that offers an API to access to such different algorithms.

3 Conclusions and future work

Current research work represents the DMC as the part of vf-OS project’s being performed within Horizon 2020 initiative. The proposed solution being developed to satisfy both basic as well as more complex needs of VF’s in managing data flows. This includes organisation of data exchange based on Message Oriented Middleware, extracting, transforming and representing data with Data Harmonisation and Data Analytics subcomponents and storage of heterogeneous data both in terms of content and format. Important requirements for VF’s such as vast amount of data being produced as well as consumed, scalability and adaptability of infrastructure, need to cope with different data in terms of formats, origin, heterogeneity and distributiveness of data sources are also considered. The main efforts are directed towards providing a scalable, open-source, modular platform for developing various industrial applications within a shared VF ecosystem. Resulting component, in fact an independent platform, will be able to solve relevant issues related to quick and agile deployment of smart applications through enabling of necessary tools and software modules around data flows handling. Further work will contain implementation of DMC within vf-OS for different manufacturing scenarios. However, DMC can be also used as an independent component being integrated with already exciting solutions and thus offering the wide range of possible implementations as a part of other systems and solutions.

4 Acknowledgements

The research leading to these results has received funding from the European Union H2020 Program under grant agreement No. 723710 "Virtual Factory Open Operating System" (vf-OS).

References