Evaluation of Architectural Backbone Technologies for WINNER DataLab A Project Focused Point of View

Sebastian Apel, Florian Hertrampf, and Steffen Späthe

Chair for Software Engineering
Friedrich Schiller University Jena
D-07745 Jena, Germany
Email: florian.hertrampf|sebastian.apel|steffen.spaethe@uni-jena.de

***Abstract*—The WINNER DataLab aims to collect, evaluate and forecast load data used to optimise the first-hand consumption of locally generated energy within buildings, by rentee, as well as electric vehicles. Every actor within a residential area has to be considered, and integration into a centralised data stream process is necessary. As a non-hard real-time system, the WINNER DataLab has to solve enterprise application integration problems, looking at complex event processing and knowledge discovery in data. This paper targets to analyse possible architectural backbone technologies. Out of a wide range of potential technologies Node-Red, WSO2 CEP and Apache Camel are selected and compared. Those technologies with a diverse field of application are used to implement a comparable test setup. Furthermore, they are analysed through their characteristics of processing, execution, usability and simplicity. As measured, Node-RED, Apache Camel and WSO2 indicate stable and fast message processing, especially in the case of raising message throughput. Node-RED surprises with constant memory and CPU loads and seams to be exciting option in rapid prototyping.**

Keywords–System Architecture; Stream Processing; Message Routing; Complex Event Processing; Renewable Energy; Smart Grid.

I. INTRODUCTION

Research on the smart grid has become a well-known task over the last years. Our research project WINNER [15] aims to integrate electromobility, the energy consumption within residential areas and the local production of electricity by, e. g. photovoltaic systems. We do not look for electric vehicles (EVs) as consumers only. They are used via the carsharing approach so that it is possible to gain booking data. Knowing the start and stop times of rides we can schedule the charging or discharging process or create prognoses on it. Summarised EVs can contribute to the stability of the power grid and help to handle load variations and load peaks. If the EVs are not used within the next hours, you can take the electric energy from their batteries and supply it to the local power grid. Additionally, it is possible to decide when to charge the car based on available information like current energy production as well as energy market prices.

Therefore, three main tasks have to be considered and brought together within our so called WINNER DataLab (WDL). At first, we collect all the produced data and store them in a meaningful way. After that, potentials have to be found, e. g. correlating weather forecasts, electricity consumption, specific time information, and the usage characteristic of EVs to optimise external energy purchase for charging batteries. In the end, we have to optimise operation. So we could control accumulate electric energy locally or supply it to the grid. Maybe it is superior or necessary to get energy from another grid operator, e. g., in case of too less output of the local energy production.

The above-mentioned facts imply an information flow managed by data streams. These must be routed and checked for mistakes. Beyond various data sources have to be integrated, like Representational State Transfer (REST) interfaces based on Hypertext Transfer Protocol (HTTP) or mail services. Out of that, we have to integrate devices using the System, Mess- und Anlagentechnik (german solar energy equipment supplier) (SMA) protocol or other TCP-based protocols.

According to backbone technologies, we have to discuss the potentials of tools made for message routing and analysing within the WDL. We focus on event-based approaches and easy integration of external components. In the end, we ask for a tool that offers the possibility of routing messages and analysis of the contained data without dropping information. This publication summarises our decision process.

In Section II, related work about terms and projects related to our approach are discussed. Section III presents the level 0 view of our WDL, and the following Section IV lists the requirements we impose on this system. As a result of that, a short overview of possible tools is presented in Section V. Furthermore, three tools are used to implement a uniform task in Section VI and compare them in Section VII by using measurement values of latency, memory consumption and CPU load. Finally, we discuss the results in Section VIII.

II. RELATED WORK

The WDL seems to be far away from traditional database management systems. The WDL should be usable as a platform for data scientists for mining knowledge as well as a platform to attach analyses for already known behaviours and relationships directly on data streams. Furthermore, the WDL should consume data from different kinds of systems as well as produce data to optimise the usage of those systems.

Connecting various types of applications by using their provided data and processes belongs to the term of enterprise application integration (EAI) [23, P. 3]. Within the area of application integration terms like message-oriented middleware (MOM) and service-oriented architecture (SOA), as well as enterprise service bus (ESB), describe how to challenge those use cases [17, S. 1][24]. As a traditional approach, MOM

Copyright (c) IARIA, 2017. **ISBN: 978-1-61208-565-4**

describes how to use asynchronous messages to decouple applications based on messaging systems [24]. SOA, on the other hand, represents an architectural concept where applications publish their precisely defined functionalities within reusable services [24]. Finally, ESB draws an open standard, which merges those ideas and defines a distributed architecture usable to integrate applications. The architecture itself describes calls and distribution of messages between integrated applications [24].

Within the scope of EAI, the enterprise integration patterns (EIPs) are the base of tools to solve integration problems. The EIPs describe a set of reusable patterns without a particular technology reference. Base concepts within these patterns are the usage of “routing” and “messages” [22].

Beside the application integration itself, activity tracking, sensor networks and analysing of market data is a central topic within the so-called complex event processing (CEP). CEP describes a general term for methods, techniques and tools. CEP helps to process events while they happen [20, S. 163].

Bringing together EAI, MOM, SOA, ESB and CEP seem to be not clearly possible. Currently, there are multiple terms to describe the problem of integration, routing, processing and analysing. The first one, Information Flow Processing, is described in [19]. This term focuses on event processing in combination with data management to “collect information produced by multiple, distributed sources, to process it in a timely way” [19]. Another term, streaming data system, focuses on processing data streams within “a non-hard real-time system that makes its data available at the moment a client application needs it” [25].

While EAI, MOM, SOA, ESB and CEP are concepts to assemble setups based on already known behaviours and relationships between messages (or events), data sciences utilise tools to mine knowledge based on already available data. This part within the WDL uses concepts from knowledge discovery in databases (KDD), which describes methods to statical analyses, applications within the field of artificial intelligence (AI), pattern recognition and machine learning [21, S. 3].

In addition to this classification of concepts within our field of application, related work targeting onto architectural drafts in data grids and smart grids can be used. Chervenak et al. [18], e. g. describes basic principles for designing data management architectures and Tierney et al. [27] introduce concepts how to monitor such grids. Furthermore, Appelrath et al. describe in [16] the process of developing an IT-architecture for smart grids as a result of a German research project, and Rusitschka et al. [26] present a computing model for managing real-time data streams of smart grids within the scope of the energy market. Unfortunately, these approaches are not directly applicable to our use case. Either they are large scaled, or focusing on data storing and mining. However, they can be considered within our architecture, which has to fill the gap between smart grids, data storing, possibilities for data mining as well as non-hard real-time event processing.

III. Architectural Draft

At this point, the level 0 view of the WDL is discussed. Taking a look at the data sources and data sinks you can get a better understanding of what the WDL should do.

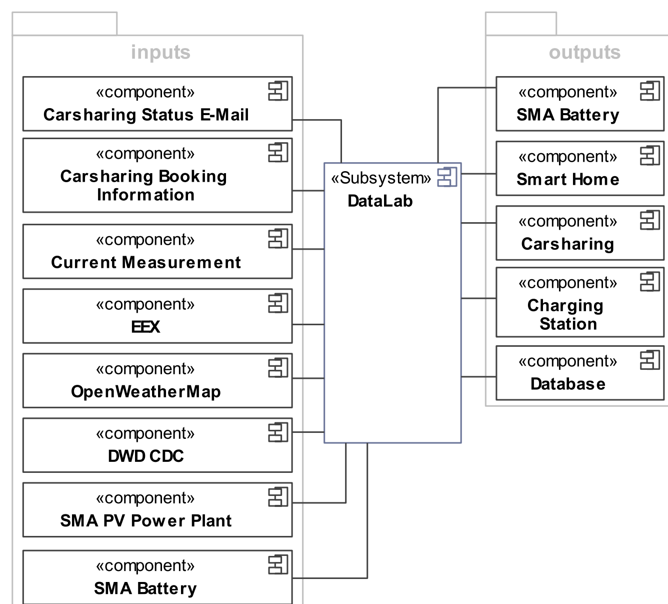

Figure 1. Level 0 view of the WDL.

At first, there are external services. They are sending messages to the WDL, or it acquires data from them. These data packages have to be assumed as inhomogeneous, e. g. carsharing data of a booking system the WDL is connected to. That means the WDL gets information on bookings like start time and end time. Out of that current state updates on a reservation such like an earlier beginning or a defect vehicle can be received. Another data source offers messages containing information on the current electrical power consumption. The actual electricity price is obtained by an interface of European Energy Exchange (EEX). Data of photovoltaic systems or batteries are gained through SMA interfaces. The German Meteorological Service [1] offers historical information on the past weather, an application programming interface (API) of the online service OpenWeatherMap [11] is available for weather forecasts.

A system working with time series, forecasts and master data must be created. As you can see in Figure 1, the WDL is positioned between the aforementioned sources and at least five data sinks. These refer to controllable devices like a charging station, a battery or Smart Home systems. On the other hand, data is delivered to the car sharing service and dumped to a database. Within our setup, a KairosDB is used as data storage as it is suitable for working with larger time series and quite easy to use.

An unanswered question is how the different components can be integrated and how analysis as well as event processing can be handled. The WINNER project focuses on the intelligent integration of components of the residential area into the Smart Grid. That means predictions must be made to get an overview of the future power consumption and the electricity production. Either one charges the batteries or one uses the stored energy to overcome load peaks. The prediction mechanism might be implemented by using artificial neural networks (ANNs) or regression methods. Thinking about energy production predictions, you might need to receive information from hardware components like SMA-devices and

Copyright (c) IARIA, 2017.
ISBN: 978-1-61208-565-4

weather services. These specific data formats require reshaping to use them in prediction mechanism. Using input filters, output filters, and stream routing the arriving information is transformed and sent to the prediction and dump units. These units save the data, send commands or just forward information to external devices.

IV. Requirements

One can divide up the list of requirements into three subsets. The first one refers to the system in general; the second touches the various components and the third covers the aspects of architecture and functional groups.

Thinking of the system in general shows that the ability to process time series data is required. An incoming message contains a time value referring to a point and a value e. g. the result of a measurement. The WDLs task on an incoming message is to associate the arriving values with a data source. Possible data sources are photovoltaic installations, batteries, power consumption measurement devices or actual weather data. Out of that, the system must handle forecast data. They are special because a complete time series and a time value, which refers to a validity point are included. At the time this point describes the time series is valid. Contemplable data sources are weather forecast services or EEX. The last category covers master data without time dependencies. Booking information or general data on devices and services belong to this group. This data may be very unstructured like text-only entries.

Focusing on technical aspects derived from our architectural draft in Section III, the WDL needs the ability to process JSON, XML and CSV values. Out of that proprietary formats have to be handled as well. Especially photovoltaic and smart metering installations tend to send production data in proprietary formats.

Central non-functional requirements are scalability and reliability. The latter refers to interfaces receiving data from external services and devices. On occurring errors incoming and shortly arrived messages should not get lost.

Keeping the interfaces in mind, one has to think of the necessary contact points to other services or the environment in general. The consumer interfaces of the WDL have to accept HTTP requests, especially while communicating with REST services. Similarly, FTP servers must be communicated with. The WDL must receive and process e-mails as well. Likewise, a file-based data transfer is needed. Finally, there are interfaces to external services using proprietary communication formats via TCP or UDP. The developed system has to enable the reception of messages sent by them. In contrast, the message producing components of the WDL primarily need to communicate via HTTP. Particularly the interface to a database can be made up of simple REST client services sending HTTP-based messages.

After paying attention to input and output components, the internal processes of routing and filtering shall be characterised. Asynchronous processing describes the most important requirement. Message queues or small buffer databases may decouple various components so they can work without waiting for each other to terminate. Furthermore, incoming messages caused by occurring events have to be converted into an internal format. To achieve this the WDL can extend these data packages with additional information. But after processing unneeded contents must be removed as well. Alongside external descriptors have to be mapped to internal descriptors and vice versa.

TABLE I. Tool overview and classification. Classification is based on tools to handle application integration (AI), stream processing (SP) and knowledge discovery (KD).

Name	AI	SP	KD
Apache Camel	✓	✗	✗
Apache Storm	✗	✓	✗
Apache Spark	✗	✗	✓
Apache Hadoop	✗	✗	✓
Apache ServiceMix	✓	✓	✗
Siddhi	✗	✓	✗
ESPER	✗	✓	✗
WSO2 CEP	✓	✓	✗
RapidMiner	✗	✗	✓
KNIME	✗	✗	✓
Node-RED	✓	✗	✗
JBoss Fuse	✓	✓	✗

The WDL has to transform the incoming data into an internal format for further processing. Additionally, the WDL has to be capable of providing data for doing manual statistical evaluations and analysis. Furthermore, the WDL has to be capable of triggering automatic evaluations and forecasts as additional components. This work is done while keeping the CEP pattern in mind.

Within this paper, we leave out the specific aspect of data storage. That means different databases are not discussed or compared. The built prototype uses a KairosDB to persist time series data. It was chosen because of an existing simple HTTP-based interface that provides easy access.

V. Tool Overview

The WDL requirement analysis illustrates an EAI task with KDD topics. Furthermore, results gathered from KDD could result in CEP related tasks, which have to be considered as well. The following list of tools covers these tasks. Of course, this list is not complete. There are a lot of tools available to handle specific tasks within the area of EAI, KDD or CEP. Our selection focuses on widely used, platform independent and easily accessible tools with suitable licenses models. Thus, the list of our selection contains mainly open source tools.

Selected tools will be classified into at least one of our primary topics: (1) tools to handle KDD related tasks, (2) tools to solve EAI related tasks and (3) tools to implement CEP related tasks. Additionally, there are (4) tools providing runtime environments to execute solutions solved with tools from class (1), (2) and (3).

Table I lists our selection of considered tools. Furthermore, this table classifies them within our previously identified main topics. Apache Camel is an open source lightweight framework to solve EAI problems based on an implementation of EIPs in [22]. Furthermore, a lot of components are available to extend the functionality of Apache Camel [4]. Apache Storm is an "open source distributed realtime computation system" with

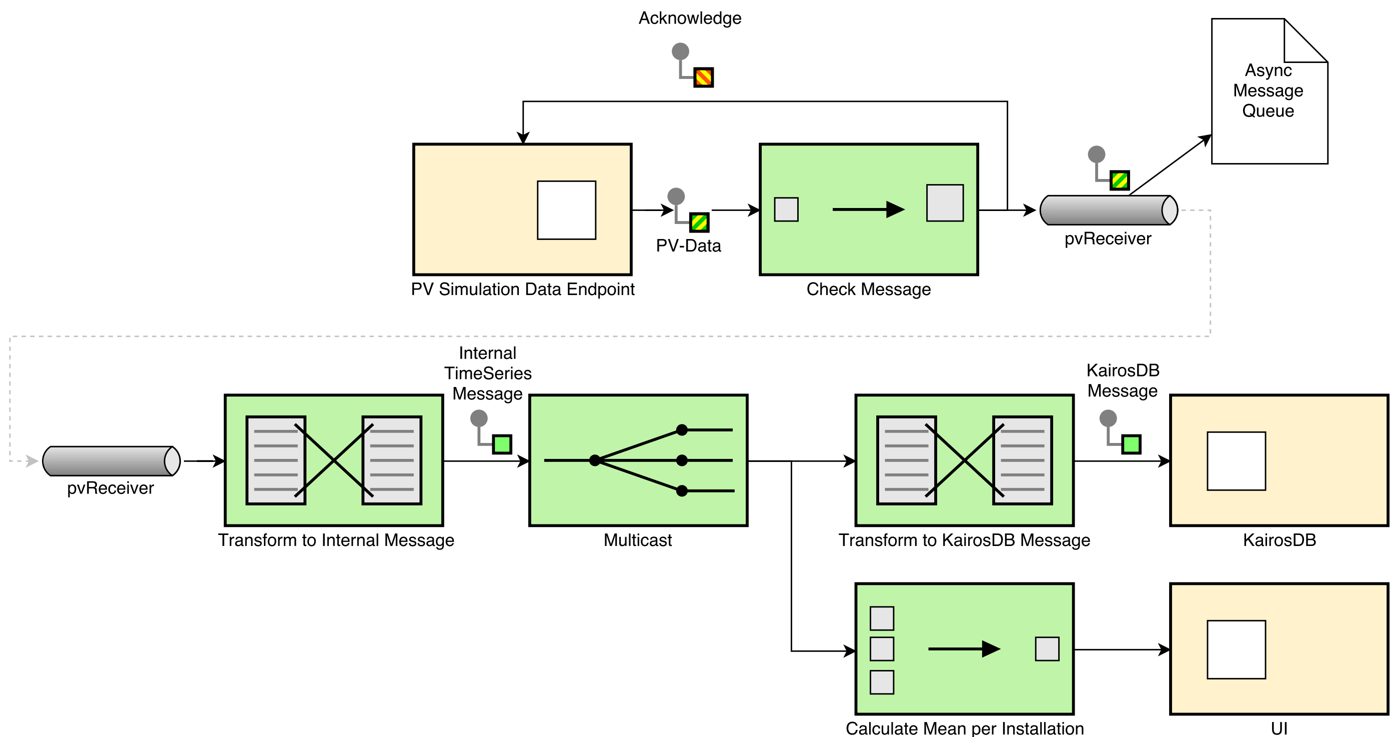

Figure 2. Visualization of our general prototype based on EIP notation.

a lot of use cases like "realtime analytics, online machine learning, continuous computation". This scalable environment can handle a lot of data streams within a specific Storm topology [5]. Apache Spark [6] and Apache Hadoop [3] are tools for knowledge discovery in data. They differ in performance as well as their internal approaches in data storage and processing. Apache Service Mix [2] and JBoss Fuse [8] are integration containers, which include other tools like Apache Camel. Siddhi [14] and ESPER [7] are CEP engines and can be used as standalone tools as well as an integration within tools like Apache Camel. WSO2 CEP is a runtime environment for the CEP engine Siddhi, which adds user interfaces for external and internal usage [13]. RapidMiner [12] and KNIME [9] are tools for knowledge discovery in already existing data. It is also possible to integrate interfaces to access data streams and use a wide range of algorithms to analyse collected data. Finally, Node-RED is a message processing framework with internet of things (IoT) roots and can be used to solve application integration problems quicklys. This framework is based on Node.js, can be extended with additional packages and deployed into cloud services like Bluemix [10].

VI. Prototype

We have selected three tools based on our preselection in Section V, which we want to use within a uniform test setup. This selection focuses on tools from different fields of application: (1) Node-RED because of its simplicity within the field of IoT, (2) WSO2 CEP because of its Siddhi engine for complex event processing and (3) Apache Camel as the reference implementation for EIPs in combination with Wildfly as Java EE based runtime environment.

The comparison is done with an uniform test setup with a simplified task, which combines the integration of a REST-based data source which encodes data with JSON, a KairosDB based data sink with HTTP interface which consumes JSON encoded data as well, and a calculation of the mean according to a particular sender device, e. g. a photovoltaic station, has to be calculated across multiple messages. The time window of these multiple messages is ten seconds. That means if sender "Station A" sends a message at 10:00:00 am the values "Station A" sent between 09:59:50 am and 10:00:00 am are used for calculating the mean. The result is delivered via HTTP request to an external service which consumes JSON encoded data as well as the data source and KairosDB.

The data source in our test setup gets its messages from a generative photovoltaic data endpoint in configurable timings. This source device transmits structured data like the tuple "(time,energy,station,id)". The first value of the generated data tuple represents a long value as a point in time, the second value a double based energy value of solar insolation. Furthermore, a tag containing the string based station name is included. Finally, the last value is a string based identifier of this single message for further time measurements. The identification value does not contain any relevant information in the context of energy data aggregation. It is only used to register and match the outgoing and incoming messages on the peripheral systems around the measurement environment.

The KairosDB endpoint of this setup gets its message as structured data like the tuple "(name,value,tags,time,id)". This tuple corresponds to the structure of data that are sent to a KairosDB instance for storing also. The included ID is not needed for the process of storing the data but necessary for matching the messages afterwards. Finally, the aggregation endpoint of this test setup gets its message as the same struc-

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-565-4

tured data like the data source tuple “(time,station,energy,id)”,

The internal message routing has to be implemented across Node-RED, WSO2 CEP and Apache Camel as shown in Figure 2. This figure illustrates the test setup and its components by using the notation of EIPs. The selected transformation and routing steps refere to the already mentioned requirements in Section IV and architectural draft in Section III to cover some kind of data source, transformation, processing, reverse transformation as well as dumping.

A. Node-Red

Node-RED is a JavaScript based message processing framework with IoT roots and can be used to solve application integration problems quickly. The framework is executed with Node.js and uses NPM for dependency management. Implementing the test setup mentioned above within Node-RED web client can be done by using a bunch of function nodes, nodes to create HTTP endpoints as well as change nodes. Change nodes are designed to modify the structure of our currently handled message object. Function nodes, on the other hand, are designed to execute custom scripts onto a particular message. Finally, nodes to create HTTP endpoints ranges from HTTP server nodes to some path which can be called, HTTP response nodes which have to be placed within a message processing path which starts with an HTTP server node and HTTP client nodes to call external resources.

The implemented setup is shown in Figure 3. As mentioned, the messaging pipe starts with “PV Receiver” to create an HTTP server endpoint for “/endpoints/pvenergy”. The message is piped onto an HTTP response node as well as to the primary processing path. The path starts with a function node to clean, enrich and transform incoming messages into the internal format. The result is forwarded to the database handling as well as the aggregation processing. Our database handling creates KairosDB compatible messages by using a template node and submits the resulting message by using an HTTP client node. The aggregation processing utilises the other function node to implement the aggregation function. This function node describes a simple memory to persist messages within a time window of ten seconds as well as calculating the mean within this window for the particular installation. The aggregation handling is finalised with a switch node to determine “NaN” values and an HTTP client node.

Summarising, Node-RED is a quickly providable platform for fast prototyping which can integrate various data sources as well as data sinks. Unfortunately, it is tricky to develop collaboratively. Well, each developer can maintain its environment, but Node-RED-Flows are managed by Node-RED itself; synchronising them between different development platforms is hard. Furthermore, any particular use case, e. g. aggregate values from messages have to be implemented manually or by using additional NPM-based components which can be added directly in Node-RED. However, it is possible to integrate a broad range of endpoints with standardised formats and protocols. Handling proprietary endpoints requires more efforts in development.

B. WSO2 CEP and Siddhi

WSO2 CEP is a tool that runs within a Java Virtual Machine (JVM). This CEP-environment offers a graphical user interface within a web browser. Using this an input receiver (named “SolarReceiver”) and two output publishers (named “AveragesLog” and “SolarPublisherDB”) are created (Figure 4). The receiver accepts data if they are JSON-formatted and sent via “HTTP-POST” request. The HTTP request is answered automatically if it is sent to the right unified resource locator (URL) of the mentioned receiver i. e. “HTTP://localhost:9763/endpoints/SolarReceiver”. The messages arriving at “SolarReceiver” are redirected to the data stream “SolarRaw”. From this queue, the data packages are picked by a so-called execution plan named “toInternalFormat” and inserted into stream “SolarIn”.

The syntax of Siddhi can be used within execution plans just as JavaScript functions. e. g. incoming string objects can be converted to timestamps. Starting with message stream “SolarIn” the internal workflow begins. Splitting is necessary because two output publishers are required. The lower branch from Figure 4 just processes the messages by adding a metric attribute and sending them to an HTTP interface of a KairosDB instance.

In contrast, the upper branch from Figure 4 performs a more complex task. The average of the last ten seconds must be computed corresponding to the last station that sent a value. This is done by taking messages from “SolarIn” and sending them to a helper stream if they are not to old. Beforehand an id for one measurement is added matching the time of arrival. If a new message arrives the helper stream is matched against it by using the station name. The average is calculated over the resulting messages and put to the output stream “averagesclean”.

The measurement values are discussed later. But some facts need to be mentioned at this point. WSO2 CEP offers a graphical user interface that aims to provide access to some script files. Out of that data streams are illustrated. The direct manipulation of data streams happens while editing script files. Other options like tracking of messages or a CPU log provide support for software engineers.

C. Apache Camel and Wildfly

Apache Camel is a Java-based EAI-framework, which is lightweight and extendable. It can be executed as a standalone routing system or within middleware infrastructures like Spring, Java EE, Apache ServiceMix or JBoss Fuse. Implementing the test setup mentioned above within Apache Camel can be done by utilising a REST endpoint and describing a route which channels incoming messages to our HTTP database and aggregation endpoints. Apache Camel offers a large number of implemented patterns, which are described within [22], as well as the option to implement custom processes, for example within “Beans”. Furthermore, it is possible to extend the framework with own components for further functionalities.

Figure 2 visualises general and the finally implemented route within Apache Camel. Its components are shown in Figure 5. The route itself is implemented by using the so-called “Java Domain Specific Language (Java DSL)” in Apache Camel. This route is implemented within “DataLabRouteBuilder” and describes the REST endpoint, which uses a servlet to process a specific resource and utilises SEDA to decouple incoming message flows from database and aggregation flows locally. SEDA is a lightweight in-memory message queue component within Apache Camel. The decoupled route

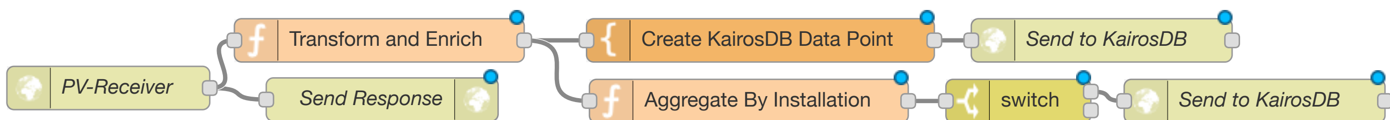

Figure 3. Node-RED implementation of the example from Section VI.

Figure 4. Message processing using WSO2 CEP.

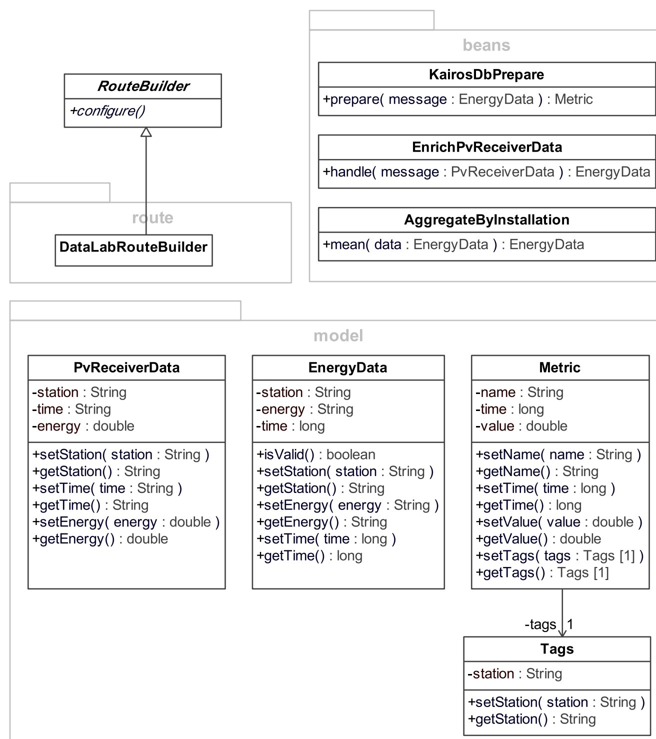

Figure 5. Apache Camel implementation of the example from Section VI.

contains the transformation and enrich bean "EnrichPvReceiverData" to transform external "PvReceiverData" into internal "EnergyData" as well as a multicast to handle the database and aggregation route. The database route contains another bean "KairosDbPrepare" to transform internal "EnergyData" into "Metric" datatypes for "KairosDb". The aggregation route includes the aggregation bean "AggregationByInstallation" itself, which is implemented as stateful bean to save messages within a time window of ten seconds and finally calculate the mean for a particular installation. Both routes are completed with an HTTP client call onto the respective external endpoint.

Finally, Apache Camel is easy to use, especially when used in combination with Maven as build and deployment tool. It is possible to describe routes within Java DSL as we did or use XML-based description to build those routes. Furthermore, Apache Camel is primary a routing engine. Any particular use case, e. g. aggregate values from messages, has to be implemented manually or by using additional libraries.

VII. Evaluation

In this section, we want to test the aforementioned prototypes. To guarantee the same conditions for every application Docker containers are used on the same machine. These containers encapsulate the runtime environment as well as the prototype itself. Our test machine runs on Debian GNU/Linux 9.0 Stretch using an Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz. Because of the main task of our prototypes is routing messages some exclusions are necessary. First, the application sending information to the routing engine is installed on another machine. Furthermore, the service which receives information the routing engine sends is placed on another machine too. This setup admits for quantifying the response time, memory consumption and CPU load of the various Docker containers or the applications within them omitting the aspect of additional load of sending and receiving applications.

The sending device transmits JSON-based structured data tuples like "(time,energy,station,id)". One could think of a solar system with a particular station identifier sending the actual energy production. The frequency of sent messages is configurable and initially set to 10 per second for the first measurements. Later we will increase them multiple times up to 200 messages per second.

The JVM for our WSO2 CEP instance and the Apache Camel prototypes are fixed to use 1024 MB of memory. We use the measured values of "jstat" for calculating the memory consumption of the tools mentioned above with a time resolution of one second. Furthermore, we sum up the usage of survival space ("S0U" and "S1U"), eden space ("EU"), old space ("OU"), metaspace space ("mu"), and compressed class space ("ccsu"). A node.js module measures the memory consumption of Node-RED, i. e. the "heapUsed" value. Out of that, the CPU load is measured by the "top" command every second. We get the response times of the various systems by measuring the time of sending and the time of receiving messages in milliseconds. The arrival timestamps of messages corresponding to database operations and aggregations are measured separately.

A. Results

At first, we want to describe and discuss the respond time, the memory consumption and the CPU load. As you can see in Figure 6(a) and Figure 6(b) the mean time between sending and receiving never increases up to more than 1 second.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-565-4

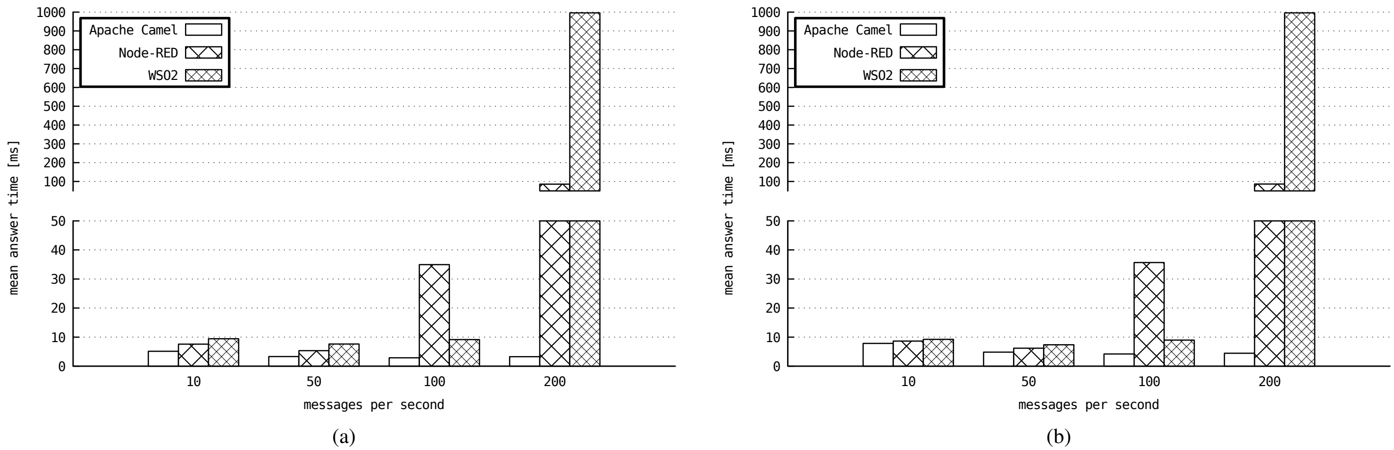

Figure 6. Mean response times of tested systems with various message frequencies for database (a) and aggregation (b) messages.

Apache Camel shows a mentionable effect of becoming faster by receiving more messages. All considered tools exhibit the same behaviour on database and aggregation messages. Out of that, we have to mention, that there is no significant difference between the response times of aggregation 6(b) and saving processes 6(a). The progressions of both cases are similar.

Figure 7 shows four segments separated by vertical dotted lines: The left one refers to the message frequency of 10 messages per second, the right to 200. In between, there are parts with increasing message rate (50 and 100). Watching the CPU load, we see an expectable process. The more messages are sent, the more CPU load is reached. WSO2 CEP uses the processor the most at least while processing 100 messages per second or more.

Figure 8 describes the memory consumption of the evaluated tools. The segments are placed according to Fig 7. We identify a memory peak for Node-RED at the beginning of the 100 messages per second section. Apache Camel presents a trend of using the less memory, the more messages arrive. WS02 CEP shows fluctuation in metaspace, which may be caused by runtime generated classes per message to execute JavaScript based functions.

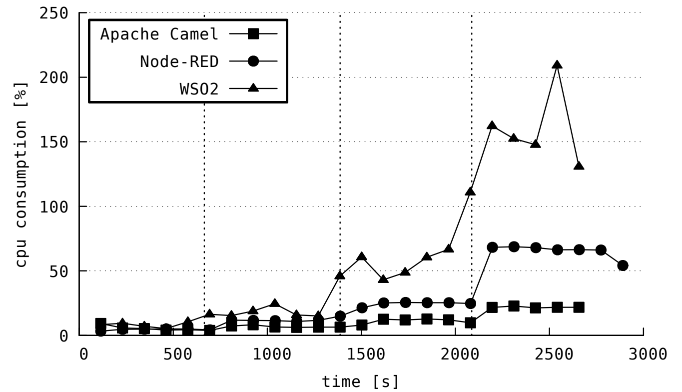

Figure 7. CPU load within four phases of sending (100 values aggregated).

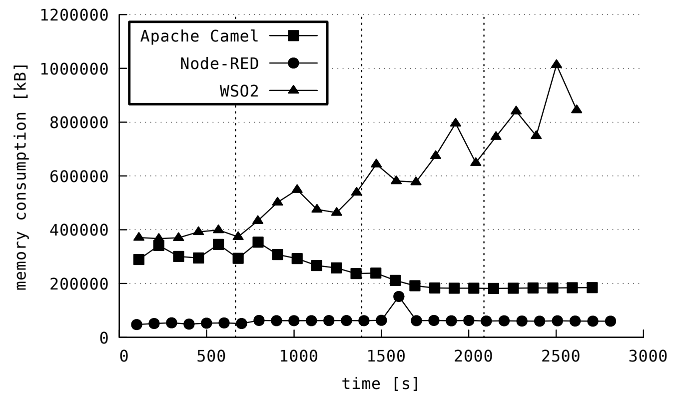

Figure 8. Memory consumption within four phases of sending (100 values aggregated).

B. Comparison

Watching only memory consumption in Figure 8 Node-RED works with the lowest. Apache Camel uses three to four times as much memory as Node-RED but does not demand the CPU that much (Figure 7). The less memory consumption of Apache Camel while processing a larger amount of messages is caused by the “eden space utilisation”. Measuring this, we see that the consumed memory decreases down to a constant minimum. WSO2 CEP shows characteristic minima of memory consumption and CPU load (Figure 7, Figure 8). Garbage collections cause these. Especially the memory graph indicates that the points of lower consumption values can be located in the middle of a sending phase not only between them.

For further evaluation, the response times can be compared. All test messages were answered. That means database information, as well as aggregation information, was not dropped. Figure 6 illustrates the fact that messages of both categories are sent with the same time delay. WSO2 CEP reaches a limit when receiving 200 messages per second. At this point, only applications allowing a time delay of one second can be built. Up to a message frequency of 100 messages per second, WSO2 and Apache Camel show a solid response time below ten milliseconds. Especially Apache Camel becomes faster while

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-565-4

handling more messages. This may be caused by the decreased overall usage of memory.

VIII. Discussion

Thinking about the main goal focusing on the architectural backbone technologies for our WDL the selected technologies cover different aspects as required. WSO2 CEP, Apache Camel and Node-RED are not directly comparable. WSO2 CEP is an environment for handling complex events, which use Siddhi to redirect message flows as well as creating high-level events based on multiple low-level events. Apache Camel, on the other hand, is an implementation of EIPs which also allows to route message. Its advantage is primary to solve integration problems, which results in much more effort to handle complex events. Finally, Node-RED is an readily usable environment to create message flows within the scope of IoT. The environment enables developers to easily integrate endpoints and handle related tasks in case of occurred events.

Apache Camel seems to be an efficient framework in case of routing messages. It is usable as a standalone application, and it is possible to use this framework within a wide range of environments like Spring, Java EE (e. g. Wildfly), Apache ServiceMix and JBoss Fuse. Our measurements show a quiet strange behaviour when raising the amount of messages per second, which cannot be comprehended fully. Taking a closer look onto the measurements show, that the eden space within the JVM is not used that much which may save garbage collection time. This might be an explanation for our strange behaviour. However, it is possible to quickly setup new routes and integrate them with a broad range of potential endpoints.

Node-RED, on the other hand, seems to be an easy to use prototyping platform at least because of its graphical user interface. This easiness applies as long as the integrated endpoints use standard formats and protocols. CEP has to be handled separately, in the case of clean code development processes it is necessary to implement additional nodes. Surprisingly, Node-RED runs quietly efficiently. The memory consumption, as well as the CPU load and response times, illustrates that, despite JavaScript, all messages are handled fast and efficiently.

Watching WSO2 CEP, there are some remaining challenges. The complexity of using windows and aggregation functions on one stream with messages from different logical sources forces the user to research intensively. A desirable feature is missing. There is no opportunity to create own objects and saving them. The included event tables only offer the option of saving whole messages. The feature of adding custom JavaScript functions cannot be used in its entirety (at least not in the release we used). A weird phenomenon shows up when comparing long values. Even if you declare all variables of streams using matching types the log of WSO2 CEP shows parsing errors. These refer to string-to-long conversions, which actually should not happen.

Finally, the evaluation lack discrete measurments on usability and simplicity. Usability and simplicity are rated, within the discussion, based on our development experience while implementing the described prototypes.

IX. Conclusion

The WDL has to be able to handle data streams as mentioned in different manners. Beside the integration and routing itself, there are tasks in the area of complex event processing as well as knowledge discovery in data. Our first reflection of architectural backbone technologies covers those aspects. Based on our experiences and measurements gathered from this test setup, we can make some decisions. In the case of a complex heterogeneous environment with different kinds of interfaces, Apache Camel seems to be a right choice. It is usable within a wide range of conditions and able to handle a lot of technologies to cover integration problems. Furthermore, in the case of handling complex events, WSO2 CEP seems to be the right choice. Unfortunately, its surrounding environment does not cover our requirements. So there are three possible approaches: (1) build CEP algorithms based on beans manually within Apache Camel; (2) integrate WSO2 CEP as a backend system; (3) extract Siddhi and integrate its functionalities into Apache Camel. However, Node-RED has its advantages in rapid prototyping and fast message processing. It might be usable as front end system to easily integrate standardised external interfaces as well as an additional platform for experiments within a productive setup. Nevertheless, everything you can do with Node-RED seems to be possible with Apache Camel too. The main difference can be found within the usability, the deployment process and the underlying language. Adapting knowledge discovery in such setups, independent of which routing engine is used, should be possible by using a database and route messages as required or by integrating available public interfaces from tools for knowledge discovery within Apache Camel or Node-RED.

However, next steps might be a final concept for the WDL by using results of this analyse as well as the implementing of this concept. Further, it is neccessary to integrate mentioned data sources into this finally implemented WDL-prototype, e.g., PV, weather forecasts, booking information, and electricity consumption.

Related implementations and instructions for further analyses of this test setup can be found within a public repository[1].

X. Acknowledements

The research project Wohnungswirtschaftlich integrierte netzneutrale Elektromobilitat in Quartier und Region (WINNER) is funded by the Federal Ministry for Economic Affairs and Energy of Germany under project number 01ME16002D.

References

[1] Climate Data Center. URL http://www.dwd.de/DE/klimaumwelt/cdc/cdc_node.html.
[2] Apache ServiceMix, 2011. URL http://servicemix.apache.org.
[3] Apache Hadoop, 2014. URL http://hadoop.apache.org.
[4] Apache Camel, 2015. URL http://camel.apache.org.
[5] Apache Storm, 2015. URL http://storm.apache.org.
[6] Apache Spark, 2015. URL http://spark.apache.org.
[7] Esper, 2016. URL http://www.espertech.com/esper/.
[8] Jboss Fuse, 2016. URL https://developers.redhat.com/products/fuse/overview/.
[9] KNIME, 2017. URL https://www.knime.org.
[10] Node-RED, 2017. URL https://nodered.org.

[1] https://github.com/winner-potential/smart-2017

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-565-4

[11] OpenWeatherMap, 2017. URL http://openweathermap.org/price.
[12] RapidMiner, 2017. URL https://rapidminer.com.
[13] WSO2, 2017. URL http://wso2.com/products/complex-event-processor/.
[14] Siddhi Complex Event Processing Engine, 2017. URL https://github.com/wso2/siddhi.
[15] WINNER, 2017. URL http://www.winner-projekt.de.
[16] Hans-Jürgen Appelrath, Petra Beenken Ludger Bischofs, and Mathias Uslar. *IT-Architekturentwicklung im Smart Grid*. Springer Gabler, Heidelberg, Germany, 2012.
[17] David Chappell. *Enterprise Service Bus*. O'Reilly, 2004.
[18] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23(3):187 – 200, 2000. ISSN 1084-8045. doi: http://dx.doi.org/10.1006/jnca.2000.0110. URL http://www.sciencedirect.com/science/article/pii/S1084804500901103.
[19] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From datastream to complex event processing. *ACM Computing Surveys*, 44(3):1–70, 2012.
[20] Michael Eckert and François Bry. Complex event processing (cep). *Informatik Spektrum*, 32(2):163–167, 2009.
[21] Christian Gottermeier. Data mining: Modellierung, methodik und durchführung ausgewählter fallstudien mit dem sas enterprise miner. Diplomarbeit, Universität Heidelberg, 2003.
[22] Gregor Holpe. Enterprise integration patterns. In *Proceedings of 9th Conference on Pattern Language of Programs*, September 2002. URL http://hillside.net/plop/plop2002/final/Enterprise%20Integration%20Patterns%20-%20PLoP%20Final%20Draft%203.pdf.
[23] D.S. Linthicum. *Enterprise Application Integration*. Addison-Wesley information technology series. Addison-Wesley, 2000. ISBN 9780201615838. URL https://books.google.de/books?id=LIYadz3qEyEC.
[24] Falko Menge. Enterprise service bus. In *Free and Open Source Software Conference*, 2007.
[25] Andrew G. Psaltis. *Streaming Data - Understanding the real-time pipeline*. Manning Publications Co., 2017.
[26] S. Rusitschka, K. Eger, and C. Gerdes. Smart grid data cloud: A model for utilizing cloud computing in the smart grid domain. In *2010 First IEEE International Conference on Smart Grid Communications*, pages 483–488, Oct 2010. doi: 10.1109/SMARTGRID.2010.5622089.
[27] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A grid monitoring architecture, 2002.