Prototype of an event-driven Future Internet Cockpit - Real-Time Sensor Events for Complex Event Processing

Robin Böhm
University of Duisburg Essen
paluno - The Ruhr Institute for Software Technology
Schützenbahn 70, 45117, Essen, Germany
robin.boehm@stud.uni-due.de

ABSTRACT

In this project we developed a prototype architecture for an event-driven cockpit in the Future Internet(FI) context. The prototype is about monitoring a business case of perishable goods in the logistic domain to improve the reaction time issues through the combination of some state-of-the-art technologies.

It uses a service platform that provides sensor event data for things in the real-wold that are provided via the websocket protocol - this platform based on the Internet of Things (IoT) concept. These events are aggregated and interpreted by an Complex Event Engine(CEP) and visualized via an HTML5 cockpit in a web-browser. Through the chosen architecture the latency between the actual business event and the visualization is nearly real-time.

1. INTRODUCTION

The aim of this project is to combine some of the state-of-the-art Future Internet(FI) technologies and using them to implement a prototype for a specific use-case that bring out the advantages of such a system. The chosen use-case is a business process management in the logistic sector that handles with perishable goods. To manage this task it is necessary to manage trucks, routes and have an overview about the current state of the deliveries. We focused on the task of reducing the latency between the actual business event and the delivery of the information to improve the possibility to react quickly to this events and increase the value of this information as shown by Hackathorn[11].

We are using the Internet of Things(IoT) and Internet of Services(IoS) concepts to linking the real-world with our business world. So we are able to use real-time sensor events to create our virtual model. We are able increase the amount and quality of event-data that can be used for calculations and combine them to more valuable complex events. To handle the messaging inside of our prototype we decided to use an Event-driven-Architecture(EDA) in combination with an Complex-Event-Processing(CEP) Engine to defining the rules for our complex events.

We are providing a cockpit for monitoring the business model with some useful visualizations.This cockpit is implemented with modern web technologies so it is usable on many devices out of the box over the specifications. It contains a map that shows the current state of given routes with an rating calculated on the raw events. Also a table that offers information in more textual way.

This paper starts with an introduction about the basics in section 2. After a survey of related work in Section 3. The Use-Case is described in section 4 and our design considerations of the different components are part of section 5. The next section 6 contains details about the implementation with a technical stack component overview followed by the conclusion in section 7.

2. BASIC

The idea of **Future Internet** is to enhance the architectures for the current state of the Internet. There are many ideas that are elaborated and discussed. The **Internet-of-Things** term first used by Kevin Ashton in the year 1999 [**?**]. He describes the IoT as enhancement from human-entered data to real world data that is captured by machines.

> If we had computers that knew everything there was to know about things - using data they gathered without any help from us - we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best.

That is a quote from an article he published 2009 in the RFIDJournal [10]. There is another concept of FI that could enables us to use all this information that is called **Internet of Services**. This is a concept that describes next-generation services that are world-wide available and could be used. They are highly adaptive and can be integrated in own services [13]. A service platform that is named COSM and described itself as 'Internet of Things Platform' on the website[12] is implementing this kind of idea. This platform enables developers and companies to connect devices and exchange data over an defined API that use the websocket protocol.

The websocket protocol is an http upgrade protocol that enables a two-way communication between a client and a remote host [2]. It was developed to provide a mechanism for browser-based applications that need this kind of communication wihout relying on multiple HTTP connections e.g. long polling[1].

To handle this high amount of realt-time data in information systems is likely to use an **event-driven-architecture** (EDA). An EDA is defined as one that has the ability to detect events and react intelligently on them. Also its designed to handle real-time or nearly real-time changes in business conditions.[14]

To add some more more meaningful value for some special business case **Complex Event Processing**(CEP) can be used. CEP is a method that tracks and analyses events and combine them to more valuable complex events. A complex event can be defined through rules and combined, integrate or aggregate other events. Also temporal rules can be defined that detecting the occurrence of events in a defined time interval.

3. RELATED WORK
TODO!!

4. USE CASE
Basically our use case is about managing the delivery of perishable goods. This is a high risk business process in the logistic domain. If some goods arriving late or there are some problems with the cooling system it is necessary to get the information of the real-world state as quick as possible to do a trade-off what could be the best action to reduce risk or costs.

Our scenario comprises the planing and alternatives in escalation situations of the delivery of these perishable goods via trucks over a defined route. These trucks are equipped with sensors that provides us the monitoring data that is used for automated real-time aggregation and interpretation and share the results in a cockpit application like it is described in the Internet-of-Things(IoT) concept. Also the routes are equipped with sensors that enables us to monitor them. The overview in such a business monitoring tool is modeled a cockpit with summary of meaningful information that should support the decision.

The cockpit provides an ampel-system with the colors green, yellow and red that visualizing the current state of the routes and trucks depending on defined threshold. Also the prototype is using complex threshold for combinations of singular states of a route or a truck that currently on this route that are also visualized in a map. The cockpit should be available via different devices like on the computer or on smart-phone.

5. CONCEPTIONAL DESIGN
As basic architectural design we are choosing an event-driven-architecture(EDA). An EDA is loosely coupled and can be used for distribution and scalability of software components. Because every component could have multiple listeners for his own events or act as sender for multiple other components. For our prototype we build service components that

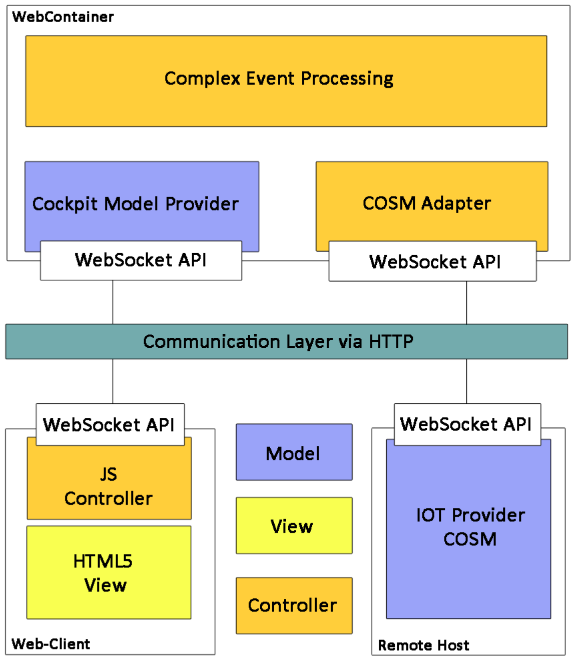

Figure 1: System technical architecture

are loosed coupled and communicating only via events. At first we using a connector that receives basic events from the COSM Service API. The component connects to the API and subscribe itself to the relevant data-streams that are prepared for this demo-case. Once the component is registered to the service-api, it receives an event via push every time a change of state appears at the data-stream. Through the static connection that happens in real-time without any overhead to create a new connection. When the event message is arrived via the socket it get parsed and transformed into the internal event structure and directly send to the components that are listening. A registered listener in our system for this COSM-Component Events is the Complex-Event-Processing(CEP) component.

This component is a processing unit that is programmed with rules that results again in events. This rules are designed for the special use case of our cockpit. It forms one or many events in a temporal context to complex events that also can be the source for new rules. It is a detection for events that could be a derivation for complex events and so we adding new more valuable events than the simple events could do. This (complex) events are used by the third component that registered as listener. This component contains the business model. The model is updated with every event the component receives. To visualize this data we are providing the information as a service via a websocket api. Our javascript client act as visualization unit that could connect to the service component and receive updates via websocket every time a state-change happens at then model site. Through this chain of event-processing also the client receives the events triggered by the COSM-API in real-time and can provide this information for multiple clients.

- EDA
- loose coupled components
- Add more event sources
- supply more processing units
- event driven architecture is extremely loosely coupled and well distributed
- communication via events
- even the source from other service is pushed via events
- 3 components
- COSM, ESPER, JS FrontEnd
- cosm
- websocket protokol
- cosm API
- transformation
- internal event structure
-

6. IMPLEMENTATION

This chapter treats with the different aspects and components of the actual implementation. It begins with an overview about the used technology stack followed by a description of the components.

6.1 Technology

As runtime environment for our backend we used oracles's JavaEE[5] implementation and as programming language Java 6. As basic application server the Apache Tomcat 7 [6] that provides a HTTP Server and the Servlet API 3.1[7] that enables the websocket support for servlets that is used to communicate with the external COSM Websocket API[12].

For the complex event processing engine we choose the esper engine[8] [1]. It provides a high-speed event processing that offers an Domain Specific Language (DSL) that enables us to easily define rules in the Event Processing Language (EPL).

To create the view frontend we using modern webbrowser technologies like HTML5[4] and ECMA 5.1 better known as JavaScript [3].

[1]Esper is provided under the open source GPL GNU License, that mean you have to have to distribute your project also as open source project if you want to use this. An alternative is Siddhi[9] that is published under the Apache Software License v2.0.

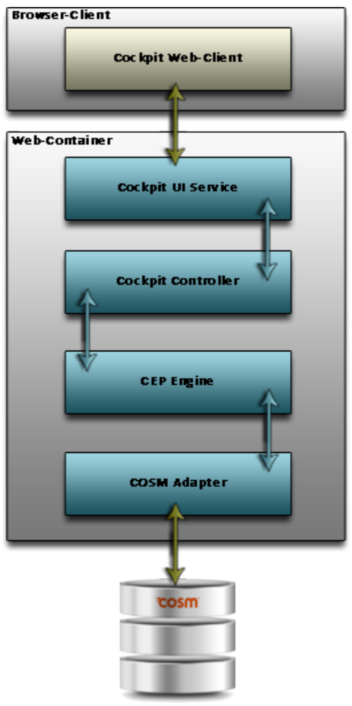

Figure 2: Angiogenetic Switch

6.2 The COSM Adapter

COSM is a online database that offers sensor-data from devices that tracks environment data from objects. It provides an API that allow to push and receive data. This data is organized in feeds that is specified as a context-specific collection of measurement data - defined by the creator. This feed contains a set of datastreams, each of this represents an individual sensor. Every datastream must have a unique alphanumeric number. It also can define some user-defined tags.

The COSM API currently based in HTTP requests. It conforms with the principles of RESTful . So you are able to use the HTTP verbs to determine a action on a data-object. GET Retrieves the current state of the object, PUT : Sets the current state of the object, POST : Creates a new object and DELETE : Deletes the object. The data-format is JSON what suited to modern web-based applications.

The service is using a API-Key based authentication that can be send via request header or parameter of the URL. If you create an account on their website, you getting an API-Key witch is full-privileged like you account. It is also possible to request new API-Keys for special feeds that could be managed individually.

The websocket support is listed as a beta-feature in the documentation. With this technique you getting an bi-directional communication socket that allows you to send messages without the HTTP overhead for every message. Once you are connected to the server, the protocol is wrapped HTTP Request in JSON to send actions to the API. So you are able to use the same GET,PUT,POST and DELETE commands as normal over the HTTP API.

To have a real benefit using the api over the websocket api it is possible to subscribe to every particular feed or datastream you are interested in. When some resource is updated the cosm-server sends immediately datastream representation of the current state via the socket. That enables you to receive real-time push notification about your streams.

What is cosm? What are datastreams? resources API Key

HTTP API / Websocket API

Asynchronous HTTP Client Suscribe to an resource getting Push Events

Wrote a little client to Update the resources via http Request.

Impl Listener for internal communication, event forwarding to engine

The COSM-Adapter Module basically is implemented as an Async-HTTP-Client that connects to the COSM WebSocket API. This API uses an JSON Format that emulating a HTTP-Frame. In this frame you are able to set method, header and body part of the a request.

- Introduction
- Diagrams
- Real-Time event monitoring
- Short Desc
- Backend in Java
- Build with Maven
- Tomcat Maven Plugin to easy run
  - COSM
  - Route Model as COSM Sensors
  - Implementation COSM
    - Java
    - COSMWebSocketEngine
    - Simple Interface
    - start,stop,listener
    - AsyncHttpClient
    - Websocket Protokol
    - API
    - API Key
    - Response, HTTP Resp Wrapper
    - http docs beta socket server
    - SUBSCRIBE to DataStream
    - unsuscribe for closing
    - Objekt Parsing
    - mapped to events
    - Push To Esper via Listener Interface
  - Esper
    - General Concept
    - Complex Events
    - Rules
    - Listeners
    - alternative Siddhi
  - JS Frontend
    - Websocket Event Model
    - Short Desc Components
    - Google Maps Route
    - jQuery Table visualization

7. CONCLUSION

- FI
- State-of-the art technologies
- real time events
- Logistic Information System / Business Process Management
- Multi-Device Support HTML5

8. REFERENCES

[1] September 2010.
[2] 2011.
[3] 2011.
[4] 2012.
[5] 2013.
[6] 2013.
[7] 2013.
[8] 2013.
[9] 2013.
[10] K. Ashton. That 'internet of things' thing, June 2009.
[11] D. R. Hackathorn. Real-time to real-value, January 2004.
[12] C. Ltd. Cosm - internet of things platform.
[13] S. Shenker. Fundamental design issues for the future internet. *Selected Areas in Communications, IEEE Journal on*, 13(7):1176 –1188, sept. 1995.
[14] H. Taylor, A. Yochem, L. Phillips, and F. Martinez. *Event-Driven Architecture: How SOA Enables the Real-Time Enterprise*. Addison-Wesley Professional, 1 edition, Feb. 2009.

APPENDIX