Karlsruher Institut für Technologie

Institut für Telematik
Pervasive Computing Systems / TECO
Leiter: Prof. Dr.-Ing. Michael Beigl

A System Design for Interacting with Linked Data in the Manufacturing Domain

Bachelorarbeit

von

Matthias Schedel

an der Fakultät für Informatik

Verantwortlicher Betreuer:	Prof. Dr. Michael Beigl
Betreuender Mitarbeiter:	Andrei Miclaus, M.Sc.
Bearbeitungszeit:	01.10.2017 – 31.01.2018

Karlsruher Institut für Technologie

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und weiterhin die Richtlinien des KIT zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Karlsruhe, den

Zusammenfassung

Durch großen Fortschritt der Sensor-Technologie in den letzten Jahren ist die Menge der durch Industrial Internet of Things(IIoT) Geräte produzierten Daten stark gewachsen. Diese Daten eröffnen neue Möglichkeiten für Unternehmen, die die mit den neuen Datenmengen verbundenen Herausforderungen erfolgreich lösen können. Eine der viel versprechenden Lösungen ist Linked Data. Linked Data Prinzipien sind im Web bereits weit verbreitet. Für viele industrielle Anwendungsfälle fehlen jedoch noch immer ausgereifte Lösungen [ZLEKS17].

Diese Thesis dokumentiert das Design und die Implementierung einer System Architektur zur Akquirierung und Visualisierung von Linked Data unter der Verwendung von Container- und Micro-Services-Prinzipien. Das Gesamtziel der Architektur ist die Verlinkung und Bereitstellung von Linked Data für Fabrikboden Wartungs- und Überwachungs Aufgaben. Die Bereitstellung erfolgt über ein allgemeines Applikations-Interface, welches die zugrunde liegende Komplexität der Linked Data Strukturen abstrahiert.

Bei der Entwicklung des Systems standen die Möglichkeit der einfachen Interaktion mit den Linked Data Strukturen des Fabrikbodens mit Hilfe web-basierter Linked Data getriebener Applikationen für den Benutzer im Vordergrund. Zur Demonstration des proof-of-concept wurden vier Visualiserungs-Applikationen entwickelt, die auf diesem App-Interface ansetzen. Der Fokus des Architektur-Designs ist es die Prinzipien Wiederverwendbarkeit, Fehlertoleranz und Robustheit der einzelnen System Komponenten zu gewährleisten. Nach Design und Implementierung wurde das System in einer Nutzerstudie evaluiert.

Abstract

Through strong progress in sensor technology in recent years, the amount of data produced by IIoT devices has rapidly increased. This data provides new opportunities for companies, who know how to solve the new data challenges. One of the promising solutions is Linked Data. While Linked Data principles are already heavily established on the web, manufacturing environments still lack mature solutions for many use cases [ZLEKS17].

This thesis documents the approach of using a container-based micro-service architecture to design and implement a Linked Data acquisition and visualization architecture. The goal of the architecture was to publish and connect Linked Data from different sources and to provide a generalizable application interface(API) for users that hides the complexity of the underlying System.

The main goal was to allow easy integration of web-based Linked Data-driven apps and graph visualization apps to support complex workflows. As a proof of concept four example applications have been developed that make use of the generalizable API. The focus of the architecture design was to ensure the principles re-usability, fault isolation and resilience of individual components. After design and implementation, the architecture with its overlaying app-interface has been tested and evaluated in a user study.

Contents

Zusammenfassung iii

Abstract v

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Structure . . . 2
- 1.3 Scientific Contribution . . . 3
- 1.4 Additional Resources . . . 3

2 Background 5
- 2.1 Building the Industrial Internet of Things . . . 5
- 2.2 Semantic Web and Linked Data . . . 6
- 2.3 Micro-services and Containers . . . 9
- 2.4 Metcalfe's Law . . . 10

3 Industrial Use Case 13
- 3.1 Linked Data on the Shop Floor . . . 13
- 3.2 Data Acquisition . . . 14
- 3.3 Data Visualization . . . 16

4 Related Work 17
- 4.1 LOD2 Technology Stack . . . 17
- 4.2 Provenance Aware Linked Sensor Data . . . 18
- 4.3 Web of Things . . . 19
- 4.4 Mobile App Orchestration . . . 20
- 4.5 Ontology-based Visual Querying . . . 22
- 4.6 Linked Open Data Based Health Visualization System . . . 24
- 4.7 Conclusion . . . 26

5 Design 27
- 5.1 Overall Requirements on the System Level . . . 27
- 5.2 Linked Data Design . . . 28
- 5.3 Supporting the Shop Floor Workflow . . . 30
- 5.4 System Requirements . . . 33
- 5.5 System Architecture . . . 36
- 5.6 User Interface . . . 45
- 5.7 Evaluation Design . . . 49
- 5.8 Conclusion . . . 50

6 Implementation 53
- 6.1 System Overview . . . 53
- 6.2 Semantic App . . . 55

6.3 Storage App . . . 57
6.4 Command Center App . . . 60
6.5 Visualization Apps . . . 63
6.6 Conclusion . . . 65

7 Evaluation **71**
7.1 Measures . . . 71
7.2 Tasks . . . 73
7.3 Overview . . . 74
7.4 Results . . . 77
7.5 Discussion . . . 80
7.6 Conclusion . . . 82

8 Conclusion and Future Work **83**
8.1 Conclusion . . . 83
8.2 Future Work . . . 84
8.3 Outlook . . . 85

Bibliography **87**

Appendix **91**

1. Introduction

This chapter introduces the reader to the field of industrial data and gives an overview of the structure of this thesis.

1.1 Motivation

The traditional boundaries between hardware and software are fading. Creating opportunities for a software-enhanced networked physical world. One of these opportunities Industry 4.0(I 4.0), coins a name which describes the intentions of companies to digitalize and network their production, including machines, applications and people.

An essential part of I 4.0 is monitoring and maintenance of Industrial Internet of Things(IIoT) devices. Data produced by these IIoT devices play a critical role in shop floor tasks that ensure proper operation of manufacturing processes.

However, not all data on the shop floor is created equal and equally accessible. One of the causes of this inequality is the use of proprietary formats and the strictly hierarchical aggregation of data, meaning all its data items are related to each other by hierarchical relationships. If data is aggregated strictly hierarchical, the creation of data islands gets triggered as stated in Kemper et al. [Ala13].

> "Strictly hierarchical information aggregation leads to separated data islands preventing a holistic view of knowledge extraction."

For instance, historical machine data at the manufacturing level is separated from Enterprise Resource Planning(ERP) data at the enterprise control level- This sepeeration prevents a holistic analysis of correlations between data values like machine parameters and details of product configurations.

Furthermore, many of the relationships between data, that are generated by different IIoT devices are not available. Therefore, it is difficult for operators in many cases to associate cause and effect and to get an overview of the interaction in a complex distributed system.

The application of Linked Data principles promises to reduce hurdles in the understanding of this data. This thesis presents the design of a proof of concept Linked Data interaction system that may provide the basis for possible solutions to this problem.

The system is designed to be incorporated into the ScaleIT infrastructure, an ecosystem of loosely coupled mashups of apps [MCS+16].

The system architecture consists of three parts, each responsible for certain aspects of monitoring and maintenance. A semantic part provides a data API that can be addressed by IIoT devices on the shop floor and a Linked Data adapter for the transformation of shop floor data into shop floor Linked Data. Second, a storage part that provides granular access for query functions to execute upon the stored Linked Data. And third, a visualization part that provides inter-actable visualizations of shop floor Linked Data.

The visualization part includes a command centre that orchestrates simultaneously running instances of visualization apps to carry out different tasks on the shop floor.

The system design is optimized to record, process and provide data from heterogeneous sources. Its design is based on a distributed micro-service architecture combined with Linked Data and IoT principles like decentralization, digital twins and device inter-connectivity. The usage of these principles may decrease the effort and improve the understanding of processes and events including cause and relation of emerging events.

1.2 Structure

This document is structured in eight chapters. **Chapter 1**, the current chapter gives an overview of this document and provides additional resources that have been created for this thesis.

Chapter 2 gives an overview of terminology and technologies used in the domain of Linked Data and Industrial Internet of Things(IIoT). It also introduces several paradigms of software and Linked Data design, like the Linked Data adapter pattern applied throughout the design process in chapter 5.

Chapter 3 introduces the manufacturing scenario in which the architecture will be used and evaluated. It also provides a motivation for a real-world use case. A factory that uses automated optical inspection(AOI) to monitor its printed circuit board(PCB) production.

In **chapter 4**, previous work done in the field of Linked Data processing architectures is presented. The chapter also presents user studies of other Linked Data interaction systems. One of these systems is an ontology-based visual querying system that is tested in an industrial maintenance scenario.

Chapter 5 presents the designed system architecture. It lays out the arguments leading to different design decisions while comparing them with possible alternatives. An example is the selection of a Linked Data storage type. It also presents a collection of applied micro-services architecture patterns, like the side-car pattern, and the arguments leading to their selection.

Chapter 6 documents the process of implementation of the design described in chapter 5. This includes a detailed outline of technical design decisions made along the way. An example is the selection of a Linked Data storage solution. It also contains a detailed description of how and by using which technologies individual design decisions, like user context based inter-app-communication, have been realized.

Chapter 7 presents the study in which the developed architecture is evaluated. In here the procedure of the study is explained in detail. This includes the exact process, like the wys tasks that had to be performed and how each participant is introduced to the system. The results of the study and an analysis are also included.

Chapter 8 outlines possible future work, that builds upon the current system architecture. An example here is the development of new visualization apps. It also includes a summarization of the key points and insights during the writing of this thesis. In the end, an outlook of possible future developments of scientific and economic relevance is portrayed.

1.3 Scientific Contribution

The objective of this thesis is to create a proof of concept architecture, that provides access to and information about the relationships of objects on the shop floor. Additionally, it investigates how different Linked Data enabled shop floor user interfaces can be combined through a common context in order to create complex search and retrieval use cases, that give controllers and maintenance workers a better view of the shop floor.

The system is evaluated regarding its usability and fit for purpose.

1.4 Additional Resources

This thesis does not provide the technical details needed to implement the described architecture, which strongly depends on the respective situation in which the system is to be used.

It focuses more on the underlying design principles used, which will then allow skilled software developers to implement the architecture for themselves. In addition to that, it provides lessons learned along the way.

For more technical details and for interested readers who want to reproduce the scenario by themselves and understand the implementation process the following resources can be conducted.

1. Github repository shop floor system: https://github.com/matthiasSchedel/ShopFloorVisualization
2. ScaleIT tutorials: https://medium.com/@scale_it_org
3. ScaleIT platform: https://scaleit-platform-documentation.readthedocs.io/en/latest

2. Background

In this chapter, the main monitoring and maintenance actors in this scenario are introduced. They are Industrial Internet of Things(IIoT) devices and Linked Data. This chapter also presents important principles of Linked Data utilization applied in the design process.

2.1 Building the Industrial Internet of Things

Starting twenty-five years ago the world wide web is changing the way we live and do business. Ten years later, through advancements in network and sensor technology, it is now not only possible to connect information but also to connect physical objects like sensors, displays or speakers. This is the beginning of the now called Internet of Things(IoT), which is estimated to consist of roughly twenty-six Billion devices by 2020 [RvdM13].

Industrial Internet of Things

Over the years companies of different domains started to incorporate IoT devices in their production workflows. At first in the communication and entertainment industry, but later also in the health and finance sector. With raising maturity of IoT technologies it became possible to integrate these IoT devices in increasingly more complex domains like manufacturing. This is called the Industrial Internet of Things.

Sensors are IIoT devices, which record everything that happens on the shop floor. Enabled by technical progress in the fields of sensor technology and electronic mass-production, the amounts of data produced by these sensors has rapidly increased in recent years.

These new amounts of shop floor data can be used to improve maintenance processes on the shop floor. This improvements help to support the goal of maintenance in the shop floor environment: Predict failures before they happen. The process of failure prediction on the shop floor is called predictive maintenance.

The idea of predictive maintenance is to use sensors and machine learning to predict error sources before failures occur. This predictions allow to save valuable resources and reduce unplanned downtime in the best case to zero. Until recently the processes needed to support predictive maintenance tasks were far too expensive for most manufacturing companies. But recent progress in sensor technology, IoT and machine learning promises

to change that.

Standardized Domain Model and Linked Data

When it comes to technological approaches for sharing data produced by IIoT devices on the shop floor, there are two popular philosophies applied in the past by companies from different industrial sectors: Standardized Domain Model and Linked Data. This section introduces the main representative of both philosophies and compares them with each other.

Standardized Domain Model. The main representative of the Standardized Domain Model is the Standard for the Exchange of Product Model Data(STEP). STEP is an electronic exchange standard for product data between manufacturing systems. It is defined in ISO 10303 and allows for structuring and integrating of heterogeneous data. In addition to computer-aided design(CAD) data exchange formats, it also covers a wide range of different product data formats from different product types and different product life-cycle stages [Pra01].

Linked Data. Open services for life-cycle collaboration(OSLC) standard is the biggest industry-wide representative for Linked Data. It is developed by an open community with the goal to create specifications between shop floor tools and entities. The OSLC standard offers two techniques to integrate Linked Data in applications, via HTTP and via HTML user interfaces.

Comparison. OSLC and STEP have certain areas in common and differ in other. On the one hand, OSLC offers a more generic schema that can be easier adapted by the user to different domains. On the other hand, OSLC only supports lightweight integration mechanisms like REST an does not support advanced export and import functionalities like STEP. OSLC is not designed for long-term archiving.

OSLC can be linked to outside resource without special transformations while STEP needs an instantiation of the outside resource according to its schema. Both approaches share the possibility of adding semantics to data. STEP is more mature and industry-wide more adapted, whereas OSLC can be integrated with IoT devices that expose data via HTTP.

Technologies applied in OSLC originally came from the world wide web. ”OSLC specifies a minimum amount of protocol and a small number of resource types to allow two such tools to work together relatively closely. OSLC is built upon Linked Data, a collection of tools and frameworks used together to publish and connect structured data from different sources seamlessly” [Com12].

The architecture in this thesis will be designed by using Linked Data principles to investigate, how well it suits the scenario instead of common solutions based on the standardized domain model. The following section presents the most important technologies and principles involving Linked Data.

2.2 Semantic Web and Linked Data

The Semantic Web, a subset of Linked Data technologies, coins a technology stack that empowers people to create data stores and build vocabularies to make their data more machine-readable. The Semantic Web technology stack consists of technologies such as RDF, SPARQL and OWL. The Resource Description Framework(RDF) is a data format for storing directed labelled graphs, which represent information on the web. There exist multiple formats that can store RDF representations. The most widely used are JSON-LD, Turtle, RDF/XML, aRDF and N3. For the system architecture, JSON-LD has been

```
{
    @context: "http://schema.org/",
    @type: "Person",
    name: "Matthias Schedel",
    jobTitle: "Student"
  }
```

Listing 2.1: RDF JSON-LD Linked Data sample of the ”Person” type

chosen because of its compatibility with the ScaleIT platform among other reasons. The following code snippet describes a JSON-LD Linked Data sample of the ”Person” type.

SPARQL is a protocol and RDF query language, which can be used to retrieve, manipulate and search data stored in RDF data formats. The Web Ontology Language(OWL) is a semantic markup language, which extends the vocabulary of RDF.

Linked Data

Both SPARQL and OWL are Semantic Web technologies used for creating Linked Data by publishing structured data in a machine-readable way on the web and linking them to other data sources. When creating and publishing Linked Data the following four principles are important to guarantee a consistent level data quality.

The following four Linked Data principles are stated by [BHBL09] et al..

1. Use of uniform resource identifiers(URIs) as names for things.
2. Use of HTTP URIs that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards(RDF, SPARQL).
4. Include links to other URIs so that they can discover more things.

Triples

Triples are used to store information about relationships between Linked Data entities, which are represented by URIs or labels. In contrast to URIs, labels only contain a string of characters without a URI. A triple consists of subject, object and predicate(verb), which are all three addressed by URIs. Only objects can contain labels. The use of URIs allows for unambiguously referencing of nodes in other statements.

Figure 2.1 contains different types of triples. They are Subject-predicate-object triples and subject-predicate-literal triples, where the literal only consists of a string without a URI.

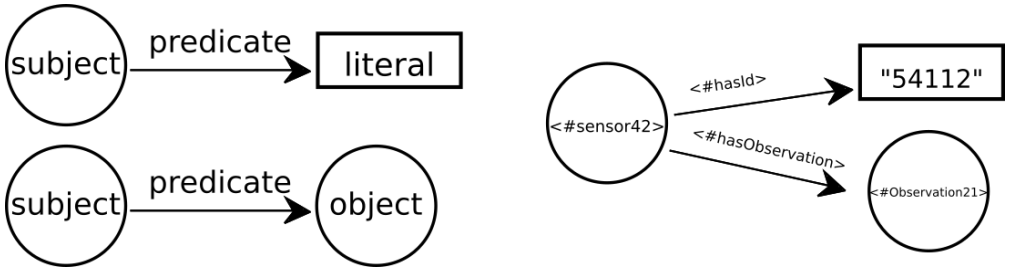


Figure 2.1: Subject-predicate-object triples and subject-predicate-literal triples.

Triples are stored in RDF triple stores, a Linked Data storage system optimized for storage and retrieval of triples through semantic queries. Semantic queries can be created by using a semantic query language like SPARQL. The graph in figure 2.2 shows different DBpedia datasets interlinked with other Linked Data sets on the web.

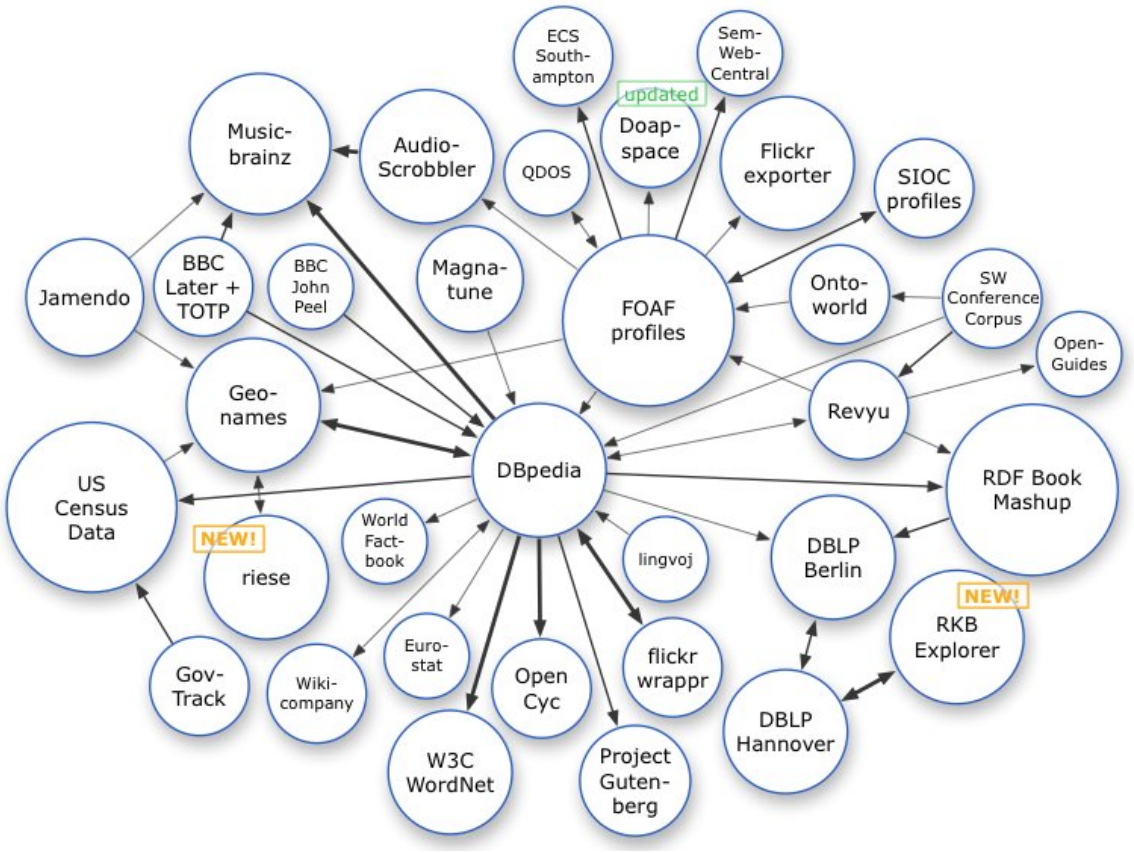


Figure 2.2: Biggest DBpedia data sets and their relations

Benefits of Linked Data Use

Use of Linked Data provides several benefits. They are uniformity, integrability, de-referencability and coherence among others [MMS14].

Uniformity. All Linked Data sets share a uniform data model: The RDF statement data model uses triples to represent information.

Integrability. Because of uniformity all Linked Data sources share the RDF data model. This makes it very easy to achieve semantic integration of different Linked Data sets.

De-referencability. The URIs used to identify entities, can also be used to look up information about the respective entity from the web.

Coherence. Triples which contain entities described by URIs from different namespaces establish links between these namespaces. This creates an RDF link, which connects the two entities while preserving the relationships in both of the namespaces with the inter-linked entities.

Linked Data Adapters

The combination of previous presented Linked Data technologies can be used to create data-driven apps that increase value by adding semantic information to the data they present to the user.

The principle of data-driven apps is applied to the principle of the Linked Data value chain, a concept stated by Latif et al. [LSH+09]. The Linked Data value chain combines the ideas of data adapters and value chains. Here, as displayed in figure 2.3, services can act as different entities in different roles to transform data to increase its value by enriching it with additional information.

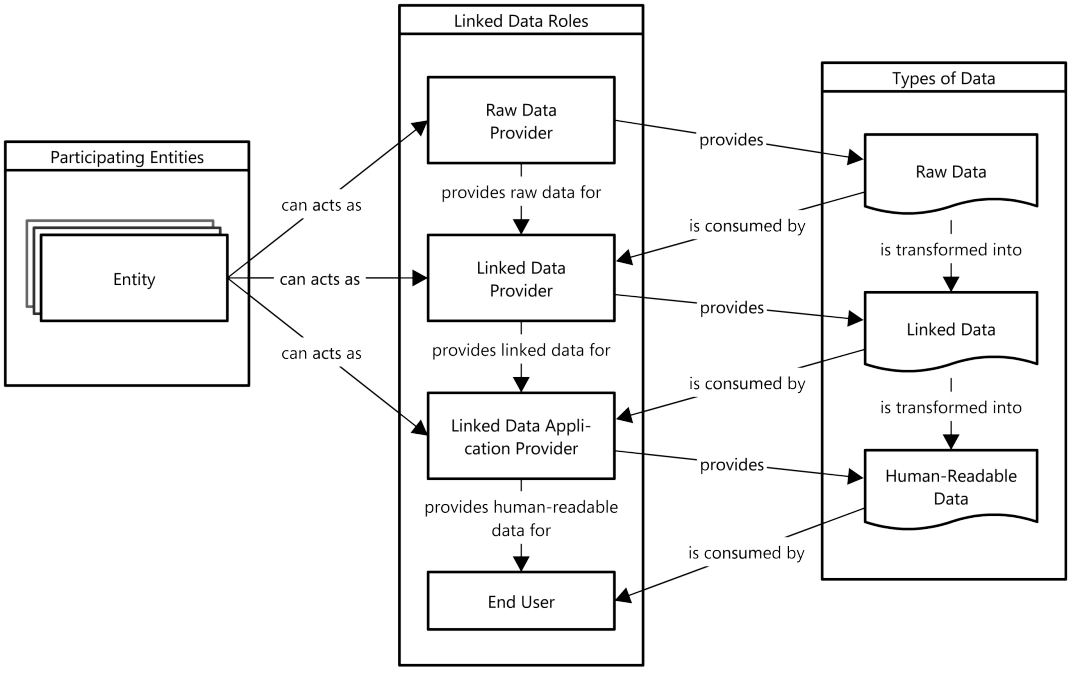


Figure 2.3: In the Linked Data value chain services can act as different entities in different roles to transform data [LSH+09].

Participating entities on the shop floor like services, machines, AOI devices and human shop floor operators can occupy the Linked Data roles displayed in figure 2.3 in the manufacturing scenario.

- **Raw Data Provider:** Provide any kind of data in any non-RDF format(raw data).
- **Linked Data Provider:** Provide any kind of data in a machine-readable Linked Data format(Linked Data). This data is provided by using de-referenceable URIs, SPARQL endpoints.
- **Linked Data Application Provider:** Provide any kind of data readable by humans(human-readable data), without requiring additional information or technical skills from them.

2.3 Micro-services and Containers

There exist different software design paradigms, that support the realization of Linked Data adapters. This work uses some of these paradigms like micro-services and containers to realize Linked Data adapters in the architecture design of the system.

Micro-services

Micro-services are narrowly focused services with the following characteristics [EP17].

1. They are responsible for a single piece of functionality.
2. They are individually deployed.
3. They consist of one or more processes.
4. A handful of them can be maintained and replaced by a small team.
5. They are individually replaceable.
6. Their communication must be standardized through APIs and loosely coupled to avoid any dependencies.

An easy way to deploy micro-services is containerization.

Containers

Containers are lightweight packaged executable pieces of software that can run stand-alone in any environment. They contain everything they need to run, like libraries, code and configuration settings. Containers run isolated from their surroundings, which helps to reduce conflicts between transitions from development to staging and production environment.

2.4 Metcalfe's Law

Digital twins are digital copies of physical objects like machines and sensors representing information about the current state of physical objects. These virtual representations create access interfaces, which provide uniform access to the current status and heterogeneous data produced by these physical objects. Because micro-services can operate de-centralized they can instantiate and run digital twins.

Data presented by these uniform access interfaces can be extended to Linked Data by using a Linked Data adapter to provide additional information about relationships between IIoT-devices, like machines stationed on the same production line. This increases the device inter-connectivity, which is represented by the number of digital twins in a network. Metcalfe's law states as displayed in figure 2.4: The value of a network is proportional to the square of the number of nodes. The nodes in our scenario are digital twins, which run on IIoT devices [Met95].

Handler et al. state describe that also works if all members can provide links for members that otherwise wouldn't be able to communicate directly [HG08]. If every service, machine or other network agent owns direct or indirect links to every member in the network the value of the network raises as stated in Metcalfe's law.

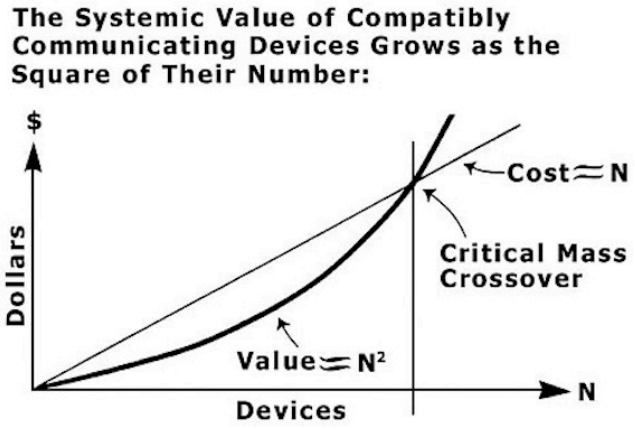


Figure 2.4: Metcalfe's law states the value of a network is proportional to the square of the number of nodes, in our case digital twins, it contains [Met95]

3. Industrial Use Case

Demands to companies and their shop floor environments are rising. To secure a sustained increase of product quality while introducing more and more complex production steps, companies are looking for new ways to improve their monitoring and maintenance systems.

3.1 Linked Data on the Shop Floor

Manufacturing of printed circuit boards(PCBs) is a domain where monitoring is essential. Because of their fragile components and complex production monitoring systems have to keep a continuous track record of everything that happens on the shop floor. To support these complex production monitoring systems Linked Data comes into play. Linked Data is used to manage and keep track of all relations and correlations of different components on a board and the machines involved in production of boards and components.

ScaleIT

This manufacturing scenario is a part of ScaleIT. A research project with the goal of increasing the device inner-connectivity and decreasing fragmentation of manual processes on the shop floor [sca].

To achieve this, ScaleIT is developing a technical platform scaling under economical conditions and provides effective support for complex workflow scenarios. In addition to scaling and complex workflow support, it provides intelligent tools to interact with virtual shop floor representations. In the ScaleIT architecture, software building blocks are represented by apps, which are used to interlink data. With this architecture, ScaleIT aims at making the production process more efficient through transparency and visualization support.

Manufacturing Use Case

This scenario is motivated by the use of automated optical inspection(AOI) devices as sensors, which control the quality of printed circuit boards during the production process. Printed circuit boards can hold several different components like capacitors, transistors and inductors. These components are soldered and baked on top the board during the construction process. With decreasing size of boards and components, the framing conditions to be meet become more strict. To adapt to this, inspection machines have to become more precise.

In this scenario every product consists of a product carrier, carrying a printed board. On each board, ten chip components will be placed, soldered and baked onto the board. After each step, a sensor will inspect the quality of the preceding step. For each board temperature, humidity and other values are measured on its surface. For each chip component, the values of shifts in x- and y-direction are measured as well as the rotation of the chips around the y-axis.

Figure 3.1 shows an AOI device scanning a soldered PCB during production.

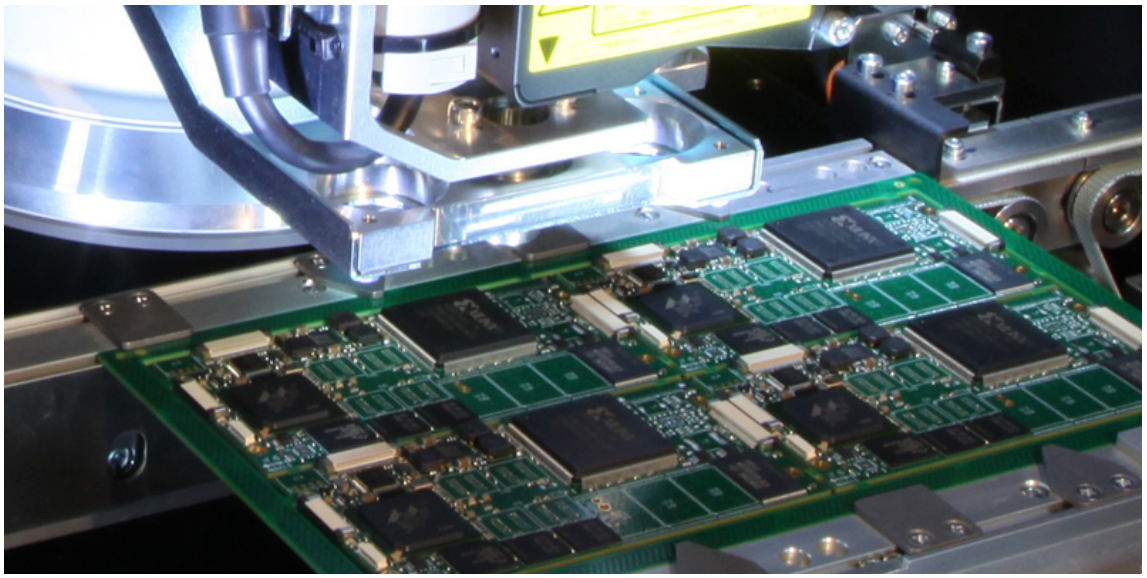


Figure 3.1: An ACI device scanning a soldered PCB during production.[Int]

Workflows on the Shop Floor

For monitoring on the shop floor, there are two essential workflows. First, the recording of occurring events in the following called data acquisition and the visualization of this recorded data combined with the shop floor structure in the following called data visualization.

3.2 Data Acquisition

Data acquisition works as follows. A sensor creates an observation record of a product in the production line. This observation record consists of three parts. They are observation data, like board temperature and humidity, observation meta-data, like serial numbers and software versions, and observation data of individual component, like the amount of x-shift and y-shift.

The sensor making the record sends the raw observation data in a byte string format to the sensor API of the shop floor system. The sensor API checks, if the observation data is correct and serializes it as a JSON, shown in listing 3.1.

It then sends the serialized data to a Linked Data adapter service, which then creates a JSON-LD, shown in listing 3.2, using the shop floor ontology and the valid JSON serialization. The JSON-LD is then sent to the storage component, which stores the observation record in the triple store database. From there it can be utilized for data visualization.

This JSON file is then converted to a JSON-LD by using the shop floor ontology.

```
{
    TimeStamp:12-12-2016:12:13:17,
    SensorId:12,
    ErrorCode:0,
    values : [
            Component1: [
               xShift:-0.1,
               yShift:+0.3,
               rotation:12
               ],
            Component2: [
               xShift:+0.0,
               yShift:-0.1,
               rotation:3
               ],
               ...,
            Component10: [
               xShift:+0.0,
               yShift:+0.0,
               rotation:5
               ]
            ]
}
```

Listing 3.1: Excerpt of a JSON file generated by the shop floor sensor API.

```
{
    @id: "http://scale-it.org/shopfloor/observation:5500",
        @graph: [
        {
            @id: "http://scale-it.org/shopfloor/observation/173:5500",
            @type:"http://scale-it.org/shopfloor/observationType/123",
            @sensor: "http://scale-it.org/shopfloor/sensor/145",
            @product: "http://scale-it.org/shopfloor/product/42",
            "http://scale-it.org/shopfloor/observationResult/":{
                @id:"http://scale-it.org/shopfloor/observationResult/12",
                @errorCode:0,
                @timeStamp:12-12-2016:12:13:17,
                @type:"...",
                @observationResultValue:{
                    @id:"http://scale-it.org/shopfloor/observationResult/122",
                    @valueComponent1:{ ... },
                    @valueComponent2:{ ... },
                    @valueComponent3:{ ... },
                    ...
                }
            }
        },
        ...
    ]
}
```

Listing 3.2: JSON-LD excerpt created from valid sensor observation data.

3.3 Data Visualization

Humans can process differences in line length, shape orientation or color without significant effort. These findings can be used to design a user interface presenting information about the shop floor in a way to support its users to make things like example identification of error patterns easier.

Supporting Workflows on the Shop Floor with Data Visualization

Data visualizations can help shop floor operators to navigate complex information structures more easily and to better understand relationships with their causes and effects on the shop floor. They make use of the previously acquired information about shop floor events, in our case sensor observation records are stored as Linked Data in the triple store database. The visualizations are created by querying the triple store database and creating visual representations of query results.

Workflows on the shop floor make use of their user context. A user context consists of a user context server and a group of visualization apps that are subscribed to this server. User contexts can be accessed by multiple users and from different user clients at the same time. Every visualization app in a user context reacts to events that happen inside the app and to events that happen inside other apps in the same user context in predefined ways.

A typical workflow of using data visualization on the shop floor can be like the following.

A shop floor manager starts the shop floor app dashboard with the intention to examine data from a specific observation record. First, he opens a list view app and searches for the observation by its record timestamp.

In the next step, he opens a tree view app and re-selects the observation in the list view app, which opens its tree-representation in the tree view app if both share the same user context.

He navigates through the tree by clicking on the respective nodes. After he found the desired information he closes the dashboard.

4. Related Work

A high amount of research has been conducted in the fields of Linked Data interaction systems. This chapter introduces different architectures of systems, designed for the interaction with Linked Data on the shop floor. It also presents various studies conducted to evaluate the usability of these systems. At last the related work from the manufacturing domain is complemented by a system design and evaluation of a Linked Open Data based health visualization system.

4.1 LOD2 Technology Stack

Linked Open Data 2(LOD2) is a technology stack, which is defined as "an integrated distribution of aligned tools which support the whole life cycle of Linked Data from extraction, authoring/creation via enrichment, interlinking, fusing to maintenance" [Aue14]. It is designed for versatile use by clear interfaces enabling plugging of third-party implementations. The technology stack utilizes the Debian package system for software integration and deployment. The system makes use of SPARQL interfaces and standardized vocabularies to allow easy integration of new tools.

LOD2 is a project from 2010-2014, funded by the European Union, with companies from different domains involved. Its goal is to showcase the benefits of Linked Data in the scenarios publishing, corporate data intranets and Open Government Data. The result is the LOD2 stack.

The LOD2 project team developed a REST-based WebAPI providing a unified application interface of the LOD2 stack. With an increasing maturity of this API, developers don't have to focus anymore on the details of different APIs.

The basic architecture of the LOD2 stack is displayed in figure 4.1. All package components act upon RDF data and communicate via SPARQL with the RDF data store. The data store is implemented with OpenLink Virtuoso to manage knowledge base graphs as well specific systems graphs describing system behaviour and status.

The LOD2 workbench, included in the system as a demonstrator, provides visualization widgets to visualize different types of Linked Data. The figure shows visualization widgets displaying statistic and spatial Linked Data.

The principles of a general REST-interface, data visualization widgets and the use of SPARQL for communication with the RDF data store will be applied throughout the design process of our system.

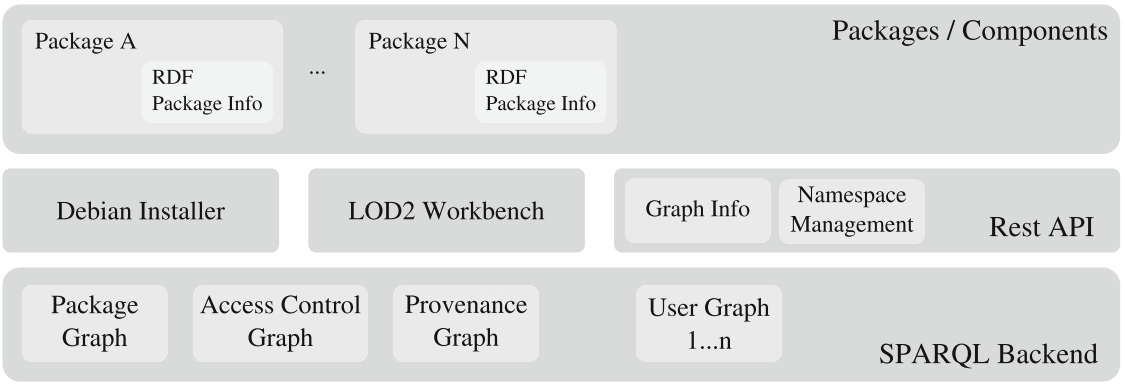


Figure 4.1: LOD2 stack architecture all components act upon RDF data and communicate via SPARQL with the system wide RDF data store [Aue14].

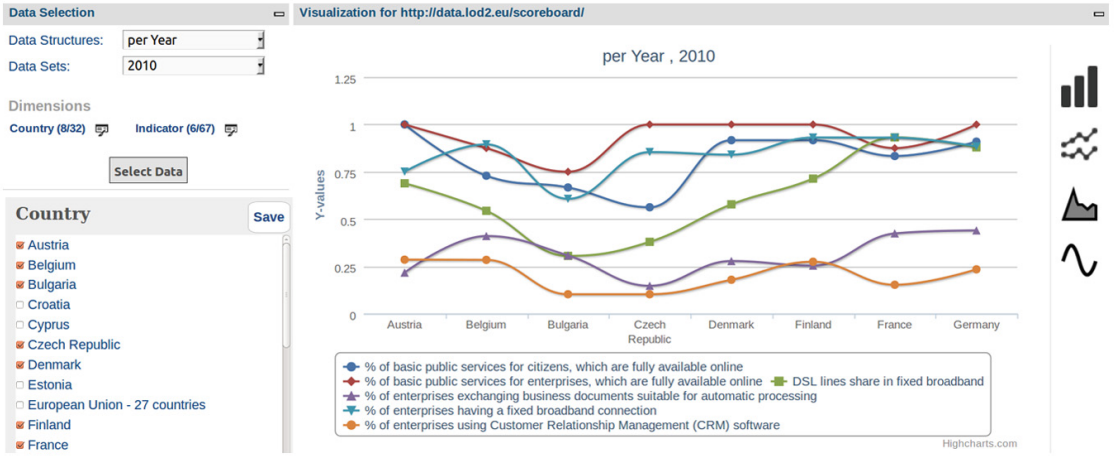


Figure 4.2: A visualization widget of the LOD2 stack to visualize statistical Linked Data [Aue14].

4.2 Provenance Aware Linked Sensor Data

Patni et al. [PSHS10] developed a framework to model and query provenance information, linked to sensor data and published as Linked Data. Their ontology-driven approach includes a representation model and a query infrastructure called Sensor Provenance Management System(PMS).

The motivation is to find all sensors, which recorded observations for a specific blizzard. To accomplish Patni et al. focus on acquiring information about blizzard classification properties, time-period, location and other data. The classification properties are High WindSpeed(exceeding 35 mph), Snow Precipitation and Low Visibility(less than a quarter mile), for three hours or longer.

The following sample query uses provenance information and a set of constraints to identify the sensors, which produced relevant observation data:

> "Find all the sensors which have observations related to a blizzard occurring in Nevada on 24th August 2005 at 11 AM" - [PSHS10]

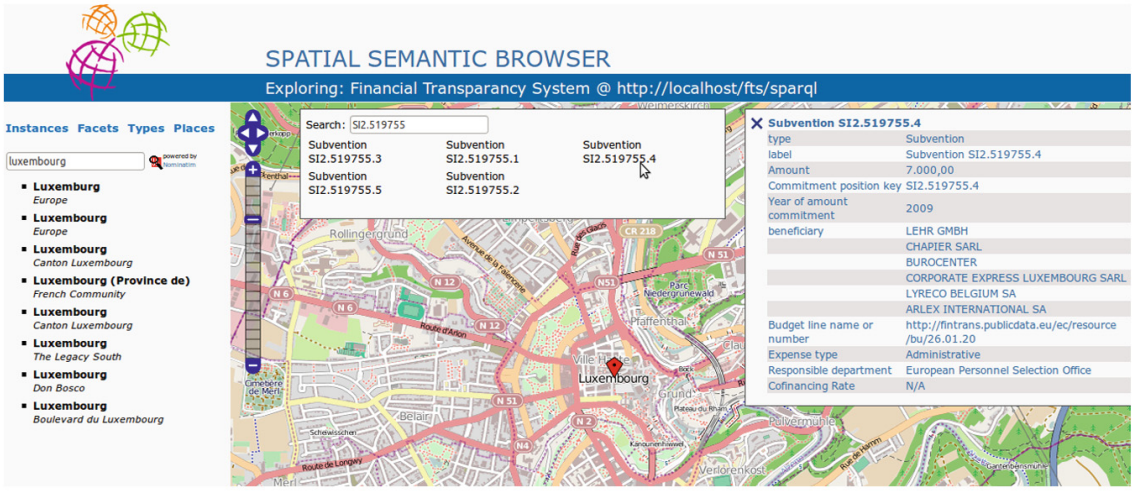


Figure 4.3: A visualization widget of the LOD2 stack to visualize spatial Linked Data [Aue14].

The transformation of raw sensor data into Linked Data is done in four phases as displayed in figure 4.4. Phase 1 describes the acquisition of raw text data, from the sensor observation service. In phase 2 this raw text data is parsed into XML. Phase 3 converts the XML data into RDF-XML by using an RDF converter API. In phase 4 the converted Linked Data are stored in the Virtuoso RDF store.

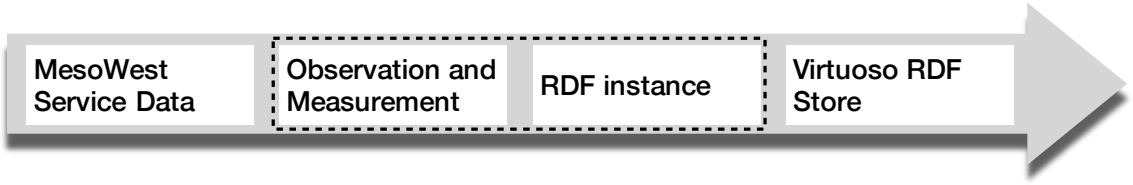


Figure 4.4: The four phases of Linked Data generation are handled by the different components of the PMS architecture [PSHS10].

The architecture of PMS handles the 4 phases of Linked Data generation. It uses the sensor provenance ontology to create the provenance representation of the previously captured data. This representation is then stored in the provenance store, where it can be accessed by SPARQL queries.

With some variation, the design of the raw sensor data acquisition workflow in the manufacturing scenario will be oriented to the four phases displayed in figure 4.4.

4.3 Web of Things

Guinard et al. [GTW10] propose the reuse and adaption of design pattern from the web for the Web of Things. They apply a REST architectural style to embedded web servers, which communicate with different IoT devices and other web servers via HTTP.

Their proposed architecture makes the functionalities of IoT devices easily accessible via a web browser with knowledge of an IoT device URI. In addition to easy access, browser interfaces are usually easier to use for users with a less technical background. They also scale easier, because they can fall back on mechanisms originally developed for the web.

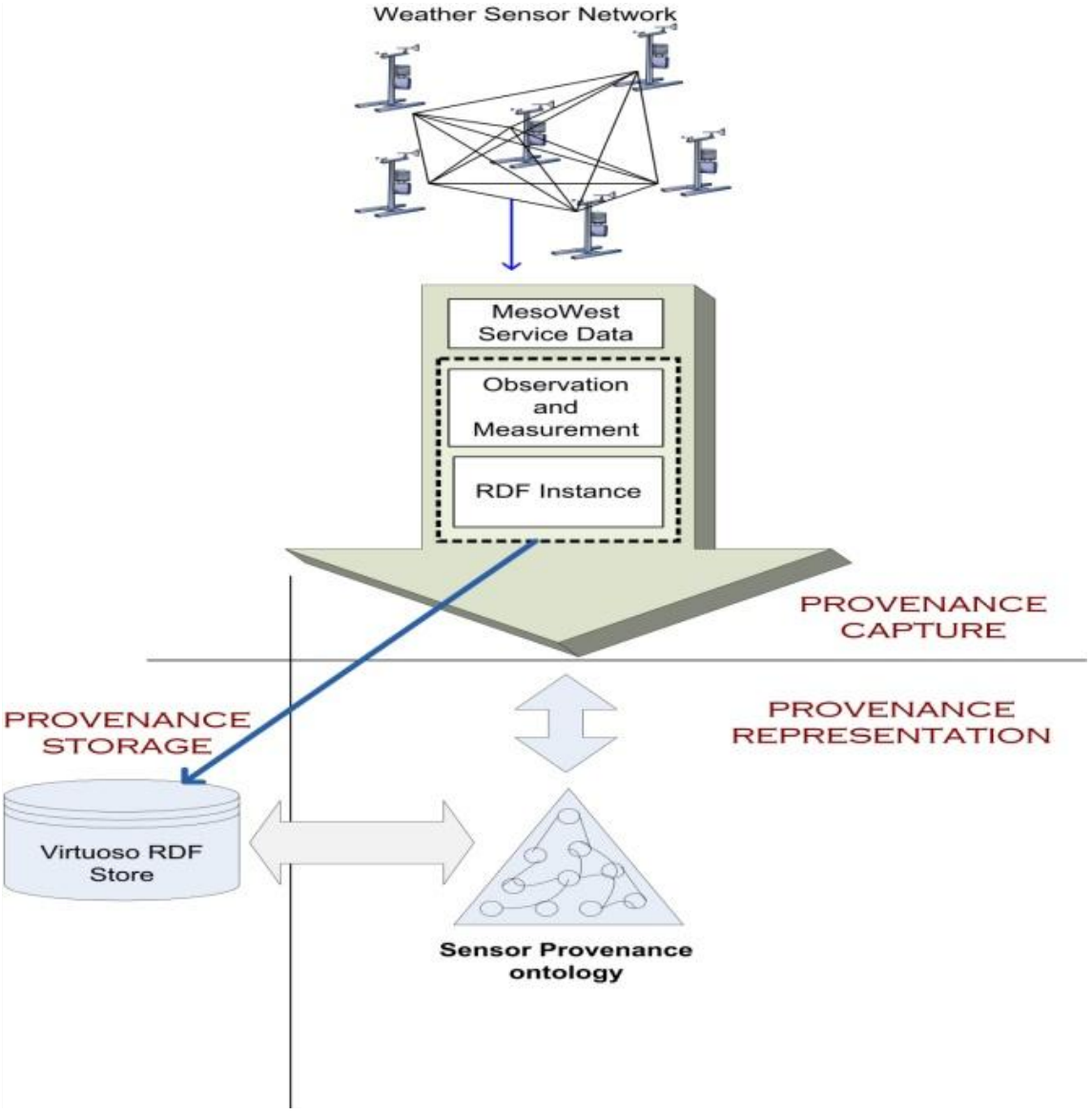


Figure 4.5: The three components of the PMS framework are provenance capture, provenance representation and provence storage [PSHS10].

The architecture presented in this document will also be based on REST, HTTP and other web technologies, but instead of IoT devices, it will be based on the IIoT devices on the shop floor.

4.4 Mobile App Orchestration

Ziegler at el. developed the concept of Mobile App Orchestration to allow the use of mobile apps in complex shop floor scenarios [ZGPU12]. The goal is to increase the value of existing data stored in existing legacy systems, by applying Linked Data principles.

The system based on this concept provides a flexible mobile app platform for users with no or little user experience. This is achieved by utilizing the links from Linked Data objects to combine limited purpose mobile apps together for the execution of complex tasks. The system is based on concepts from service orchestration and service-oriented architecture(SOA) combined with ideas from component-based software development and ontology-driven software development.

Ziegler et al. propose a strategy, consisting of three steps, for creating adaptable sets of apps, fulfilling common information needs. First, selection of suitable apps, second adaption to the context and third management of the navigation design.

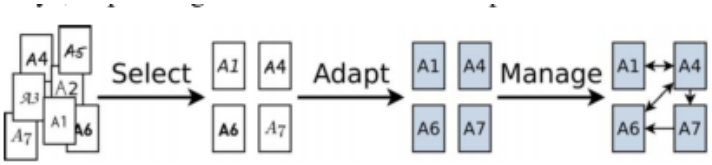
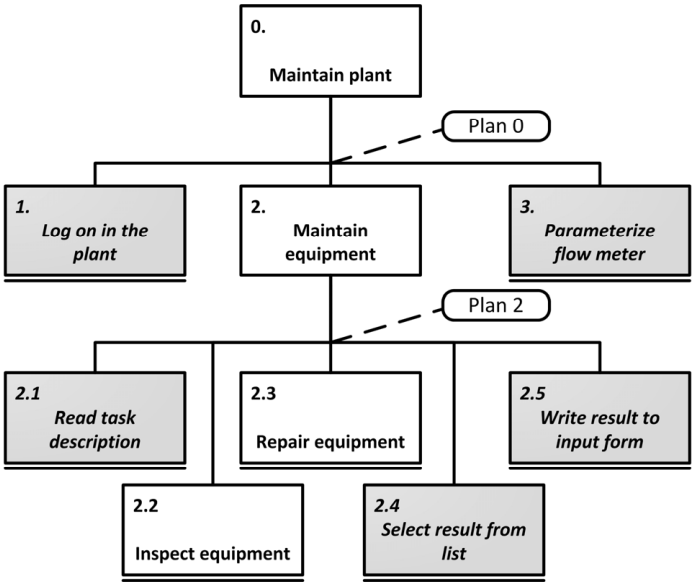


Figure 4.6: The three steps used to create adaptable sets of apps, fulfilling common information needs [ZGPU12].

To prove the concept, a study is conducted. The study is based on a scenario in the domain of mobile industrial maintenance. Participants have to carry out tasks of an external maintenance worker to assure proper functioning of a chemical plant. They can access a collaborative information space with information about the plant, maintenance management data and the status of current operations. They have to carry out the tasks, which involve inspection and overhaul tasks, by using mobile devices with access to the plant's information space. Figure 4.6 describes the tasks participants have to perform during the study and the order in which they have to be carried out.



plan	*description*
Plan 0:	Do 1 then (2, 3) in any order; Repeat 2 for each piece of equipment
Plan 2:	Do 2.1 then (2.2 or 2.3) then (2.4 or 2.5)

Figure 4.7: Task hierarchy of the tasks to be performed during in the study of the mobile app orchestration system [ZGPU12].

The participants are 11 male engineers between the age of 22 and 31. All of them completed a SUS questionnaire after they carried out the given tasks. Results of the SUS

questionnaire with a median of 77.5 are displayed in figure 4.8.

#	Score SUS
P1	65.00
P2	75.00
P3	87.00
P4	82.50
P5	80.00
P6	90.00
P7	65.00
P8	70.00
P9	80.00
P10	77.50
P11	77.50
Mean	77.27

Figure 4.8: SUS results of app orchestration [ZGPU12].

4.5 Ontology-based Visual Querying

Soylu et al. [SGS+16] conduct an experiment to measure the usability of the ontology-based visual query system OptiqueVQS in an industrial monitoring and diagnostics scenario. In the scenario, the system is extended with STARQL querying functionalities.

The scenario is built around a service centre for power plants, each responsible for monitoring and diagnostics of many thousands of gas and steam turbines with their individual components. Operators of the service centres are informed when potential problems are detected on a plant. To isolate the problem, they query raw and processed data with predefined queries. In case of new problems initially not anticipated, new queries have to be created. Creating new complex queries involves up to 2000 sensors and an IT expert has to be taken in. Because traditional database systems offer insufficient support for querying such time series data, the ontology-based visual query system OptiqueVQS is used to enable service centre operators to write such complex queries themselves. For optimal use, the system is extended to allow the use of the stream query language STARQL.

The experiment is designed as a think aloud study. The participants are three domain experts, with high technical experience, but only medium or little experience with similar ontology-querying tools. The 'turbine ontology' used for the experiment contains 40 concepts and 65 properties. There is no statement about how many of the properties are relationship type properties. After their session participants have to fill out a SUS questionnaire and a custom questionnaire for individual user feedback. Figure 4.8 shows an overview of the participants in the experiment.

#	Age	Occupation	Education	Technical skills	Similar tools
P1	37	R&D engineer	PhD	4	1
P2	54	Diagnostics engineer	Bachelor	5	3
P3	39	Engineer	Bachelor	5	2

Figure 4.9: All participants are domain experts [SGS+16].

The tasks displayed in figure 4.9 have to be solved by formulating time series queries with OptiqueVQS. For each task, a participant has three attempts. Before they start the

#	Task description
T1	Display all trains that have a turbine and a generator
T2	Display all turbines together with the temperature sensors in their burner tips. Be sure to include the turbine name nad the burner tags.
T3	For the turbine named "Bearing Assembly", query for temperature readings of the journal bearing in the compressor. Display the reading as a simple echo.
T4	For a train with turbine named "Bearing Assembly", query for the journal bearing temperature reading in the generator. Display readings as a simple echo.
T5	For the turbine named "Burner Assembly", query for all burner tip temperatures. Display the readings if they increase monotonically.

Figure 4.10: Scenario tasks [SGS+16].

```
PREFIX ns1 : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns2 : <http://www.siemens.com/ontology/gasturbine/>

CREATE PULSE WITH FREQUENCY = "PT1s"^^xsd:duration
CREATE STREAM S_out AS SELECT {
  ?_val0 ?Train_c1 ?Turbine_c2 ?Generator_c3 ?BearingHouse3_c4
      ?JournalBearing_c5 ?TemperatureSensor_c6
}
FROM STREAM measurement [NOW - "PT10s"^^xsd:duration,
    NOW]->"PT1s"^^xsd:duration
WHERE {
  ?Train_c1 ns1:type ns2:Train.
  ?Turbine_c2 ns1:type ns2:Turbine.
  ?Generator_c3 ns1:type ns2:Generator.
  ?BearingHouse3_c4 ns1:type ns2:BearingHouse3.
  ?JournalBearing_c5 ns1:type ns2:JournalBearing.
  ?TemperatureSensor_c6 ns1:type ns2:TemperatureSensor.
  ?Train_c1 ns2:hasTurbine ?Turbine_c2.
  ?Train_c1 ns2:hasGenerator ?Generator_c3.
  ?Generator_c3 ns2:hasBearingHouse3 ?BearingHouse3_c4.
  ?BearingHouse3_c4 ns2:hasJournalBearing ?JournalBearing_c5.
  ?JournalBearing_c5 ns2:isMonitoredBy ?TemperatureSensor_c6.
  ?Turbine_c2 ns2:hasName "Bearing Assembly"^^xsd:string.
}
SEQUENCE BY StdSeq AS seq HAVING EXISTS i IN seq ( GRAPH i {
  ?TemperatureSensor_c6 ns2:hasValue ?_val0
  } )
```

Listing 4.1: Sample query for a diagnostic task written in STARQL [SGS+16].

receive a brief introduction to the topic and tools along with an example. During the session, the participants are asked to think aloud, to get information about any difficulties they encounter during the task session.

The results of the system usability score(SUS) are displayed in figure 4.11.

To evaluate the system developed in this work, a user study is designed re-using some of the building blocks used in the study of Soylu et al. In this study users also have to solve specific tasks in a restricted time frame. In contrast to the turbine experiment, the tasks

SUS criteria	P1	P2	P3	Avg.
I think that I would like to use this system frequently	5	4	4	4.3
I found the system unnecessarily complex	1	3	2	2.0
I thought the system is easy to use	5	4	5	4.6
I think that I would need the support of a technical person to be able to use this system	1	1	1	1.0
I found the various functions well integrated	4	4	4	4.0
I thought the is to much inconsistency in the system	2	2	2	2.0
I would image that most people would learn to use this system very quickly	5	4	4	4.3
I found the system very cumbersome to use	1	2	2	1.6
T felt very confident using the system	4	4	3	3.6
I needed to learn a lot of things before I could get going with this system	2	1	1	1.3

Figure 4.11: SUS results of experiment [SGS+16]: Table header: Participant number. Table body: 1 for "strongly disagree" and 5 for "strongly agree".

will be performed by using the visualization tools of the system implemented in this work. The restricted time frames will be longer in this case because participants are no domain experts. For evaluation analysis, SUS and collection custom questions to gain individual user feedback will also be utilized in addition to the UEQ questionnaire.

4.6 Linked Open Data Based Health Visualization System

Public health sector faces similar problems like the manufacturing domain. Data is often only available in pdf and excel files(data islands). [TKKF14] et al. develope a system architecture to Link Public health data and visualize it, using Linked Open Data technology.

The system is developed by using the format RDF, Fuseki triple store for storage and Sgvizler for data visualization. In addition to this components, a SPARQL interface is integrated to query interlinked data sets of the Web of Data using silk, a link discovery framework.

The architecture of the system, displayed in figure 4.12, consists of four layers: A data layer, a transformation layer, a service layer and a presentation layer.

In the scenario, HIV related data elements are imported into the system from the WHO global health observatory database and automatically linked to elements from information resources like DBpedia and Bio2RDF using the Silk framework. For users without the ability to create SPARQL queries, a Linked Data search engine is integrated into the system. This search engine facilitates flexible queries and different kinds of Linked Data visualizations to give users a better understanding of the data.

To assess the usability of the system, a study is conducted with the goal to evaluate potential options for health information visualization and retrieval system development. The results of the SUS questionnaire with a value of 82 are displayed in figure 4.13.

This work emphasizes the importance of possible Linked Data applications in different sectors besides to industrial manufacturing. There it can make a big contribution in improving the understanding of data for users with no or little background in domains. Like the example in this study: Patients in the public healthcare sector, with no or little knowledge about their personal health data.

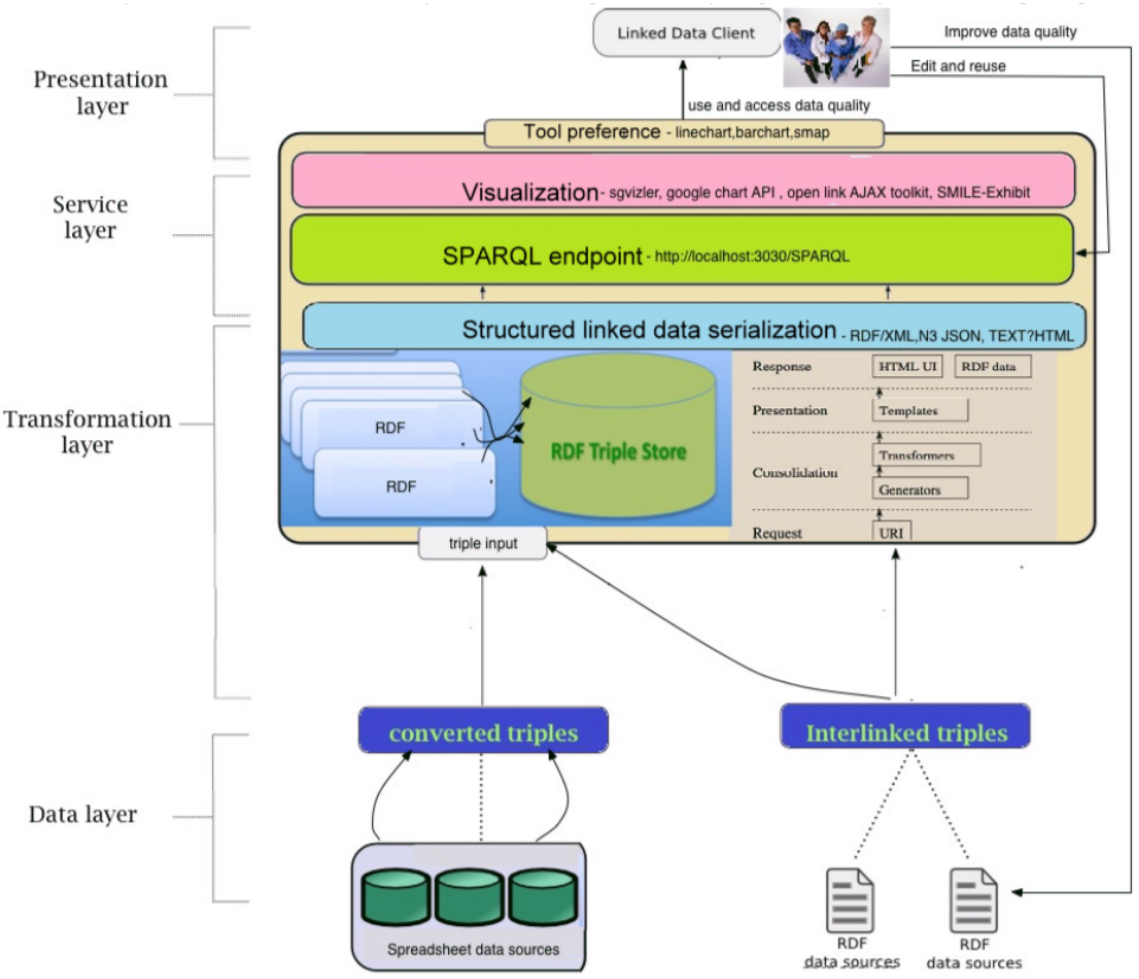


Figure 4.12: The architecture consists of data layer, transformation layer, service layer and presentation layer [TKKF14].

SUS criteria	**1**	**2**	**3**	**4**	**5**
I think that I would like to access my data this way	10(59)	3(18)	-	4(24)	-
I found the system unnecessarily complex	2(12)	5(29)	-	10(59)	-
I thought the system is easy to use	4(24)	5(29)	-	7(41)	1(6)
I think that I would need the support of a technical person to be able to use this system	6(35)	1(6)	-	7(41)	3
I found the various functions well integrated	12(71)	2(12)	-	3(18)	-
I thought the is to much inconsistency in the system	3(18)	2(12)	-	10(59)	1(6)
I would image that most people would learn to use this system very quickly	6(35)	1(6)	2(12)	6(35)	3(18)
I found the system very cumbersome to use	4(24)	2(12)	-	11(65)	-
I felt very confident using the system	12(71)	-	-	5(29)	-
I needed to learn a lot of things before I could get going with this system	7(41)	1(6)	-	9(53)	-

Figure 4.13: SUS results of the study [TKKF14]. In header: 1 for "strongly disagree" and 5 for "strongly agree". Format in body: Number of participants(total percentage).

4.7 Conclusion

The related work presented in this chapter covers different approaches to the design of shop floor interaction systems and the ways these systems are evaluated. There are three key points that will be used in the system design in the next chapter.

The first key point is the design of the four phases described by [PSHS10] for the acquisition process, generating Linked Data from raw sensor data, which is enriched by provenance information. The second key point is the use of web technologies like HTTP and REST to create uniform APIs as described by [Aue14] and to adapt design pattern from the web as described by [GTW10]. And third the concept of App Orchestration described by [ZGPU12] in mobile app scenario. This concept will be used to design a web user interface, which allows users to arrange visualization apps to best perform certain shop floor tasks.

From the evaluations presented in this chapter, two key points will be applied when creating the evaluation for our system. The first key point is the way tasks are prescribed for participants. The tasks in our evaluation will be oriented to the tasks described in the evaluations of [SGS+16] and [ZGPU12]. The second key point is the usage of questionnaires like SUS to assess the usability and other important usability metrics of the system. As it has been used in the [ZGPU12], [SGS+16] and [TKKF14].

5. Design

The goal of this chapter is to document the requirements of an architecture supporting the scenario that is introduced in the third chapter. And further to present a solution architecture, that through fulfilling these requirements, may improve monitoring and maintenance processes on the shop floor. In order to achieve this goal, principles of micro-service architecture and containerization are used in the design process.

5.1 Overall Requirements on the System Level

Modern PCB production depends on complex production workflows, involving multiple expensive and sensitive components. To assure for a shop floor system to work reliably under those conditions, several overall requirements have to be met on a system level. To illustrate each requirement an example is included.

Dynamic adaption to changes in amounts of workload. The system requires the ability of dynamically adding and removing running instances of services depending on the current workload. This is done automatically without human intervention. The behaviour of the system how to adapt has to be easily changeable by shop floor operators with a simple step process.

Example: The ability to scale the number of instances acting on the shop floor on demand(scaling systems resources from handling ten sensors to thousand or more sensors on the software site is done automatically by the system).

Adaptability and re-usability of individual components. To fulfil the demands of a modern shop floor the system requires the property of dynamic and individual configuration for specific scenarios or use cases. If the shop floor has to be reconfigured to make changes in the production it must be possible to do this by just replacing individual apps.

Example: An exchange in a single component like the storage type in the storage component would require no changes in the other components.

Recording of occurring events. To be able to reconstruct all occurring events on the shop floor and to react accordingly to each event the system must record and process every occurring event within ten seconds. If needed, the system must detect the events, a user wants to be notified about and send a notification accordingly within the time frame, specified by the user.

Example: Every time a machine processes a product a record of this action is created and stored separately to be re-used later.

Just in time capabilities. To get notified about occurring events, to take action within five seconds the system must perform all system tasks, which cannot be waited on, in a justifiable time-frame. When a service fails to act, react or to respond in a time span of ten seconds or longer, it has to be replaced within five seconds.

Failure tolerance and resilience. Every failure occurring within a service must never cause an error in a service it is collaborating with. The error has to be always be detected inside the service itself causing it to be replaced within five seconds of failure discovery. In case the error remains after the replacement a notification has to be created within ten seconds.

Example: If a failure occurs while transforming incoming sensor data the responsible unit performing the task will be restarted and after three failures a notification will be sent to created within ten seconds.

Data integration and extendable data interfaces. Data produced on the shop floor usually have different origins. This creates a situation in which the system has to process multiple heterogeneous data sources. In order to cope with this, it must be possible to add and replace handlers of heterogeneous data sources easily to the system by only changing one component.

Data interfaces have to be easily extendable for new sensor types, which make use of new data formats. The adding of the belonging mapping of their accompanying ontologies has also to be possible with minimal expenses by using Linked Data integration.

Example: When introducing a new sensor, producing a new type of data output, the steps necessary to integrate this sensor into the current system properly must only involve a minimal replacement of individual components. In this case, the formal class definition of the sensor with its attributes has to be added to the ontology.

5.2 Linked Data Design

The goal of designing the Linked Data structure for this scenario is to create the minimum viable amount of entities and relationships needed, to make the shop floor environment easily understandable for users who are not familiar with the shop floor.

This section documents the choices for main Linked Data building blocks. They are Linked Data exchange format, Linked Data storage type and Linked Data query language.

Choosing a Linked Data Exchange Format

In this scenario, JSON and JSON-LD will be uniformly used for data exchange between services. It is chosen over other Linked Data formats like turtle or RDF(XML), because of its high distribution and usage on the web and its simple serialization and de-serialization process.

Storing Linked Data

For this scenario, an RDF triple store or graph database has to be chosen for storing Linked Data. It has to offer a SPARQL query interface, HTTP interface and a web-interface with an editor. The storage solution has to be easy, to set up within the shop floor system and it should focus on ease of use rather than performance. At last, it must allow for easy

replacement in case of changes in requirements.

Graph Databases vs RDF Triple Stores

In a general view, RDF triple stores can be classified as graph databases. Both, RDF triple stores and graph databases, focus on the relationships between data, where a data point is a node and a relationship between two nodes is an edge. But, there are also some minor differences between RDF triple stores and graph databases.

RDF triple stores can only be queried by using SPARQL, while graph databases often allow the use of several query languages like GraphLog, BiQL, GOOD, SQL, SoSQL and others. Graph databases can store multiple types of graphs like weighted graphs, undirected graphs and hypergraphs, while RDF triple stores focus only on storing RDF triples. Graph databases are node-centric in contrast to RDF triple stores, which are edge-centric. RDF triple stores can provide interferences between data points through reasoning, whereas graph databases are better optimized for graph traversals.

In this system, RDF triple stores have been chosen as the database for the shop floor Linked Data. The reason for the choice is greater similarities with the semantic web, which corresponds more to the original sense of Linked Data.

Shop Floor Ontology

With regard to the evaluation scenario, a simple shop floor ontology has been created covering the needs for this scenario. Because of compatibility with RDF triple stores and human readability, OWL has been chosen as the description language for the shop floor ontology. The ontology uses only the prefix namespaces owl, rdf, rdfs and dc to keep the scope small for prototyping.

For scenarios that require more complex ontologies the following work [Kö17], developed for ScaleIt, can be conducted. It includes the design of an ontology by the structures of manufacturing companies. The design concluded by an implementation [sho], which examines the possibilities of OWL in a shop floor scenario.

Listing 5.1 shows an excerpt from the shop floor ontology description document.

Entities. Entities that are part of this scenario on the shop floor are production lines and along each production line several machines and sensors. Each production line processes different products. These products are further processed by machines and scanned by various sensors after each production step. Each product holds several components, which represent the equivalent of PCB components. During each scan, a sensor creates a scan record for the complete product in addition to individual scan components, which describe the scan results for a specific product component.

Figure 5.1 shows a graph of the shop floor entity structure surrounding one production line.

Meta-Entities. For each shop floor entity there exists at least one meta-entity which is a virtual object holding meta-information about the partner entity like firmware version numbers, data format specifications and other types of meta information. Meta-Entities are not used in the evaluation scenario to limit the scope of entities participants have to manage.

Relationships on the shop floor. To model relationships between entities on the shop floor the 'isComponentOf' attribute, the 'hasComponent', the 'dataProducedBy' and other attributes have been used to describe relationships of different shop floor entities. To restrict the scope of the evaluation scenario the amount of relationships has been limited to

```
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix shopfloor: <http://scale-it.org/shopfloor#> .
@prefix dc:   <http://purl.org/dc/elements/1.1/> .

<http://scale-it.org/shopfloor>
       a              owl:Ontology ;
       dc:description "Shopfloor ontology for shop floor entities" ;
       dc:title       "Shopfloor ontology" .

shopfloor:data a    owl:Class ;
       rdfs:comment "The class of shopfloor data models" ;
       rdfs:label   "The shopfloor data" .

shopfloor:sensordata a owl:Class ;
       rdfs:comment    "The class of shopfloor Sensor Data " ;
       rdfs:label      "Shopfloor sensor data" ;
       rdfs:subClassOf shopfloor:data .

shopfloor:data_unit a owl:DatatypeProperty .

shopfloor:data_value a owl:DatatypeProperty .

shopfloor:entity a  owl:Class ;
       rdfs:comment "The class of shopfloor entities" ;
       rdfs:label   "The entity name" .

shopfloor:sensor a     owl:Class ;
       rdfs:comment    "The class of shopfloor sensors" ;
       rdfs:label      "Shopfloor sensor" ;
       rdfs:subClassOf shopfloor:entity .
...
```

Listing 5.1: Excerpt from ontology description document in in the OWL format.

twenty.

SPARQL Queries

To retrieve information about the shop floor for visualization purposes, the SPARQL query language has been chosen because of its high usage on the web and in Linked Data applications. Listing 5.2 shows a sample SPARQL query to retrieve information about the sensor with the id '101'. The resulting response in JSON format of the sample query is shown in listing 5.3.

5.3 Supporting the Shop Floor Workflow

The main goal of the shop floor system is to acquire raw sensor data and to transform it into Linked Data. This Linked Data can then be queried by using SPARQL queries like the shown in listing 5.2. The visualization of the SPARQL query response in listing 5.3 is then presented to users in a web-interface.

To provide a base structure for the system architecture and to define the specific system requirements of the system, the system is split into three parts. Each part covers one of

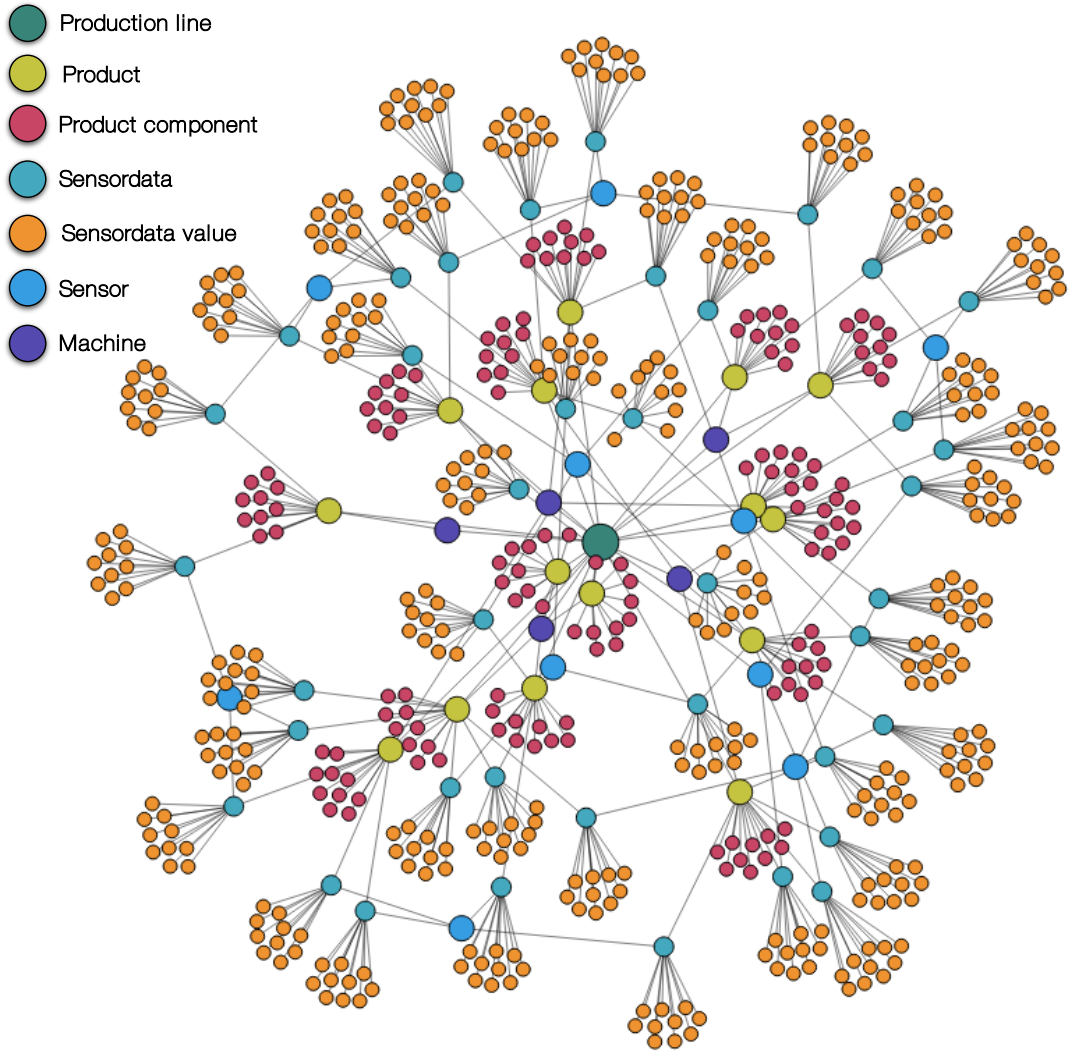


Figure 5.1: Shop floor entity structure surrounding one production line.

```
1  PREFIX shopfloor:
2  <http://scale-it.org/shopfloor#>
3
4  SELECT ?sensor ?product WHERE {
5    {
6     ?sensor a shopfloor:machine.
7       ?sensor shopfloor:id '101'.
8       ?sensor shopfloor:version ?sensor_version .
9     ?product shopfloor:produced_by ?sensor
10  }
11
12 }
```

Listing 5.2: SPARQL query for information about the sensor with id '101'.

the system's functionality. The three system functionalities are Linked Data acquisition, Linked Data storage and Linked Data visualization. Each part has its own set of specific tasks.

```
{
  head: {
    vars: [ "sensor" , "product" ]
  } ,
  "results": {
    "bindings": [
      {
        sensor: { type: "uri" , value:
            "http://scale-it.org/shopfloor#machine101" } ,
        product: { type: "uri" , value:
            "http://scale-it.org/shopfloor#product101" }
      } ,
      {
        sensor: { type: "uri" , value:
            "http://scale-it.org/shopfloor#machine101" } ,
        product: { type: "uri" , value:
            "http://scale-it.org/shopfloor#product109" }
      } ,
      {
        sensor: { type: "uri" , value:
            "http://scale-it.org/shopfloor#machine101" } ,
        product: { type: "uri" , value:
            "http://scale-it.org/shopfloor#product1010" }
      } ,
      {
        sensor: { type: "uri" , value:
            "http://scale-it.org/shopfloor#machine101" } ,
        product: { type: "uri" , value:
            "http://scale-it.org/shopfloor#product1015" }
      }
    ]
  }
}
```

Listing 5.3: Result of the SPARQL query for information about the sensor with id '101'.

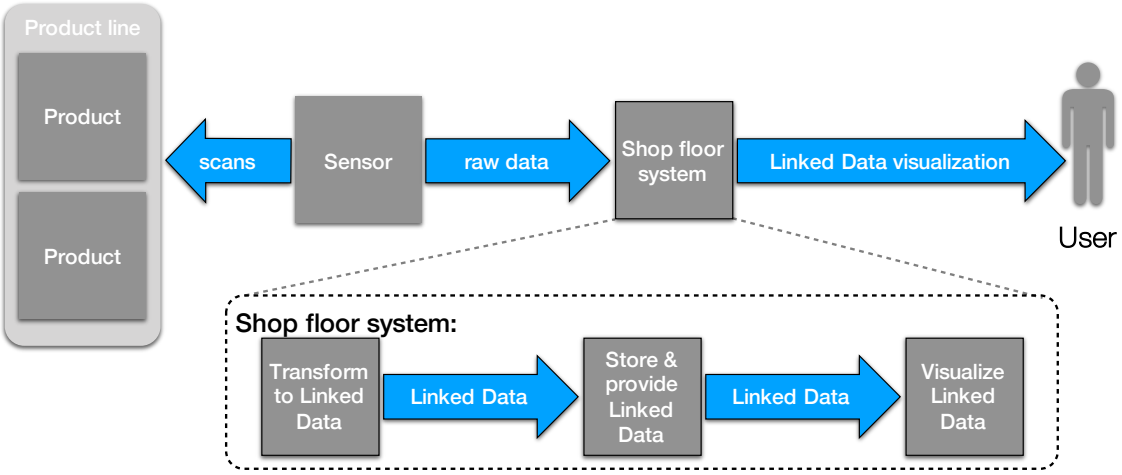


Figure 5.2: Each part of the shop floor system covers one functionality. They are Linked Data acquisition, Linked Data storage and Linked Data visualization.

1. **Linked Data acquisition.** Generate Linked Data representations from acquired raw shop floor sensor data.
2. **Linked Data storage.** Provide a storage API for storing and querying Linked Data representations.
3. **Linked Data visualization.** Create interactive visualizations of shop floor representations.

Linked Data Acquisition

The two main responsibilities of Linked Data acquisition are the acquisition of raw data and generation of Linked Data.

1. **Acquisition:** Receiving and validating incoming raw sensor data.
2. **Generation:** Generate Linked Data representation from acquired sensor data.

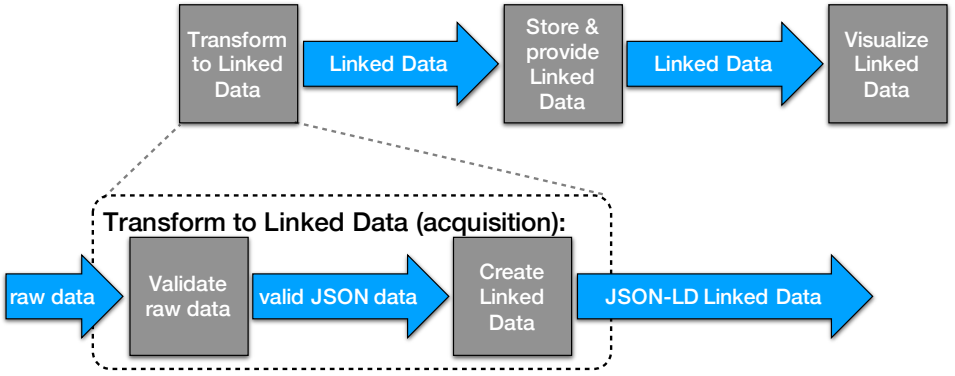


Figure 5.3: Raw data acquisition and Linked Data generation are handled by the acquisition part of the system.

Linked Data Storage The two main responsibilities of Linked Data storage are deposition and provision of Linked Data.

1. **Deposition:** Receiving generated Linked Data and store as triple in the RDF triple store.
2. **Provision:** Provide a Linked Data SPARQL query interface that lets services perform queries on the stored Linked Data.

Linked Data Visualization The main responsibilities of the Linked Data visualization are the provision of Linked Data visualizations and management of user contexts, as displayed in figure 5.4.

1. **Visualization provision:** Provide visualizations of shop floor entity structures inside of visualization apps.
2. **User context management:** Manage interaction between different shop floor visualization app instances.

5.4 System Requirements

The previous analysis of individual system parts will be used to derive the functional requirements and non-functional requirements of the system. Functional requirements describe the functionalities the system must be able to deliver. Non-functional requirements

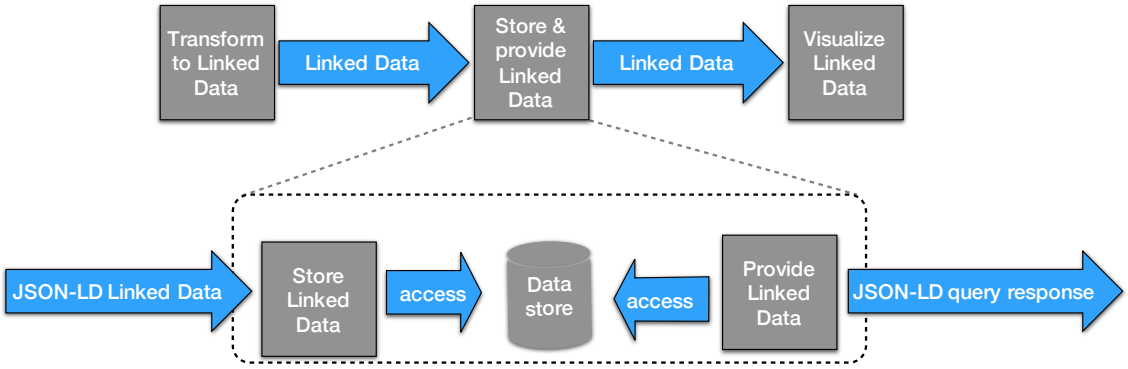


Figure 5.4: Linked Data deposition and Linked Data provision are handled by the storage part of the system.

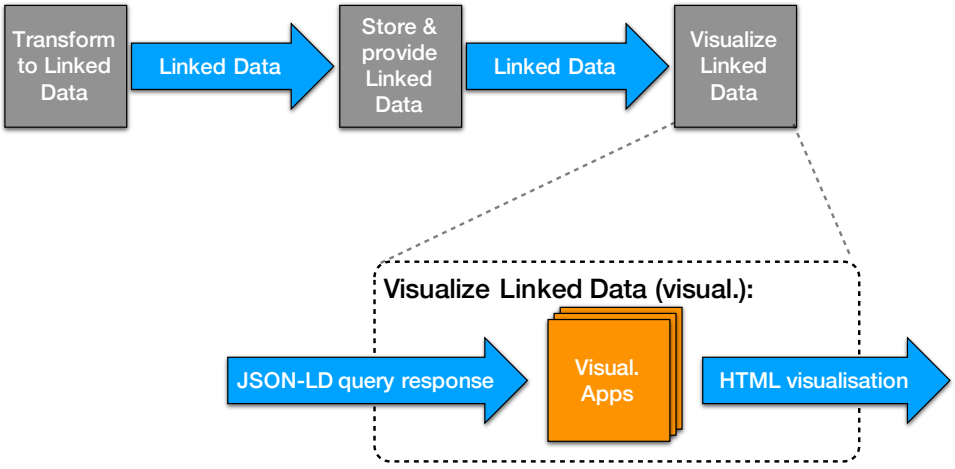


Figure 5.5: The basic interactions of the data visualization workflow: Visualization apps query Linked Data, which is being used to create visual representations of the shop floor entity structures. Users can interact with this visualizations inside visualization apps by using a web-interface.

describe how the system has to deliver these functionalities.

Functional Requirements for Acquistion

FF01: Receive raw sensor data from sensors.

FF02: Validate received sensor data.

FF03: Parse valid sensor data and enrich with provenance information.

FF04: Map parsed sensor data onto belonging Linked Data ontology class.

FF05: Serialize mapped data in a valid Linked Data exchange format.

FF06: Send serialized sensor Linked Data to the storage component.

Functional Requirements for Storage

FF07: Receive sensor Linked Data serializations from acquisition component.

FF08: Validate serializations.

FF09: Parse serializations.

FF10: Insert parsed serializations into Linked Data storage.

FF11: Allow SPARQL queries from authorized services.

FF12: Provide options for SPARQL queries via a web-interface.

FF13: Allow extending shop floor classes and attributes with a web-interface editor.

FF14: Allow editing of shop floor classes and attributes with a web-interface editor.

Functional Requirements for Visualization

FF15: Provide dashboard user interface via web-interface.

FF16: Provide different shop floor visualizations accessible through a user dashboard.

FF17: Provide user interaction and navigation inside visualizations.

FF18: Start and quit individual user sessions.

FF19: Add and remove instances of shop floor visualizations from and to user sessions

FF20: Visualizations receive and react in predefined ways to messages from other visualizations in the same user context.

FF21: Provide the possibility for visualizations to react to user interactions by publishing messages to other visualizations in the same user context.

FF22: The user can subscribe and unsubscribe individual visualizations from the current user context(toggle message interaction).

FF23: Each user session contains its own user context. Each context sends messages published by visualizations to all visualizations subscribed to the user context.

FF24: Make a new query request to the Linked Data storage at the start of each visualization instance.

Non-functional Requirements

While functional requirements ensure the system is working at all, the non-functional requirements define qualitative ways, in which the system has to fulfil its functionalities. This ensures proper functioning of the system on the shop floor. The most important aspects of this are error resilience, error recognition and interaction between service and service and service and user.

Non-functional Requirements for every Service

NF01: Respond to requests within 500ms.

NF02: Restart failing services within 2000ms.

Non-functional Requirements for Acquisition

NF03: Validate incoming sensor data requests within 500ms.

NF04: Transform valid incoming sensor data within 10000ms.

NF05: Respond to requests within 500ms.

Non-functional Requirements for Storage

NF06: Respond to query requests within 200ms.

NF07: Access data with a maximum latency of 600ms.

Non-functional Requirements for Visualization

NF08: Start new visualization apps within two seconds.

NF09: Map the shop floor structure in the visualization structure of app visualizations.

NF10: Apps are easy to combine and to rearrange.

5.5 System Architecture

The system architecture consists of modular **apps** collaborating by using HTTP and REST. The design is oriented to the principles of the 12-factor app [Ler14] and the reactive manifesto [BFKT14]. Every **app** publishes its own API as a representation of the services it provides. Other **apps** can then call this services through HTTP requests.

The structure of system components is oriented to the ScaleIT architecture. Each **app**, as defined [MCS+16] for the ScaleIT system, consists of one or more **services** deployed in containers. Containers are preferred instead of VMs for performance reasons.

Services serve a limited scope of functionalities, like sending requests to a neighbouring service or accepting requests sent to the REST API).

Figure 5.7 displays a typical shop floor app, which is oriented towards the ScaleIT-App architecture.

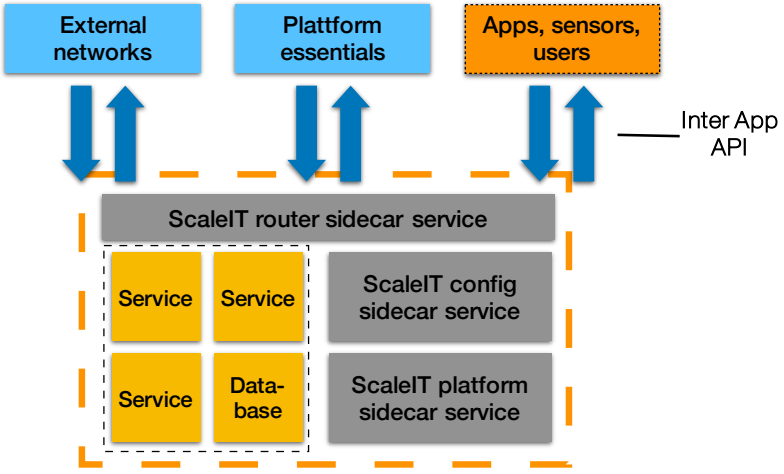


Figure 5.6: Anatomy of a shop floor app - exemplary [sca]

Services are designed to deliver a limited scope of business(yellow) or technical functionality(grey). Every app has to fulfil the following overall requirements.

Requirements on an App Level

The following requirements must be fulfilled by each app in the system.

1. An app must expose a REST API.
2. An app must communicate via Event-driven messages.
3. An app must use a unified data exchange format.
4. An app must fulfil all four Linked Data principles.

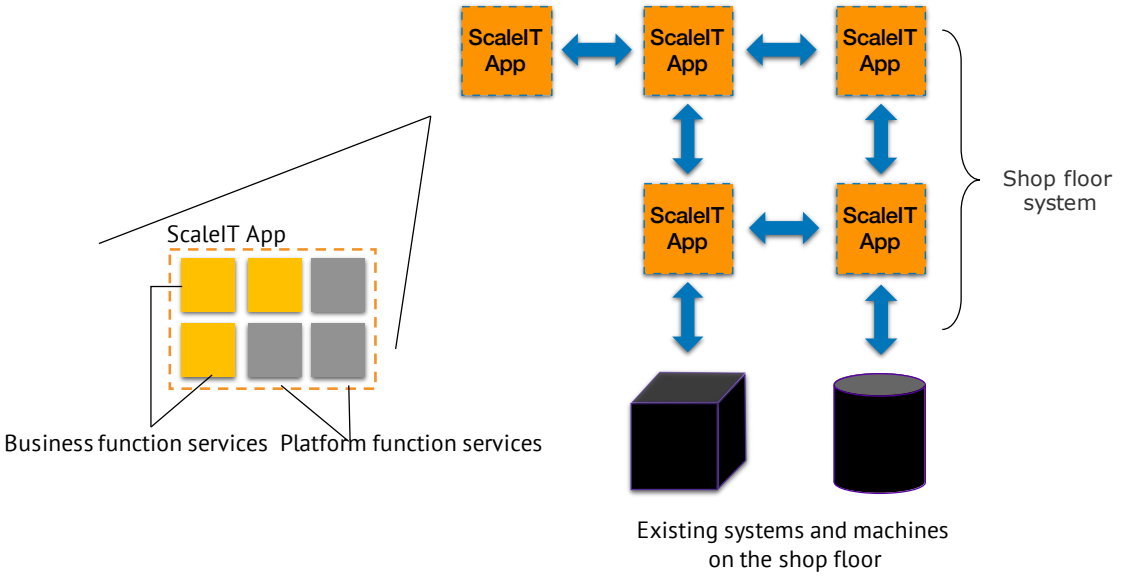


Figure 5.7: Shop floor system = shop floor apps + existing devices and machines [sca]

5. An app must support the highest LEVEL of RMM.
6. An app must provide an HTML UI at its unique URL.

The first, fourth and sixth item is best practice when developing for the web of things as described in [W3C].

Requirements on a Service Level

The following requirements must be fulfilled by each service in the system.

1. A service must own a unique code base.
2. A service must own explicit dependencies.
3. A service must store its configuration in environment variables.
4. A service must run inside a container.

These requirements are stated in the 12-factor app manifesto [Ler14], [twe].

Scoping of Apps and Services

The process of defining the scope of each service is called scoping. The main driver for scoping are business capabilities. They describe something an organization does to contribute to its business goals. An example is the transformation of incoming raw sensor data into their Linked Data representations. This contributes to the business goals of the shop floor, by helping to improve maintenance and monitoring processes.

Secondary driver are technical capabilities, which themselves don't support business goals directly but simplify and support other services supporting business capabilities. This can be a database handler managing technical details of handling complex queries by providing a simplified interface for services depending on the database.

To differentiate the two types of services in figure 5.6 and 5.7, services supporting business capabilities are coloured yellow and services supporting technical capabilities are be coloured grey. Services of both types work closely together to achieve an overall goal are combined in apps, as displayed in figure 5.6.

The requirement of adaptability and re-usability implies the architecture for individual service modules to be as small as possible and as big as necessary. Every part of a service with a high probability to be reused later is extracted into its own service scope. This implies the app design presented in figure 4.7

In the next section, the functional and non-functional requirements previously defined will be used to scope the services of each shop floor system part. Each service is defined by a related set of functional requirements.

Design of Acquisition Services

The functional requirements of acquisition are mapped onto two services, the sensor API service and the Linked Data mapping service. When designed the architecture of the acquisition services, the four phases described by [PSHS10] et al. are used as an orientation for the data flow design.

Like [PSHS10] the shop floor system receives raw sensor data from outsides sources like sensors. This raw sensor data is then parsed into a structured format, in our case JSON(Patni uses XML). In the third phase, Linked Data is generated in the JSON-LD(Patni uses RDF-XML) file format. In the fourth phase, the generated Linked Data is stored in a Linked Data storage.

Figure 5.8: The data acquisition workflow of the shop floor architecture is oriented towards the four phases described by [PSHS10].

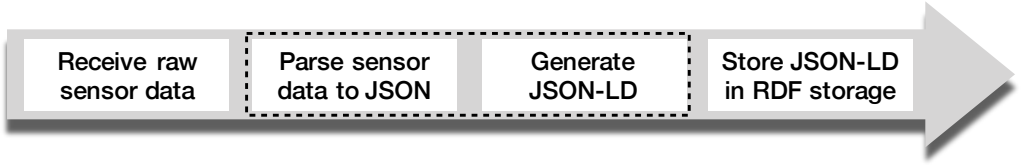


Figure 5.9: The data acquisition workflow of the shop floor architecture.

1. **Sensor API service.**

   FF01 Accept incoming raw sensor data.

   FF02 Register sensor data recording.

   FF03 Validate correctness of sensor data. Sensor data is correct if it exists a valid mapping onto a shop floor entity class.

2. **Linked Data mapping service.**

   FF04 Parse correct raw shop floor data.

   FF05 Create Linked Data mapping.

   FF06 Delegate storing Linked Data mapping to storage part.

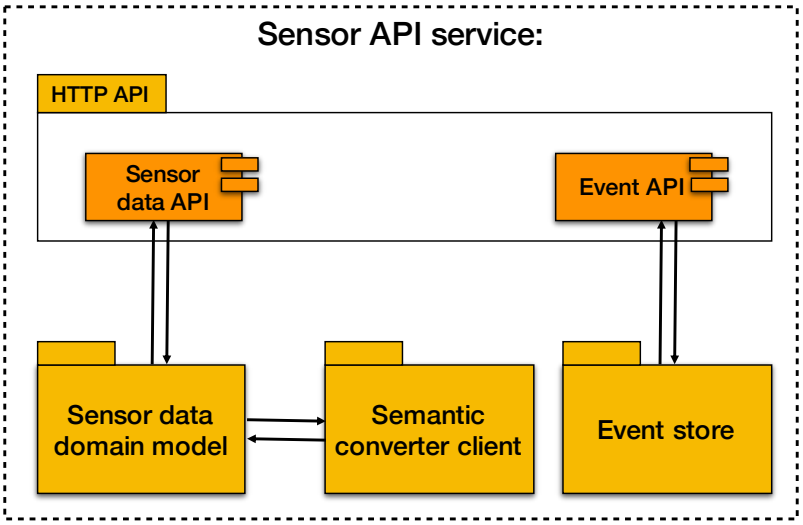


Figure 5.10: Sensor API service package architecture

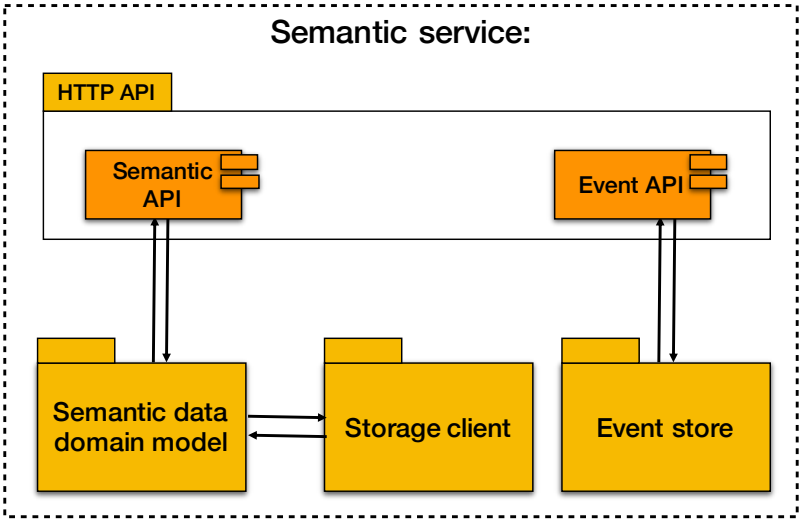


Figure 5.11: Semantic service package architecture

The package architecture of the sensor API service and the semantic service is shown in figure 5.10 and 5.11.

These two services are combined in the semantic app.

Design of Storage Services

The functional requirements of storage are mapped onto the following two services:

1. **HTTP interface service.**

   FF07: Accept JSON-LD Linked Data serializations.

   FF08: Validate serializations.

   FF09: Parse serializations.

   FF10: Insert parsed serializations into Linked Data storage structure.

2. **SPARQL interface service.**

FF11: Accept SPARQL queries through a HTTP interface.

FF12: Accept SPARQL queries through a web-interface.

FF13: Send query responses with JSON-LD representation of the query results.

The package architecture of the HTTP service and the SPARQL service is shown in figure 5.12 and 5.13.

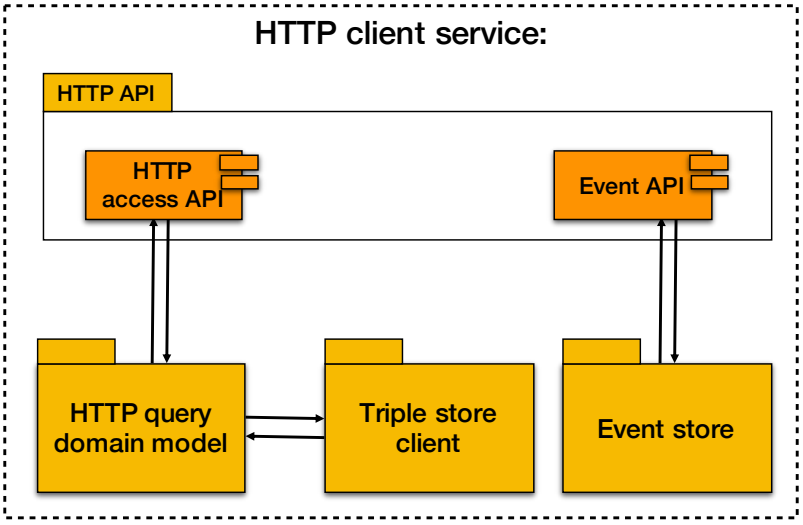


Figure 5.12: HTTP interface service package architecture

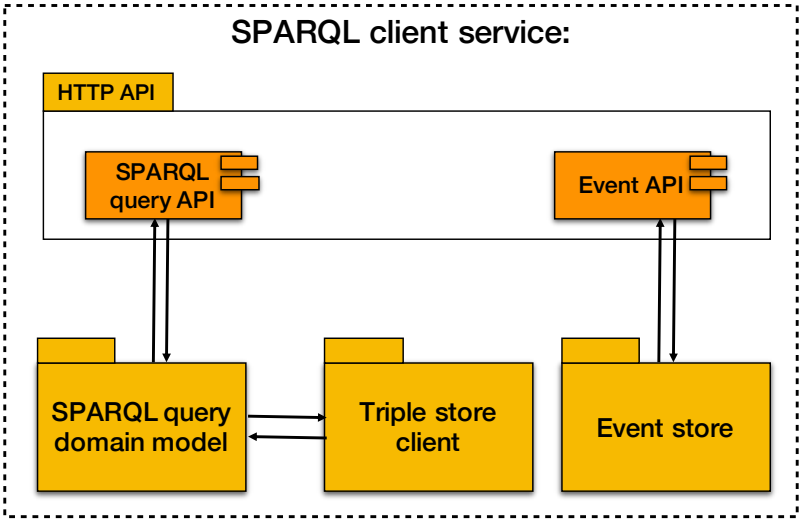


Figure 5.13: SPARQL interface service package architecture

These two services are combined in the storage app.

The system part supporting visualization is divided into two parts. Individual visualization apps and a command center managing user workflows and orchestration of visualization apps.

Design of Command Center Services

The command centre service group consists of three services:

1. **User context service.**

   FF20: Visualizations receive and react in predefined ways to messages from other visualizations in the same user context.

   FF21: Visualizations can react to user interactions by publishing messages to other visualizations in the same user context.

   FF22: The user can subscribe and unsubscribe individual visualizations from the user context(toggle message interaction).

   FF23: Each user session contains its own user context. Each context sends messages published by visualizations to all visualizations subscribed to the user context.

2. **Dashboard service.**

   FF15: Provide dashboard HTML representation in a web browser.

   FF16: Provide user access for visualization apps in the dashboard view.

   FF17: Send, receive and process visualization management user commands.

3. **Visualization app manager service.**

   FF19: Start and quit individual user sessions.

   FF20: Add and remove instances of shop floor visualizations from and to user sessions.

The package architecture of the user context service, the dashboard service and the visualization app manager service is shown in figure 5.14,5.15 and 5.16.

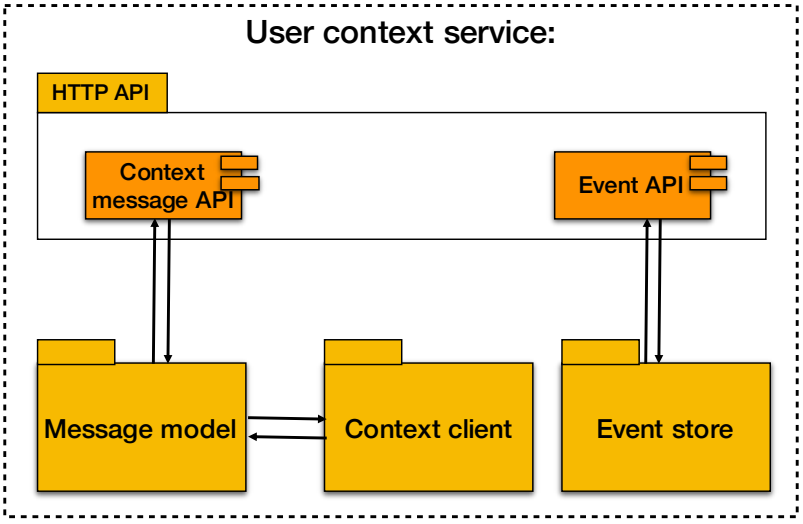


Figure 5.14: User context service package architecture

These three services are combined in the command center app.

Design of Visualization App Services

The visualization app service group consists of three services:

1. **User context client service.**

   FF20: Visualizations receive and react in predefined ways to messages from other visualizations in the same user context.

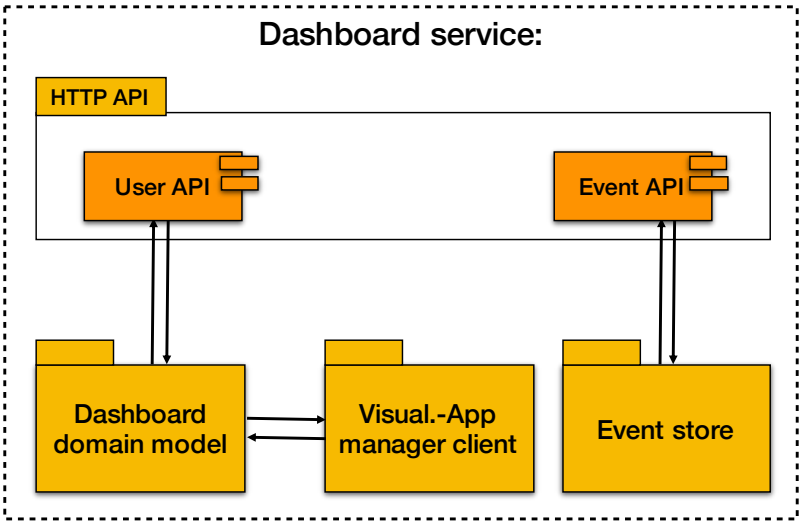


Figure 5.15: Dashboard service package architecture

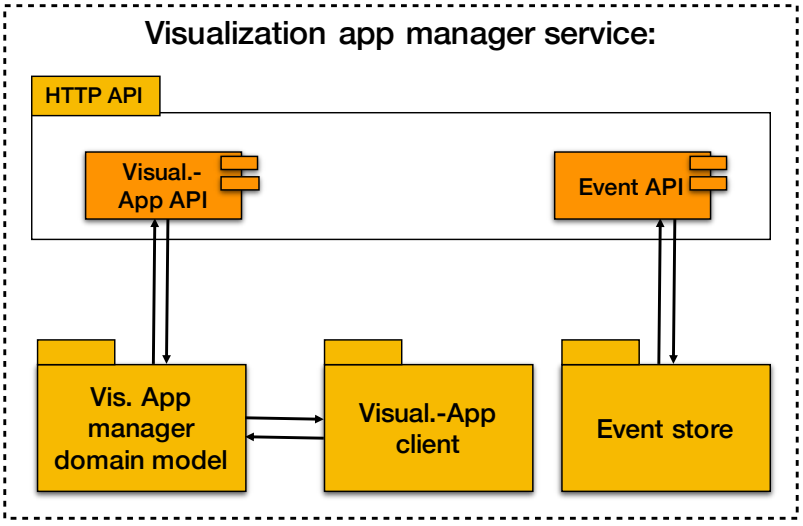


Figure 5.16: Visualization app manager service package architecture

FF22: The user can subscribe and unsubscribe individual visualizations from the user context.

FF23: Each user session contains its own user context. Each context sends messages published by visualizations to all visualizations subscribed to the user context.

2. **SPARQL client service.**

FF24: Pull a new instance of the shop floor at the start of each visualization instance.

3. **Visualization client service.**

FF17: Provide user interaction and navigation inside visualizations.

FF21: Provide the possibility for visualizations to react to user interactions by publishing messages to other visualizations in the same user context.

The package architecture of the visualization service, the user context client service and the SPARQL client service is shown in figure 5.17, 5.18 and 5.19.

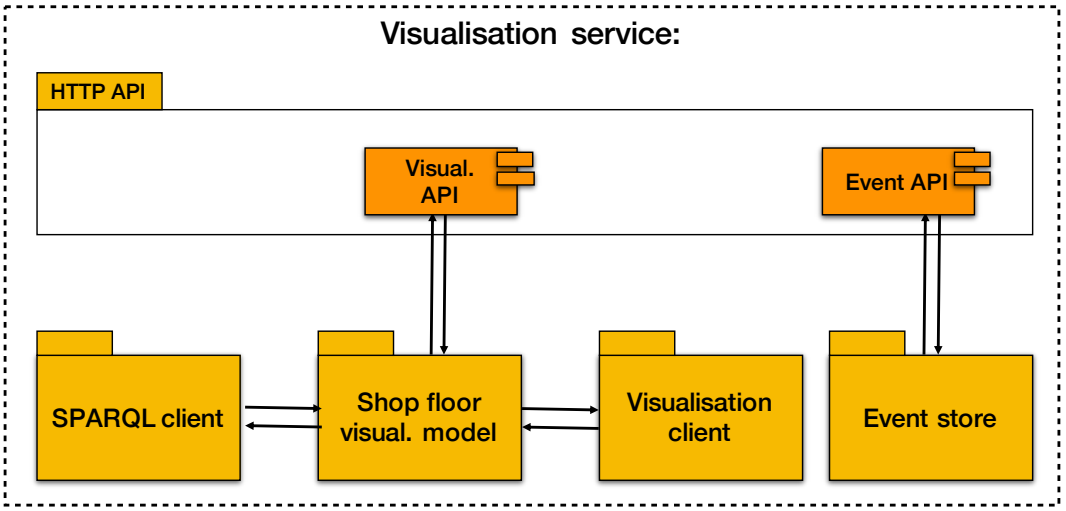


Figure 5.17: Visualization service package architecture:

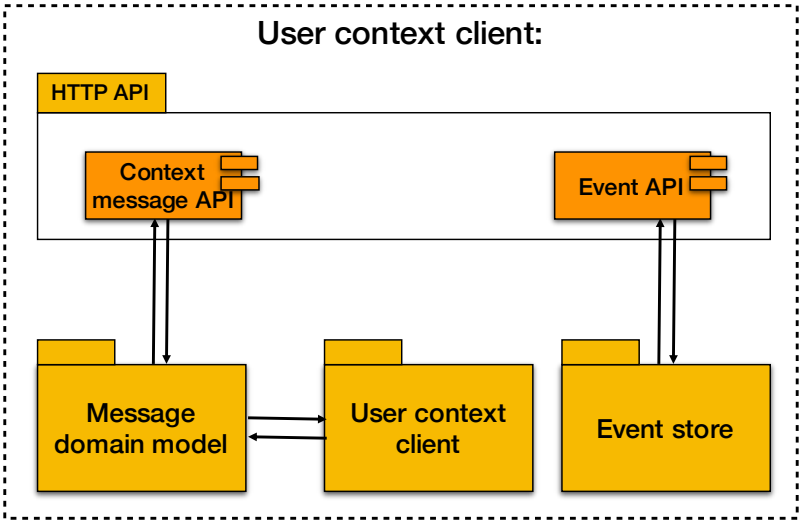


Figure 5.18: User context service client package architecture:

These three services are combined in the visualization app.

Service Design Pattern

These services of the architecture will be individually deployed by using containers. To provide encapsulated log- and monitoring-functionality for each service container management design patterns are utilized. Burns et al. [Bur16] defined various patterns for micro-service architectures, from which we will apply a subset.

The following micro-service patterns are utilized in the system design:

1. Side-car pattern to record events.
2. Adapter pattern for logging and monitoring.

Sidecar pattern. The sidecar container enhances and extends a service. Even if its possible to directly implement the functionality of the sidecar service into the main service, there are advantages of separating them [Bur16].

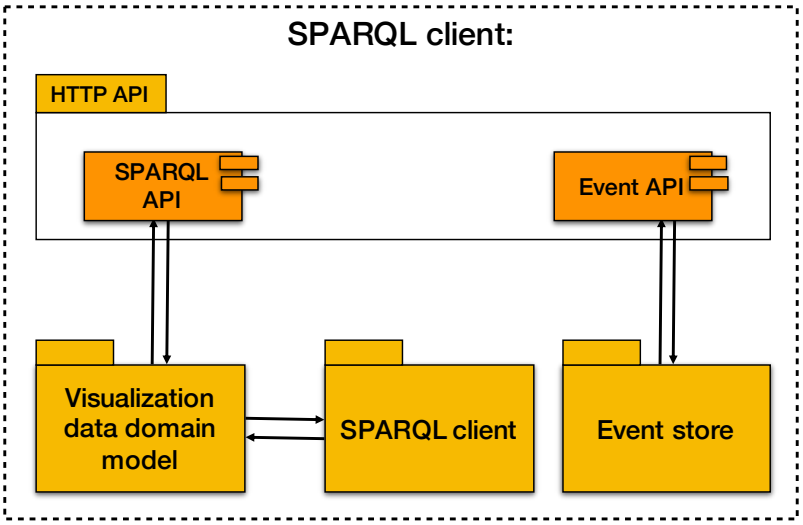


Figure 5.19: SPARQL client service package architecture:

- Re-usability of services.
- Individual packaging allows for independent development.
- Individual configuration possible(e.g. for hardware utilization improvements).

Adapter pattern. The adapter pattern presents external services with a simplified view of a service's interface. This makes it easier for monitoring applications to monitor services because now they can interact with a consistent interface, which doesn't require additional modifications. The benefits of using the adapter pattern are [Bur16].

- Inside-Outside and Outside-Inside communication can be generalized in development.
- When changing data formats only the adapter has to be replaced.

Applying patterns. In our example, we use the sidecar pattern to pair services with an event service, which periodically synchronizes the event store with the main service. The monitoring service, which monitors the current health status of the main service, is implemented by using the adapter pattern.

Event-driven Messaging

For communication between services the design paradigm event-driven messaging has been selected. Alternative approaches are command-based messaging. Event-driven messaging is chosen because it provides a more loosely coupled than its alternatives [BFKT14]. In addition to providing loose coupling, event-driven messaging s asynchronous messaging. This increases the fault tolerance and resilience because the handling of events does not have to occur immediately like in command based and query based communication.

Raising events. The raising of events is realized by applying the side-car container pattern. In figure 5.20 a business function service, namely the main service raises an event eg. successful validation of incoming sensor data. The raised event gets captured by the event store service, which then stores the event in an event store.

User context based event collaboration. To allow apps to interact with each other to use them together in complex workflows, the concept of user contexts is applied. A user context is defined by all apps subscribed to a specific user context server, which is run

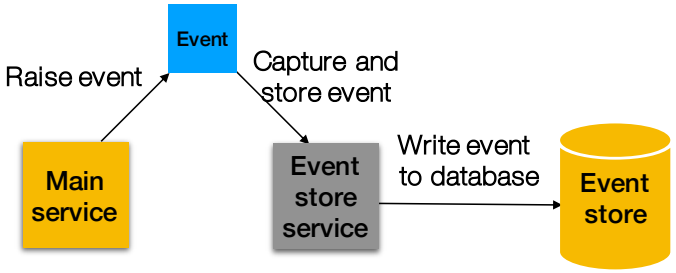


Figure 5.20: Raising of events using the side-car container pattern.

by the user context server service. The user context server itself is a part of a command centre app.

Inside a user context, multiple instances of different visualization apps can now interact with each other by subscribing and publishing to the same user context. This user contexts can be used by one or more shop floor users from different user clients.

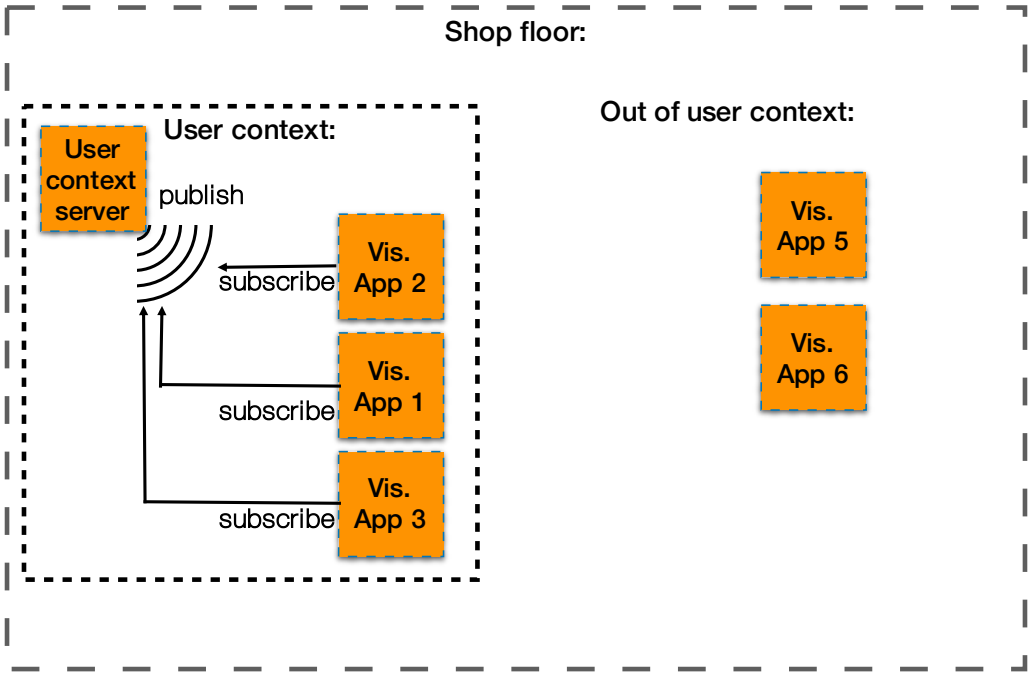


Figure 5.21: Visualization apps inside a user context subscribe to a user context server, which lets them receive messages about user events from visualization apps in the same user context.

5.6 User Interface

The user interface(UI) of the shop floor system consists of a dashboard allowing users to start, quit, modify and arrange multiple different visualization app instances.

To structure the development of the user interface, functional specifications for the dashboard and each visualization app-type have been created. For each element, a user interfaces a sketch is created, which is then used as a basis for the user interface implementation.

Dashboard User Interface

The task of the visualization dashboard is to provide a starting platform, from which users can start, quit, modify and arrange multiple different running visualization app instances. To fulfil the systems requirements, is it important to allow dynamic changes in the arrangement of apps.

Subscribing to User Context

The dashboard must provide interaction toggles for each visualization app controlling the subscribing and un-subscribing of visualization apps to the current user context.

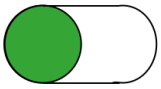


Figure 5.22: Button toggle to subscribe and un-suscribe to current user context.

A sketch of the dashboard UI is shown in figure 5.23.

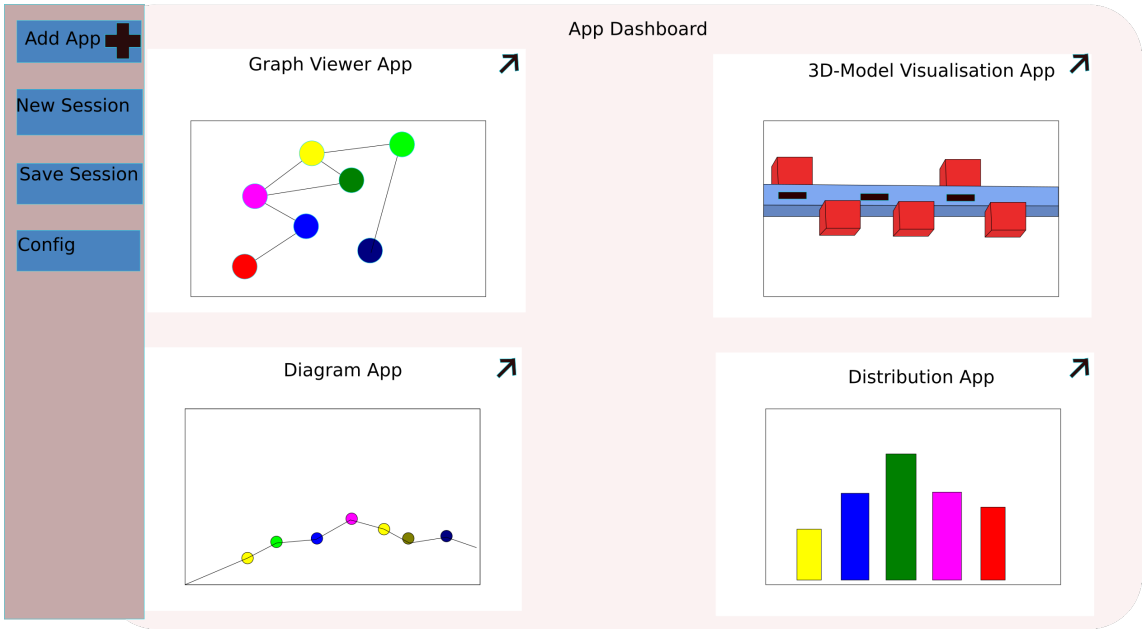


Figure 5.23: Sketch of the dashboard UI, on which visualization apps can be arranged.

User Interfaces of Visualization Apps

For the given shop floor scenario four visualization apps have been developed providing different visualizations for the user to interact with shop floor Linked Data.

Graph view app user interface. The graph view app displays a graph representation of the shop floor and an info box with additional information about the current selection.

The graph view app allows the user to perform the following actions.

1. Collapse and expand children nodes with a double-click.
2. Click nodes and links to display additional information.
3. Show and hide link-labels.
4. Show and hide nodes-labels.

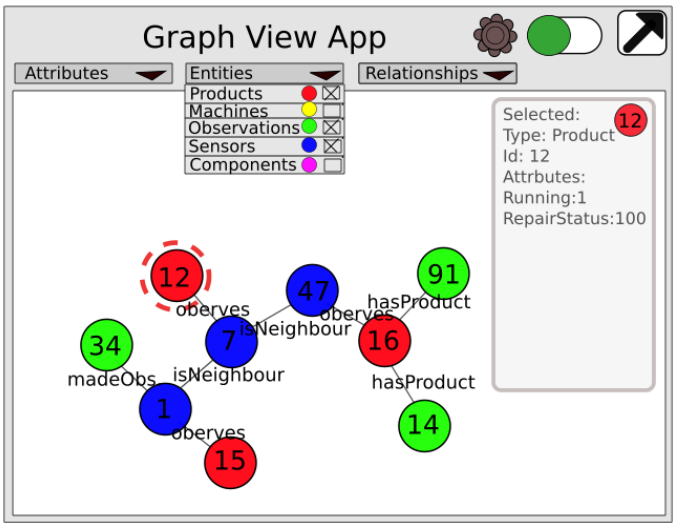


Figure 5.24: Sketch of graph view app UI

5. Disable and enable user context.

List view app user interface. The list view app allows to dynamically search in different types of shop floor entities for specific values or patterns. The list view app then filters the current entity selection after current search pattern.

The following actions are available to the user in list view app:

1. Search for relationships between shop floor entities
2. Filter for attributes, names, ids and other attributes.
3. Filter for timestamps or time intervals.
4. Disable and enable user context.

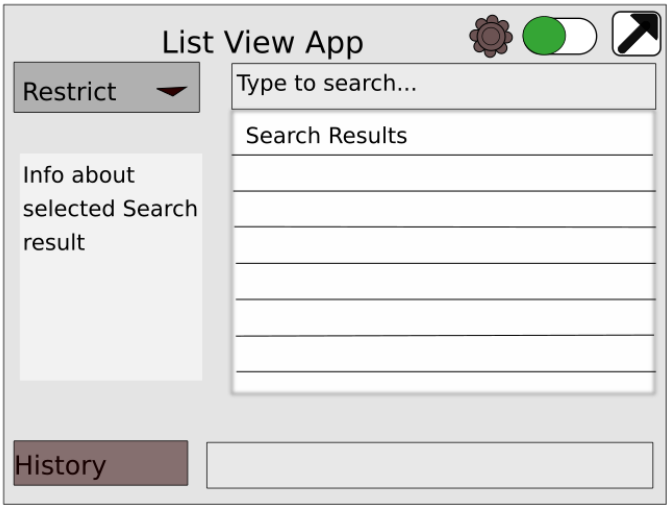


Figure 5.25: Sketch of list view app UI

3D view app user interface. The 3D view app creates a three-dimensional representation of the shop floor with the possibility of highlighting entities in different states by using colour-highlighting.

When using the 3D view app the user can perform the following actions.

1. Click individual objects in the visual representation to display further information about them. The information is displayed as an overlay on top of the 3D view.
2. Navigate along different production lines or along the currently selected one.
3. Disable and enable user context.

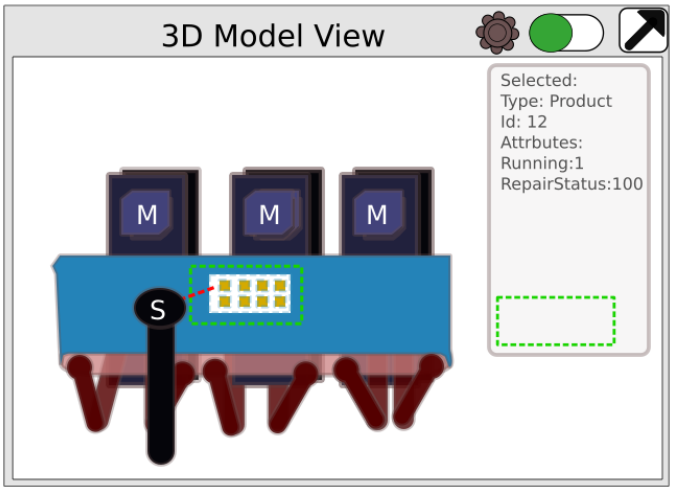


Figure 5.26: Sketch of 3D view app UI

Tree view app user interface. The tree view provides a tree representation of selected shop floor entities using the selected entity as the root of the tree. The user can navigate these tree visualizations from the root to the desired branch or leaf.

When using the tree view app the user can perform the following actions.

1. Hide and show individual branches.
2. Collapse or expand the entire tree.
3. Disable and enable user context.

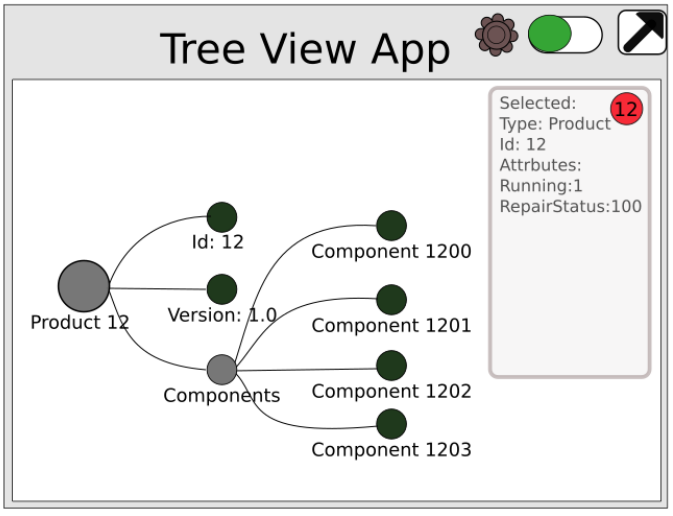


Figure 5.27: Sketch of tree view app UI

Note app user interface. A note app is developed to support participants in the evaluation study to keep track of their task results. When the note app is subscribed to a user context, the user can automatically insert the name of the most recently selected entity.

The following actions are available for users in the note app.

1. Take notes
2. Save notes
3. Directly insert the currently selected entity

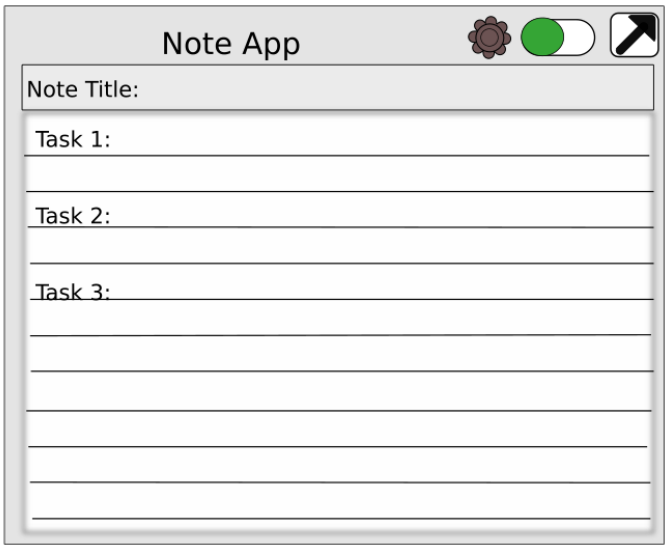


Figure 5.28: Sketch of note app UI - Note: Note app is only used during evaluation.

This user interface, including all visualization apps with the exception of the 3D view app, are evaluated in a user study. The 3D view app is not included in the evaluation process because it is not suitable for the tasks chosen to be performed during the evaluation.

The following section describes the design process of this user study.

5.7 Evaluation Design

To ensure the system design matches the requirements and to test the usability of the system in performing typical shop floor tasks a user study has been conducted. The goal is to analyze how the system, including the user interface, behaves during simulated user operations on the shop floor.

The objective of this study is also to evaluate the user experience of individual visualization functionalities. And to identify the principles and available technologies to build Linked Data based shop floor information systems, which are suitable for this scenario.

The following criteria have been evaluated.

1. Does the system support users in maintenance and monitoring tasks?
2. Is the system easy to use for users, familiar with the shop floor environment. And if not, to what degree do they have to be familiar?
3. Is 'intuitive' combination of apps possible?

4. Do apps support and complement each other?
5. Can users focus on the task at hand?

To achieve the assessment of criteria 3. to 5. users questionnaires covering SUS and UEQ will be used.

To compare the results with real-world shop floor applications and a handful of example tasks have been constructed. These tasks are simplified versions of possible real-world problems on the shop floor.

Study Composition

The following sections describe individual steps of the study a participant will go through.

User environment for the evaluation. The user working environment consists of a desktop machine with two 20", full HD monitors with one keyboard and one mouse. Before the evaluation starts a web browser is opened displaying an empty dashboard view.

Introduction to the study. At the beginning the participant is introduced to his tasks: "You use a web application to use shop floor data to solve a handful of simplified shop floor problems. Afterwards, you will fill in a questionnaire to evaluate your user experience."

Reading documentation before the tasks. Then the participants are handed a documentation the contains information about the scenario, their tools and a sample workflow with example tasks and the list of tasks they have to fulfil.

Adapting to the working environment. After reading the documentation the participants can make themselves accustomed to their working environment by trying out the visualization applications with the sample task. This has taken no longer than ten minutes. When they are ready they start with the first task.

During the tasks. For each task exists a time limit of ten minutes. If this limit is reached the participant has to go on to the next task. During the tasks, the participants are only allowed to ask questions regarding the use of the application but not regarding the handling of a task itself. During each task, the participant will write down her notes and results in the note app.

Final Evaluation

After the study process, the participant assesses her experience with the shop floor system through several questionnaires. In addition to SUS and UEQ, a handful of custom questions have been created, to gain more specific insight from participants.

To better understand the usage of individual apps and user contexts heat maps of each used visualization app have been recorded during each evaluation session. For this purpose a small heat map service will be implemented, recording heat maps for every visualization app. For the user contexts, each published message is recorded. Every time a shop floor entity gets selected inside a visualization app the apps sends a message to the user context, which publishes the message to all apps in the same context.

5.8 Conclusion

The architecture design described in this chapter is based on the principles of semantic web, micro-services and containerization. It is created by analyzing the two responsibilities of the shop floor system. First, to acquire provenance enriched sensor data. And second,

to provide a platform with a generalized API. This API allows for easy adaption of Linked Data-driven apps, customized for user needs.

One of the core principles of the architecture is event-driven communication. Event-driven communication allows the recording of everything that happens inside the system. This event-data can be used to test individual components. Or on a greater scale to the complete system in specific scenarios. Event-driven communication is also one of the system pillars to guarantee individual components are loosely coupled and therefore individually replaceable.

By creating user contexts user enable event-driven interactions between visualization apps. Visualization apps can be combined together to visualize information about shop floor entities. They can also combine acquired sensor data, physical and virtual event data of everything that happens on the shop floor. The visualizations are done in different wys, which may lead to better understanding of shop floor maintenance and monitoring tasks, the overall goal of the system architecture.

6. Implementation

This chapter describes the implementation process of the system architecture. This includes how different sets of libraries and frameworks have been used to realize the architecture designed in the previous chapter. For more technical information, Chapter 1.4 links to technical tutorials and actual source code in published repositories.

6.1 System Overview

Figure 6.1 gives an overview of the system architecture. It consists of four different apps. Every app is responsible for one part of the system's functionality. The semantic app generates Linked Data from raw sensor data, enriched with provenance information. The storage app handles the access to the Linked Data storage of the system. Different versions of visualization apps provide Linked Data visualizations via HTML. The command centre app manages user sessions and user contexts. It also provides a user dashboard used to start, arrange, manage and quit individual visualization apps.

Deployment Structure

The docker-compose tools are used for defining and deploying multi-container docker applications. They make use of the YAML file format to configure the services of an application. This services can then be started together with a single docker-compose command.

For deployment, the system architecture is mapped onto docker-compose units. One docker-compose unit for each app and one service for each docker container inside a docker-compose unit.

To run a multi-container application with docker-compose the following steps must be taken.

1. Define the environment of the involved services, by creating a Dockerfile for each service.
2. Define the services in your docker-compose file and add links between services via environment variables if necessary.
3. Run '$ docker-compose up' to run the app.

Listing 6.1 displays the docker-compose file of the semantic app. The line followed by 'links:' creates an entry in the hostname file of the respective service with the IP address of the service to link with.

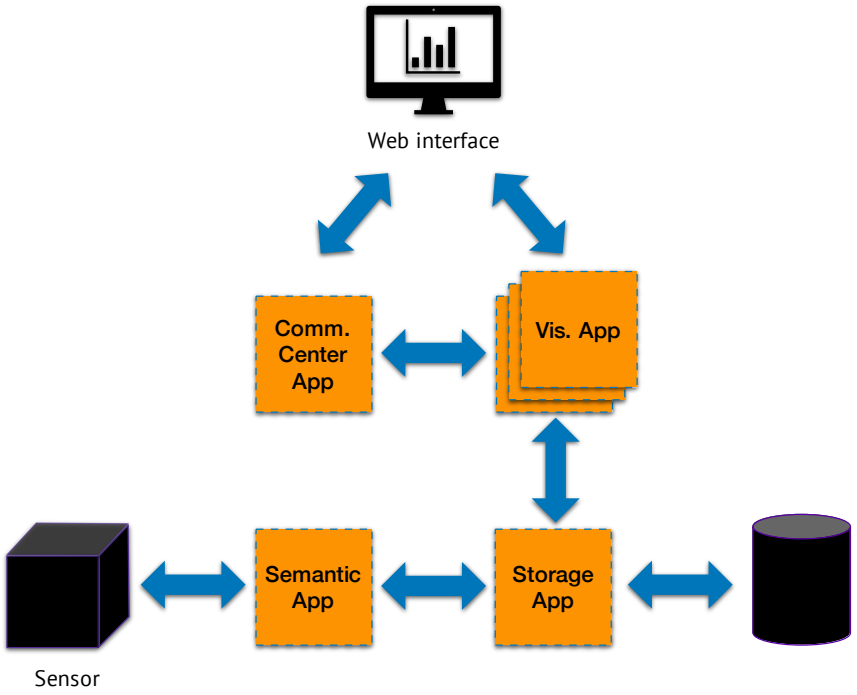


Figure 6.1: App architecture of the shop floor acquisition and visualization system.

```
version: "3.2"
  services:
    sensorAPI_service:
      build: ./sensorAPI/
      ports:
      - "5000:5000"
      volumes:
        - ./sensorAPI/code:/sensorAPI/code
        - ./sensorAPI/sensorAPI.csproj:/sensorAPI/sensorAPI.csproj
      links:
        - "semantic:semantic_service"
    semantic_service:
      build: ./semantic/
      ports:
      - "5050:5000"
      volumes:
        - ./semantic/code:/semantic/code
        - ./semantic/semantic.csproj:/semantic/semantic.csproj
      links:
        - "sensorAPI:sensorAPI_service"
```

Listing 6.1: Docker-compose file used to run the semantic app.

A Dockerfile contains the instructions needed to create the environment, in which a service will be deployed. Listing 6.2 shows the Dockerfile of the semantic API service, which uses the second version of the .NET SDK as the base environment.

```
1  FROM microsoft/dotnet:2-sdk
2  MAINTAINER Matthias Schedel <matze.schedel@gmail.com>
3
4  ENV ASPNETCORE_ENVIRONMENT="Development"
5  ENV DOTNET_USE_POLLING_FILE_WATCHER=1
6
7  ADD . /sensor_api/
8  WORKDIR /sensor_api/
9
10 RUN ["dotnet", "restore"]
11 RUN ["dotnet", "build"]
12
13 CMD ["dotnet","watch", "run"]
```

Listing 6.2: Dockerfile file used to create the sensor API service

6.2 Semantic App

The main focus of the semantic app is to be easily adaptable and extendable while being loosely coupled and individually replaceable. This ensures new types of sensors can be added quickly to the system by only replacing individual components.

Selected Technologies

To realize adaptability and extendability of the semantic app, specific technologies have been chosen allowing easy replacement and extension of individual parts.

DOT NET Core. DOT NET Core(.NET Core) is an open source cross-platform implementation of the .NET Standard. .NET core provides a subset of the functions available in the .NET Standard, specifying the .NET API of Microsoft's .NET framework. The choice of .NET is given by the responsible Institute to explore the possibilities of .NET for micro-service development of shop floor applications.

Nancy Fx and OWIN middleware. The open web interface for .NET(OWIN) abstracts from .NET servers and web applications. This allows the creation of a decoupled service architecture. OWIN is used to implement the ambassador design paradigm in a simple way to offer logging, monitoring and event-recording for all .NET based services.

Nancy is a lightweight framework for .NET, .NET Core and Mono for building HTTP-Services. It is designed to handle HTTP requests in a simple manner while providing the option to customize HTTP parameters easily. It is chosen because of the possibility of easy handling of HTTP requests, object serialization and object de-serialization.

Decoupling HTTP APIs with OWIN. Every incoming request from the HTTP API component in figure 6.2 is delegated to the standardized OWIN interface. The HTTP API component itself has no knowledge about the OWIN component. And the OWIN component only knows HTTP requests are received through the OWIN interface.

The standard OWIN interface functions here as an adapter for the HTTP requests between the HTTP API component and the OWIN component. This results in decoupling of OWIN component and HTTP API component [EP17].

Because individual components only know about their neighboring OWIN interfaces, middleware handlers can be installed in between two standard OWIN interfaces in the middle. This is displayed in figure 6.3.

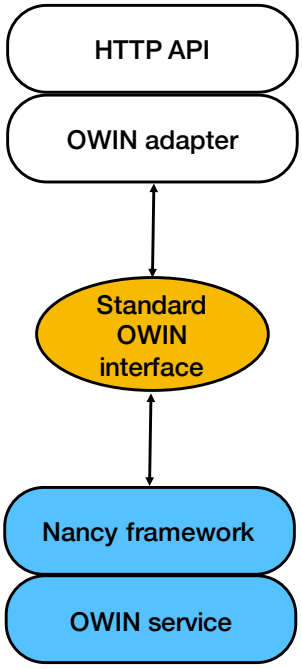


Figure 6.2: Delegating HTTP requests between OWIN component and HTTP API by using the standard OWIN interface [EP17].

Considered Linked Data Libraries in .NET

The following NET Linked Data libraries have been considered to use in the implementation of the .NET part of the system. Each of them has different strength and weaknesses, making them most suitable for particular tasks in the system.

Jena .NET. Jena .NET is a port of the popular Java-based Linked Data framework Apache Jena. It offers the possibility to create mashups of triples, query RDF models using SPARQL and to iterate through Models in .NET.

NewtonSoft JSON. NewtonSoft JSON is a flexible .NET library, providing serializing and de-serializing functionalities between .NET objects and JSON objects. It is chosen because it provides a higher performance than the native .NET JSON serializers and also provides the functionality of directly writing indented JSON code [NK13].

RomanticWeb. RomanticWeb is a native .NET ORM class solution for graph-based data. This is achieved by transparently mapping data models onto RDF statements. RomanticWeb also provides native querying abilities by using LINQ, which is then translated into the SPARQL protocol.

DotNetRdf. DotNetRdf supports managing, parsing, reading, writing and querying of RDF data. It also includes powerful APIs for working with RDF triple stores like Jena, Virtuoso and AllegroGraph. The library is also extensive, by allowing users to add their own features like custom RDF triple stores. It primarily operates on the triple and graph level. At the writing of this thesis, it provided only limited interference support and no direct support for OWL.

Selecting Libraries for Linked Data Handling

For the task of parsing sensor data, the NewtonSoft JSON library has been selected, because of its flexible object serializing and de-serializing functionalities.

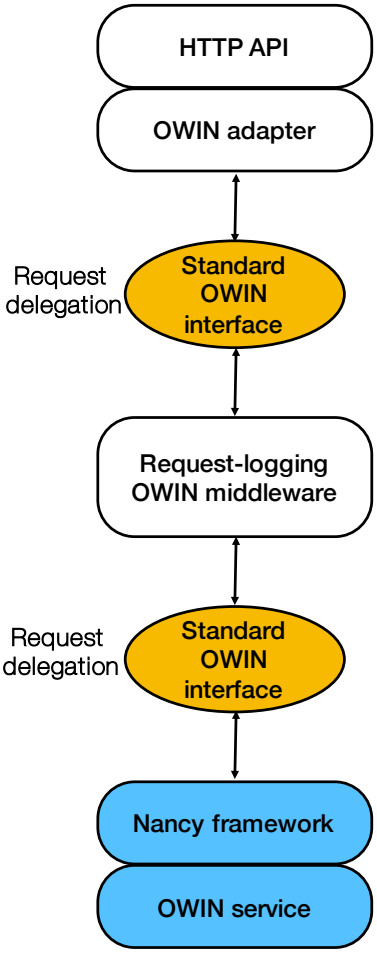


Figure 6.3: The standard OWIN interface acts as an adapter for HTTP requests between HTTP API and OWIN component, which leads to decoupling of OWIN component and HTTP API [EP17].

The RomanticWeb library has been chosen for the creation of JSON-LD representations because it provided the best solution for the creation of custom JSON-LDs.

DotNetRdf has been selected as the library of choice for interacting with RDF triple stores because it has the highest overall compatibilities with different RDF triples stores. For most of the popular RDF triple stores, there exists a client in DotNetRdf. Furthermore, it provides the largest range of functionalities, when for the interaction with RDF triple stores. DotNetRdf will be used in the technology stack of the .NET based HTTP service of the storage app.

Sensor API Service and Semantic Service

The selection decisions of the frameworks result in the following technology stacks displayed in figure 6.4.

6.3 Storage App

For the storage app two services have to be implemented, the SPARQL service and the HTTP service. In addition to the implementation of these two services, an RDF triple store has to be chosen, which will be used in combination with both services.

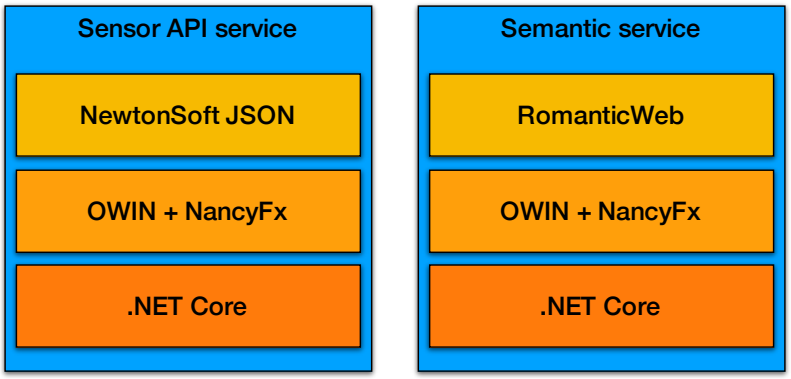


Figure 6.4: Semantic app technology stacks

Considered RDF Triple Stores

From the considered RDF triple store solutions, the first and second item is open source and the items three to five are commercial. For every commercial version, there exists a free or open source edition with reduced feature sets.

When choosing an RDF triple store, the most important features are an easy setup, a user-friendly administration interface, compatibility with the JSON-LD format and possibility of native access via DotNetRdf by using the graph store HTTP protocol.

Sesame. The open source Java framework Sesame offers a customizable API connectable to most RDF triple store solutions. It also includes functionalities for parsing, storing, inferencing and querying of stored data. Sesame provides query and update support and can handle the RDF file formats RDF-XML, Turtle, N-Triples, N-Quads and JSON-LD.

Jena TDB and Fuseki. Jena TDB is an RDF triple store solution, designed for small and static storage systems, with little frequent updates. TDB supports inferencing, but inferences have to be pre-computed before they can be used or stored for future quick access.

Apache Jena Fuseki is a SPARQL server, built on top of Jena TDB. It can both run as a web application as well as a standalone server. Fuseki is deeply integrated with TDB to create a persistent storage layer. This is achieved by using Jena text queries and Jena spatial queries. Fuseki can be also used to provide the protocol engine for other RDF query engines and storage systems. A user-friendly web-interface allows performing monitoring and administration tasks [JF14].

AllegroGraph. AllegroGraph is RDF storage system designed for large-scale, high-performance use cases. It offers advanced features like geospatial indexing and text indexing per predicate. AllegroGraph is also used as a storage component for many open source projects like TwitLogic, which aims to integrate Twitter data to the Semantic Web [Shi10].

Comparison. Figure 6.5 shows a comparison table of the presented RDF triple store solutions created by [RDE+07]. In the study, [RDE+07] et al. compared editions of the Sesame, Jena TDB and AllegroGraph triple store system against each other. For Sesame, a MySQL based, a DARPA Agent Markup Language database(DAML DB) [HM00] and a BigOWLIM [KOM05] based database edition has been used. These three triple store solutions have been compared with the version 1 and 2.0.1 of AllegroGraph and MySQL and DAML DB based editions of Jena TDB.

Choosing an RDF Triple Store Solution

	Sesame + MySQL	Sesame + DAML DB	Sesame + SwiftOWLIM	Jena + MySQL	Jena + DAML DB	AllegroGraph 1	Sesame + BigOWLIM	AllegroGraph 2.0.1
Query language	SeRQL	SeRQL	SeRQL	SPARQL	SPARQL	SPARQL	SeRQL	SPARQL
Rule support	RDFS	Limited RDFS	RDFS + most of OWL Lite	RDFS + most of OWL Lite	Limited RDFS	RDFS + a bit of OWL Lite	RDFS + most of OWL Lite	RDFS + a bit of OWL Lite
owl:sameAs	No	Using Layered SAIL	Yes	Yes	No	Yes	Yes	Yes
owl:Inverse Functional Property	No	Yes	Yes	Yes	Yes	No	Yes	No
API Extensibility	Yes, SAIL	Yes, SAIL	Yes, SAIL	Yes, Jena	Yes, Jena	No	Yes, SAIL	No
Maturity	Mature	Mature	Mature	Mature	Mature	Mature	Beta	Beta
Price	Free	Free	Free	Free	Free	Not free	Unknown	Not free

Figure 6.5: Comparison of RDF triple store solutions and their different versions [RDE+07].

The findings of [RDE+07] et al. state Jena TDB is a good candidate for this scenario. It provides SPARQL functionality and covers most of OWL Lite. The MySQL based edition has been chosen to be used in combination with Fuseki. The main advantages are no setup is required and it is easy to use, because its simple user interface, all of which makes it great for prototyping.

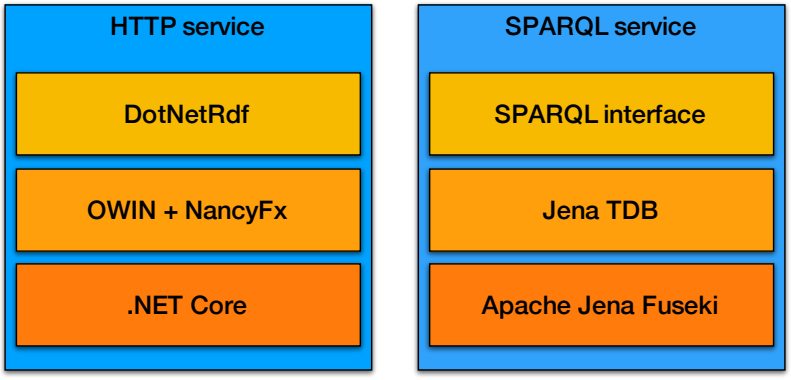


Figure 6.6: Storage app technology stacks

SPARQL Client

The SPARQL interface included in Fuseki. This allows merging of the SPARQL service and the database in one container solution to save development time. Fuseki now overtakes the functionalities of the SPARQL service, by providing its own SPARQL interface.

The Fuseki SPARQL HTTP endpoints published by the SPARQL client are the following.

- File upload - http://scale-it.org:3030/shopfloor/upload
- SPARQL 1.1 Query - http://scale-it.org:3030/shopfloor/sparql
- SPARQL 1.1 Update - http://scale-it.org:3030/shopfloor/update
- SPARQL 1.1 Graph Store HTTP Protocol - http://scale-it.org:3030/shopfloor/data
- HTTP Quads - http://scale-it.org:3030/shopfloor/

HTTP Client - Using .NET to access Triple Stores

DotNetRdf is used to access RDF triple stores within.NET. The purpose of creating an extra service in addition to the SPARQL client is the creation of a unified interface that can adapt on different triple stores and graph databases.

6.4 Command Center App

For the command center app three services have to be implemented.

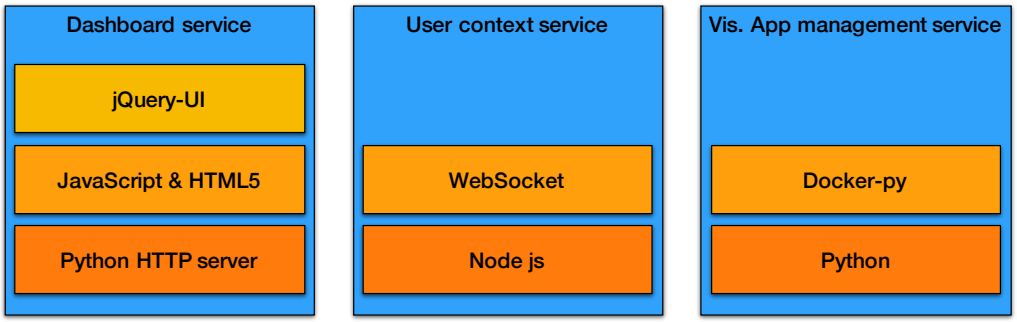


Figure 6.7: Command center technology stacks

User Context Service

The user context uses the Node javascript(Node js) framework to publish a web-socket event server. In the future, this server will be replaced by an HTTP solution to conform to the ScaleIT architecture style. Web-sockets is chosen because a similar solution has already been implemented with it. This solution fulfils the basic requirements of event-driven messaging. The implementation of the user contexts is done by using the publish-subscribe design patter. Every time a new visualization app client starts it registers and subscribes itself to the user context server by sending a request to its web-socket API. The user context server holds a list of all active visualization app clients, to which the new client is added. Every message published by a client to the user context is immediately sent to the entire list of active clients. If a client stops to respond for more then ten seconds it is removed from the client list and has to register again.

User context messages are serialized in JSON. Listing 6.3 shows an example of such a message.

For the purpose of the evaluation, the user context service has been extended by an event recording functionality. For every received message, a log-entry is created in a log-file later used for the analysis of the evaluation.

Serving the Web-Interface

For the serving of the web-interface of the dashboard service and the web-interfaces of the visualization apps, the SimpleHTTPServer module of python 3 is used. It is preferred over

```
{
  type:'utf8',
  utf8Data: {
   id: 'http://scale-it.org/shopfloor#sensordata10102',
   author:'listView1',
   timeStamp:'12:54:12 12.12.2017',
   eventType:'selection'
  }
}
```

Listing 6.3: User context message created by a list view app on the selection of a sensor data entity.

an APACHE HTTP server, because of its lower resource consumption.

Dashboard Service

The dashboard service user interface is implemented by using the javascript libraries jQuery and jQuery-UI. This provides users with an easily customizable interface.

To start and shut down individual visualization apps, requests are sent to the visualization app manager service.

The final UI design of the dashboard is shown in figure 6.8. The menu elements 'Note' and 'Studie' are only used for the evaluation process. Because users mostly work best, when they can customize their application after their own preferences, the goal of the UI design is to create a customizable dashboard. To maximize the freedom of the user in arranging his app windows a custom window manager has been created using the jQuery-UI library. Through this option, users can customize their window arrangement on the fly.

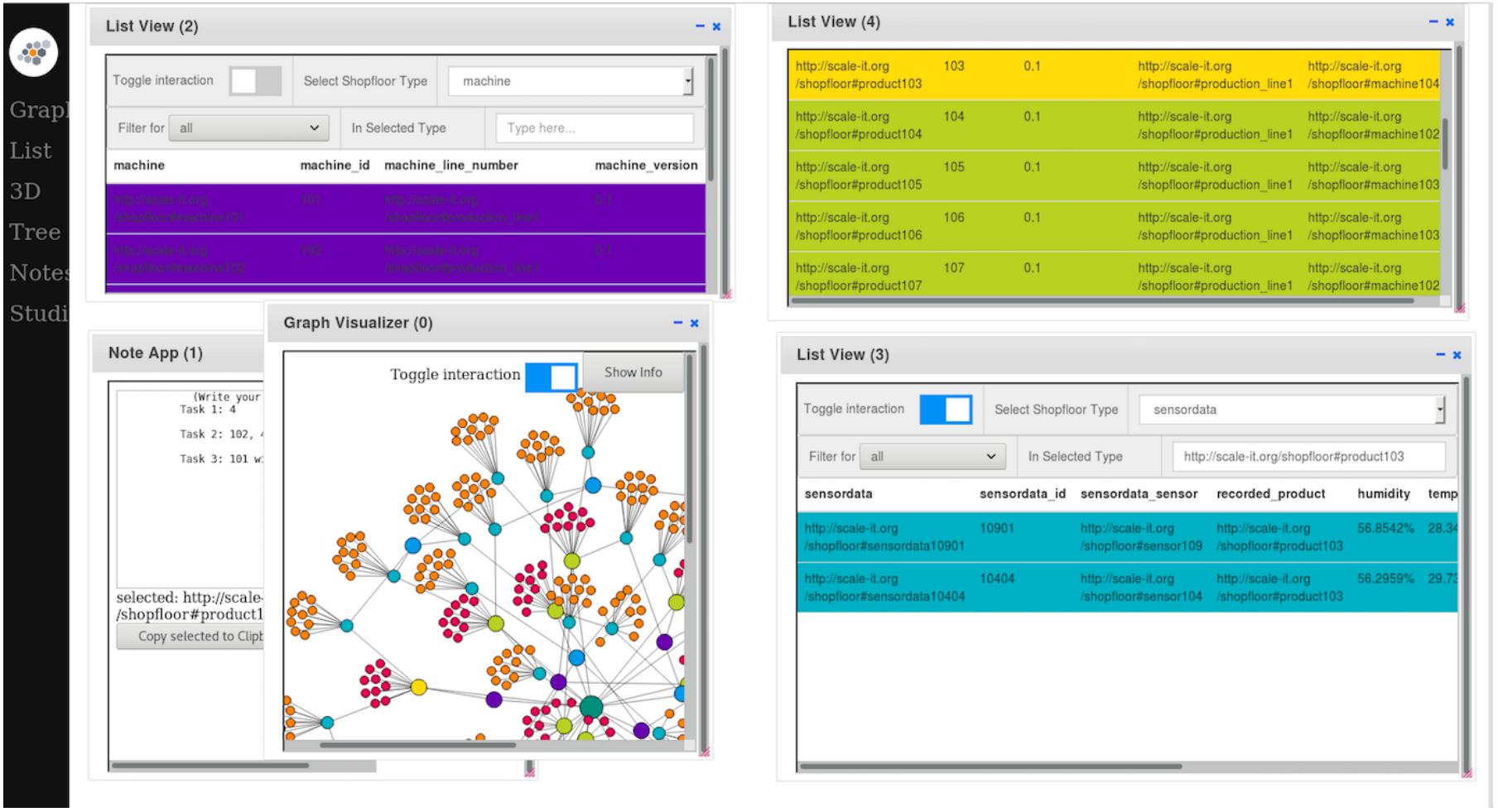


Figure 6.8: Visualization apps can be arranged inside the dashboard UI.

Visualization App Manager Service

The visualization app manager is connected to the host docker daemon. If one of its published commands is requested, it uses the host docker daemon to carry out the command. In the case of starting a new visualization app, it calls a new docker-compose command with the given parameters.

For interaction with docker, the python module docker-py has been selected.

6.5 Visualization Apps

Each visualization app is made up of three business function services.

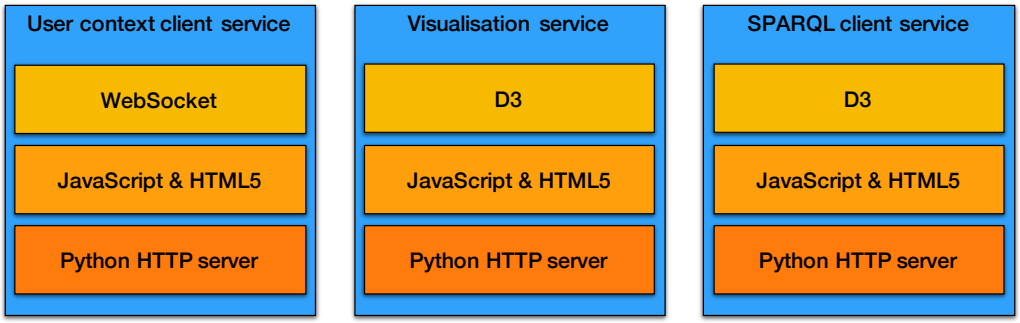


Figure 6.9: Technology stacks of visualization app

Visualization App Service

D3 is a javascript library for creating dynamic and interactive data visualizations for the web, by using the widely applied standards of CSS, HTML5 and SVG. D3 has been chosen because it allows greater control over the final visual representation and the wide range of existing examples of Linked Data visualizations.

User Context Client Service

Every type of visualization app uses a different version of the user context client extending the methods 'SendMessage' and 'ReceiveMessage' method from the SPARQL client. The implementation of this methods determines how a visualization app reacts to events inside the app, meaning possible sending of event messages. And how it reacts to events published in the same user context, meaning the receiving of event messages.

SPARQL Client Service

Uses the javascript library D3 to perform SPARQL on the SPARQL client of the storage component. Every type of visualization app uses a different kind of SPARQL client extending the 'QueryGraphData' method from the SPARQL client. The address of the SPARQL server is injected by the visualization app manager through the docker-compose command.

Visualization Service

The visualization service makes uses the extended methods in the SPARQL client service and the user context client service to offer the user a specialized visualization of the shop floor. The type of visualization differs in every visualization app.

List view app. The purpose of the list view app is to provide user-friendly querying of different types of shop floor entities. Each row in the view represents an entity instance, from which each column represents a particular attribute of the instance.

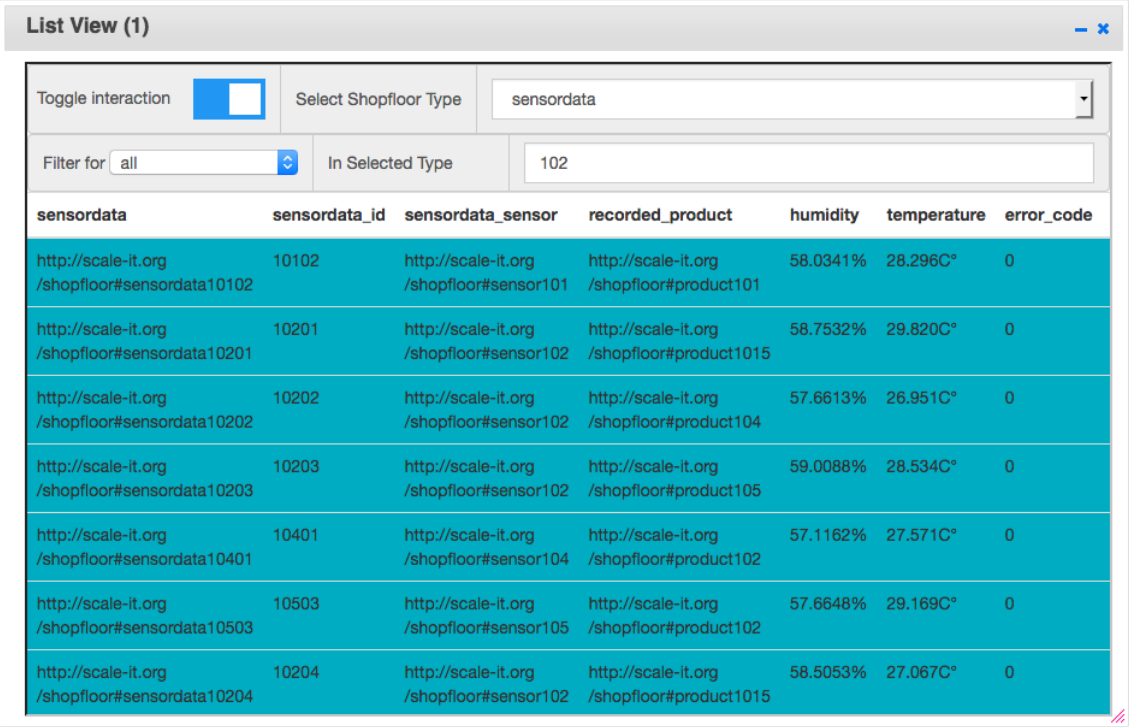


Figure 6.10: The list view app is used to view and filter different shop floor entity types.

Graph view app. The purpose of the graph view app is to give users an overview of relational structures on the shop floor. Node entities in the graph view are arranged around production lines. And sub-entities like product components or sensor data values are aligned with their 'parent'-entities.

Tree view app. The tree view app only can receive messages from the user context, but cannot send them by itself. This functionality may be added in future versions.

3D view app. Using Unity3D to for creating the 3D interface. For the 3D models used in the 3D view, open source models have been adopted and combined to create a realistic 3D shop floor visualization. To first optimize the basic functionality and usability, basic shapes have been used as placeholders for shop floor entities as displayed in figure 6.13. In the second version, these shapes have been replaced by realistic 3D models shown in figure 6.14.

To make the 3D view app more adaptable and to keep the communication with user contexts standardized in all apps a delegation component has been developed.

The delegation component delegates communication between web-browser and Unity3D. This allows using the same implementation of user context client and SPARQL client.

Note view app. The note view app provides participants with a digital notepad on which they can keep track of their task results. When they subscribe the app to the user context, they can automatically insert the name of the most recently selected entity.

Additional Tools supporting the Evaluation

To automate the generation of different shop floor entity data set a shop floor generator tool has been developed. A second tool has been developed to automatically record heat maps of the mouse hovering and mouse clicking actions from users during evaluation sessions.

Shop floor data generator. The shop floor data generator is developed as a tool to increase the process of testing of data sets used for the evaluation process. It takes

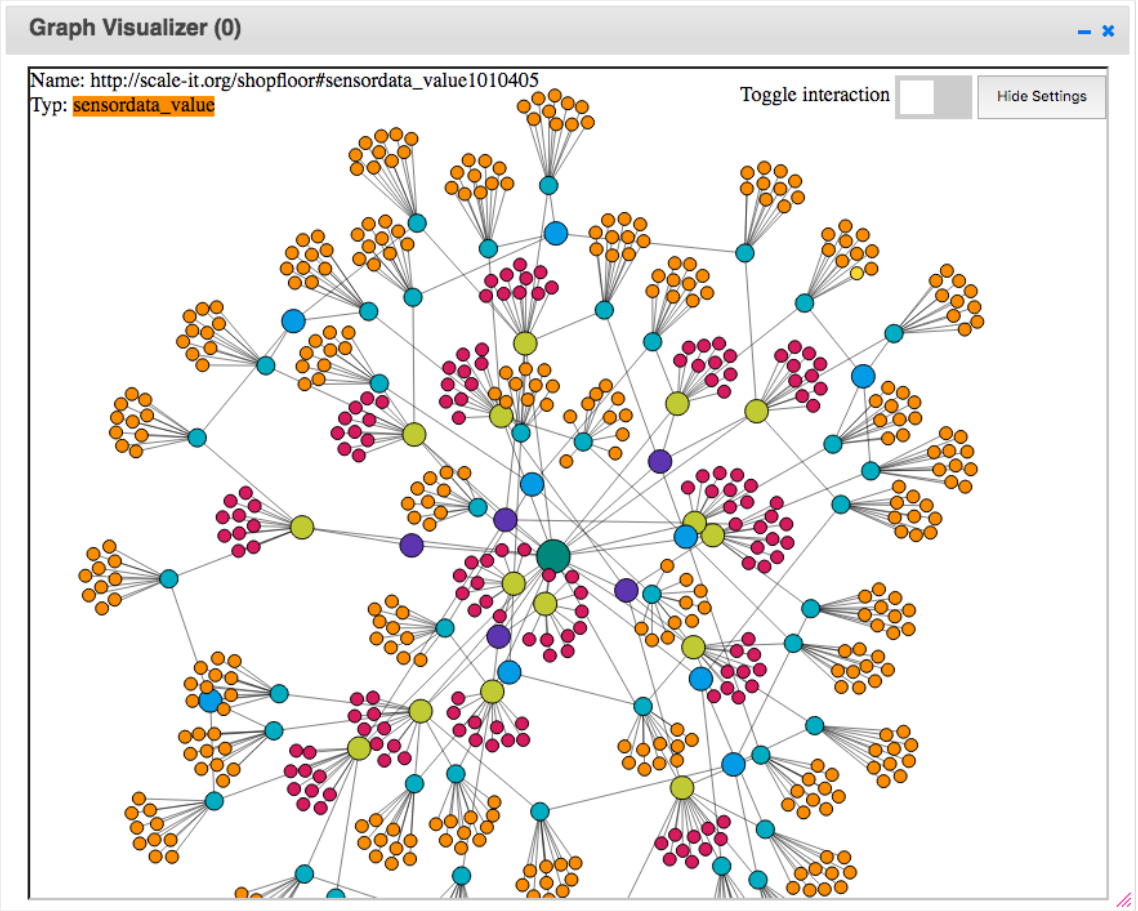


Figure 6.11: The graph view app can be used to better understand relationships on the shop floor.

the predefined entity amounts from the user through a web-interface and calculates the respective shop floor entities by generating random entries from a predefined range for entity attributes.

Heat map recorder service. The heat map recorder service, each visualization app uses during the evaluation is implemented using the javascript library Heatmap.js. During a session, all mouse move and mouse click events are recorded. After a session, they can be used to generate the heat maps. This procedure is chosen to maximize the responsiveness of the applications during usage.

Recording of user context-activity. The user context events occurring during an evaluation session are recorded to be later used for evaluation analysis.

6.6 Conclusion

The implemented architecture consists of four apps, the semantic app, the storage app, the dashboard app and the visualization app.

Every app is made up of business function services and technical function services. The business functions where defined, based on the requirements set in the system design. Technical function services support the business function services by outsourcing functionalities from a business function service.

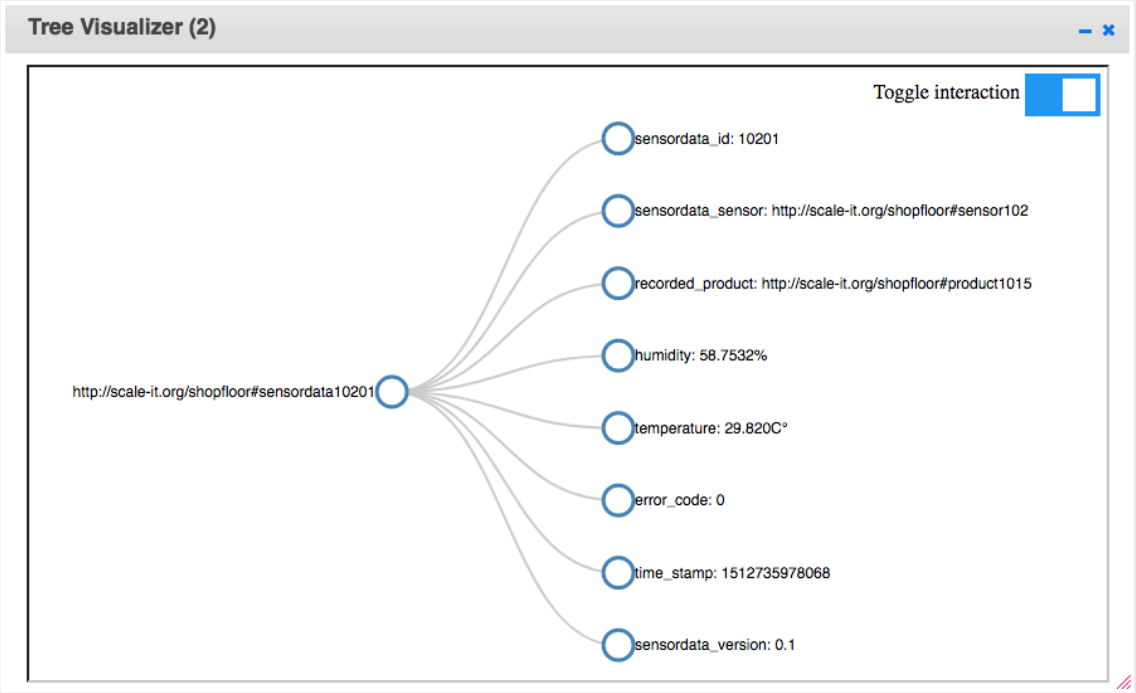


Figure 6.12: The tree view app allows for tree like visualization of individual shop floor entities.

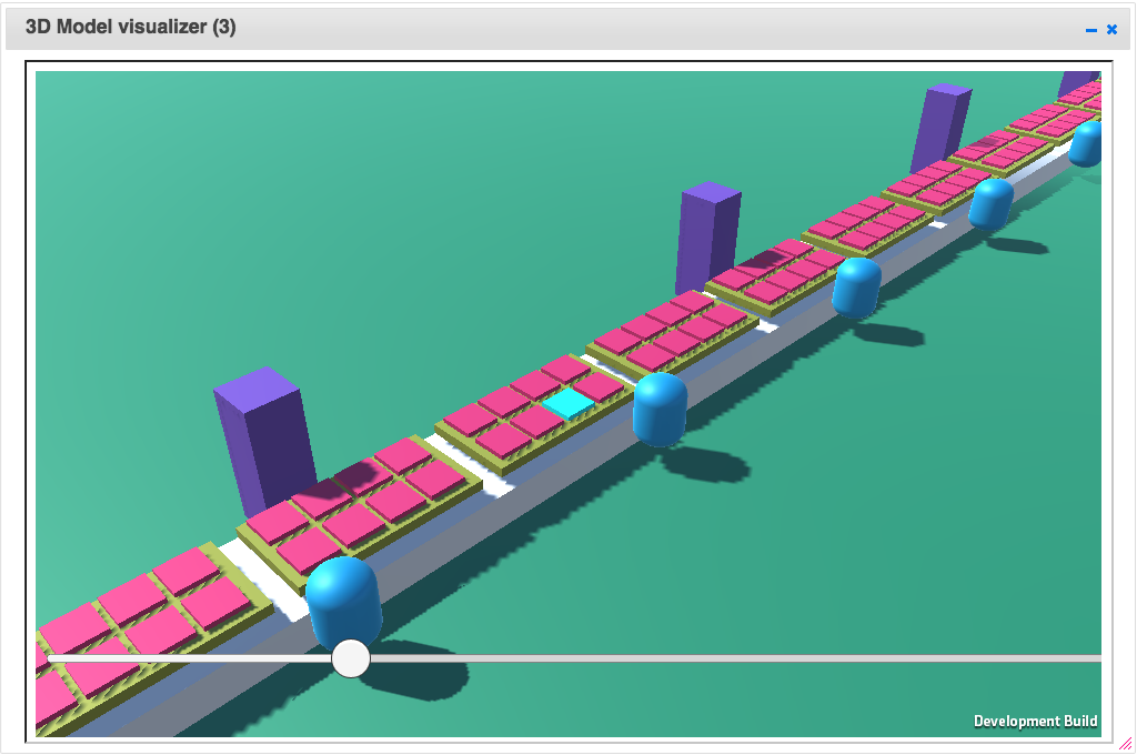


Figure 6.13: The 3D view app offers a three dimensional representation of the shop floor.

Most parts of the system could be implemented as planned in the design. A few parts of the design have been adopted during the implementation process because of the following reasons:

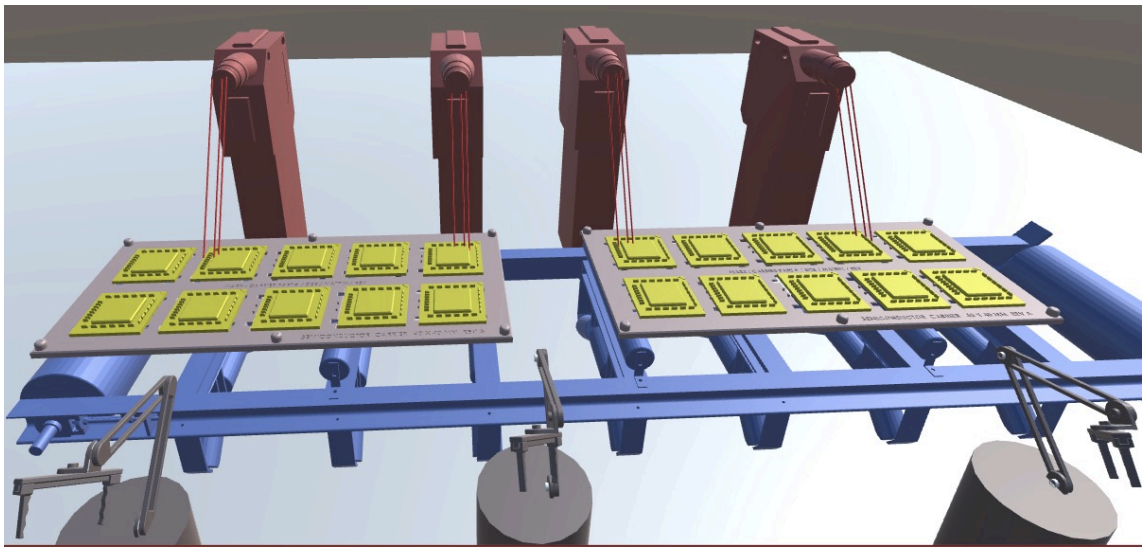


Figure 6.14: The basic shapes used in the first version of the 3D view app have been replaced by realistic 3D models.

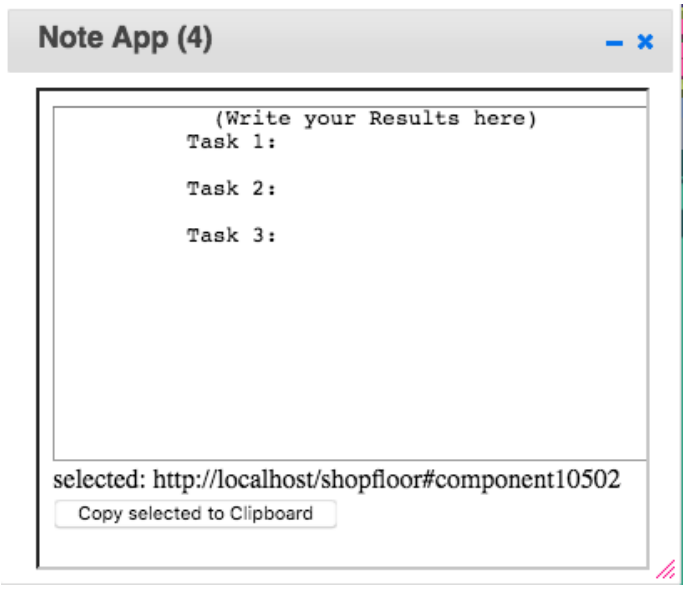


Figure 6.15: The note view app provides participants with a digital notepad on which they can keep track of their task results.

1. **Event feed only implemented for .NET based services.** The planned event feed has only implemented for .NET based services but not for other services.
2. **Strict use of HTTP and REST APIs.** HTTP and REST APIs are not strictly implemented for all services. Because of the reasons presented in the command centre app section 6.4, the web-socket has been used instead of HTTP.
3. **Replacement of the SPARQL storage client.** To reduce development time of the storage system and the SPARQL service, the SPARQL service is omitted. This is possible because the Fuseki storage solution offers enough SPARQL functionality for this scenario.

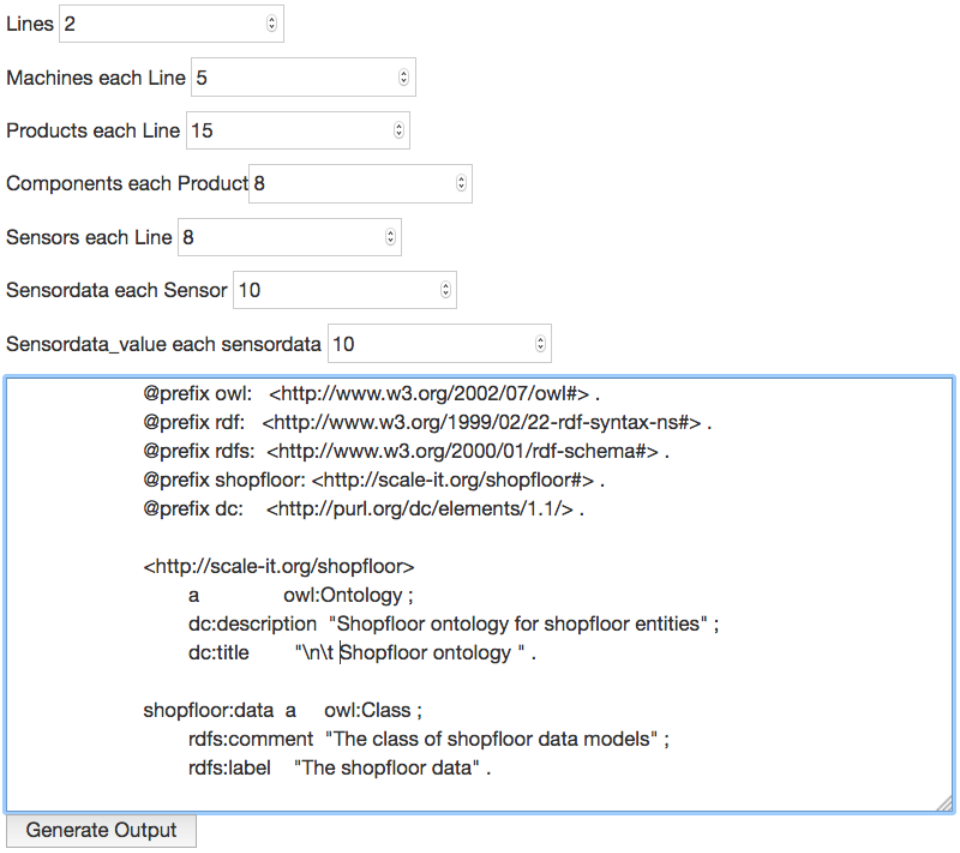


Figure 6.16: The shop floor generator allows users to generate shop floor data sets with predefined amounts for each entity type.

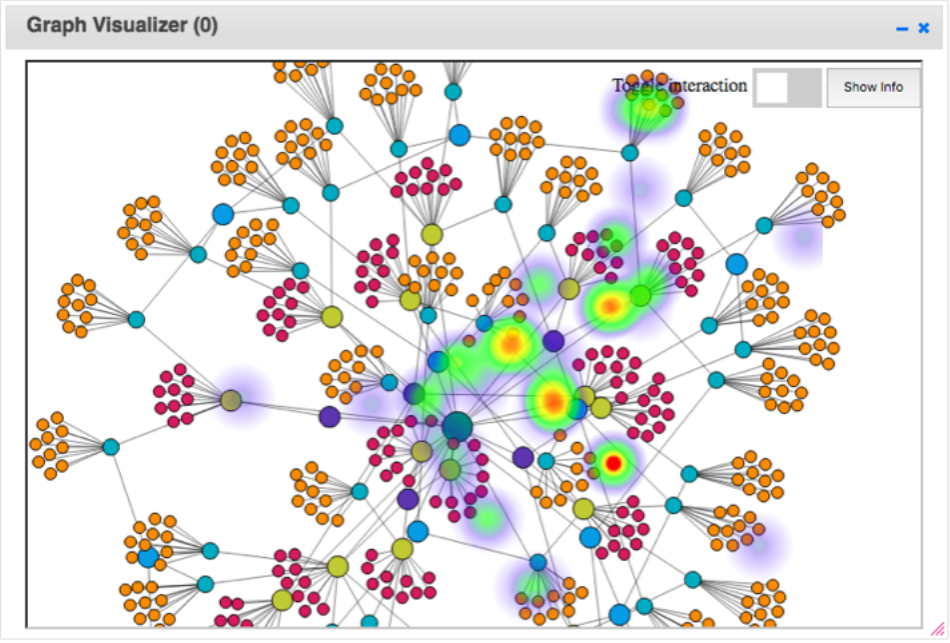


Figure 6.17: Graph view with overlayed heat map of mouse clicks.

7. Evaluation

To test the visualization part of the developed system, including the visualization apps, a user study has been conducted. The study has three goals. First to evaluate usability and user experience of dashboard and visualization apps. Second to test, how the system supports users in carrying out typical shop floor maintenance tasks. And third to observe different ways, how visualization apps are being used in shop floor maintenance tasks during the evaluation.

7.1 Measures

To capture the information needed to evaluate these three goals, a set of various measures has been selected.

SUS

To evaluate the usability of the designed system the system usability scale (SUS) is used [Bro96]. SUS is a reliable approach for evaluating the usability of a system. In detail, it measures effectiveness, efficiency and user-satisfaction of a system.

SUS uses a Likert-Scale which results in a numeric result in the range of 0 to 100. The SUS questionnaire consists of ten questions. Each answer ranges from 'Strongly disagree' to 'Strongly agree', with three options in between as displayed in figure 7.1.

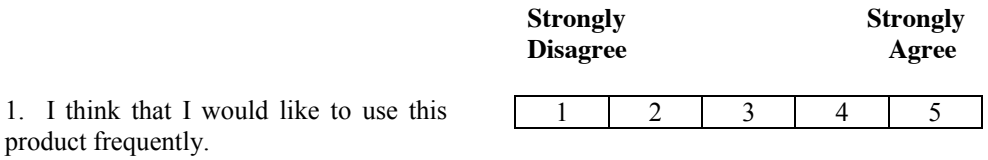


Figure 7.1: Each answer in the SUS questionnaire ranges from 'Strongly disagree' to 'Strongly agree'.

UEQ

In addition to SUS, the User Experience Questionnaire(UEQ) is used. UEQ is a simple questionnaire for measuring the user experience of a system. It consists of twenty-six items

capturing the six factors Attractiveness, Perspicuity, Efficiency, Dependability, Stimulation, and Novelty of user experience as described by Laugwitz et al. [LHS08].

Each of the six factors measures a different area of users experience.

1. Attractiveness: Measures the overall impression of the system, meaning how does a user like or dislike the system?
2. Perspicuity: Measures how easy it is to get familiar with the system.
3. Efficiency: Can users solve their tasks without unnecessary effort?
4. Dependability: Measures the degree, to which a user feels in control.
5. Stimulation: Is it exciting and motivating to use the system?
6. Novelty: Does the system catch the interest of users by being innovative and creative?

Figure 7.2 shows exemplary items for the scale of perspicuity. For each question item in the UEQ questionnaire, the answer is ranging between an adjective and its opposite with five options in between.

not understandable	o o o o o x o	understandable
easy to learn	o o o o o o x	difficult to learn
complicated	o o o o x o o	easy
clear	o o o o o x o	confusing

Figure 7.2: Items used for the scale of perspicuity in UEQ questionnaire.

Custom Questions

To complement SUS and UEQ a handful of custom questions have been added. Their purpose is to identify current pitfalls and problems of the system. But also to identify superfluous features, which can be omitted, and essential features, which can be highlighted for users. These questions don't focus on particular tasks but on the general impression of the user. Insights of answers from these questions will be used, when considering improvements for future system versions. The questions are divided in general, which address the whole system performance and specific questions, which address how the system performs under different task types.

General questions:

Q1 - What did you most like about the system?

Q2 - What did you most dislike about the system? If so, how would you improve/change it?

Q3 - Which feature did you miss when using the system?

Q4 - Which feature did you find unnecessary or superfluous?

Q5 - Did you miss to fulfil any tasks? If so, why?

To supplement these questions, another handful of questions have been created, which focus on tasks individually. Their purpose is to collect feedback used to improve the task portfolio and the quality of individual tasks for future evaluations. The specific questions are:

Q6 - Did a particular piece of information in the instructions made it easier to accomplish this task?

Q7 - Did a particular piece of information in the instructions made it harder or more confusing to accomplish this task?

Q8 - Did you miss a particular instruction, when performing this task? And how would this piece of information help you to perform the task?

7.2 Tasks

When performing real-world tasks on the shop floor, tasks sometimes include many different aspects of shop floor maintenance and may cover a large set of data, from different shop floor entities. This has to be taken in consideration. For the process of creating the tasks for this evaluation, the goal is to reduce the scope and to shorten the length of each task. This allows reducing the factors that must be considered for the analysis of system user performance. This is achieved by limiting task execution paths and increasing the number of tasks to be performed during one session.

This is achieved through the process of breaking down of real-world tasks to their core problems. As an example, when looking for error-producing members in a production chain. The core problem is to identify the event triggering the error and the chains of all possible problem sources involved in earlier interactions with the erroneous entity. Often many different tasks are based on the same core problem. This tasks can then be merged into one or two small tasks addressing the core problems of these tasks.

The tasks created for this evaluation are divided into two groups:

The first group includes the tasks only requiring an understanding of first-order relationships on the shop floor.

1. Which types of sensor data have the highest fluctuations?
2. Which sensor produces the highest fluctuations in sensor data?
3. How many sensor data recording exist for the product 'product104'?
4. Examine the product processing order on production line 1.
5. Look up the ids of the product components of the product 'product1012'.
    a) Which machines processed the product with id '10101'?
    b) In which order did they do the processing?
6. Which sensors recorded the most amount of failures in single product components?

The second group includes tasks requiring an understanding of order two or higher relationships on the shop floor.

1. How many measurements exist from products processed by the machine with id '101' that have a failure rate of two or higher?
2. Examine the relationships between sensor data and the machine with id '102', what type of relationships do exist?
3. Find a machine that processed the most products, for which one or more errors have been recorded.
    a) How many sensors record data from products processed by the machine with id '101'?
    b) How many data sets did they produce in total?

4. How often did the machine with id '104' fail, meaning how many erroneous products were recorded, that were processed by this machine in the last three days?
5. Find a machine that processed the most products, for which no errors have been recorded.
6. What machine processed products before they were measured by the sensor with the id '107'?
7. Which the machine that has to be repaired next?
8. Find all the sensors that measured the product with id '10101' before it was processed by the machine with id '104'.

The first group is used to introduce the user to the interface and to get her to know the shop floor environment. If the user got accustomed to the application and the structure of the shop floor, she will be confronted with tasks from the second group.

For the evaluation, two tasks from group one are chosen. One task as an example illustration and one task as a warm-up task. After this, the participant is confronted with two tasks from the second task group.

Figure 7.3 displays the chosen tasks for the evaluation scenario.

#	**Task description**
T1	Look up all product component ids of product 'product1012'.
T2	How many sensor data recordings exist for product 'product104'?
T3	Find a machine that processed the most products, for which no errors have been recorded.
T4	Find a machine that processed the most products, for which one or more errors have been recorded.

Figure 7.3: Chosen tasks for the evaluation scenario.

The first two tasks are chosen because they are easy to understand, but at the same time already require the use of user contexts and multiple visualization applications. The second two tasks are chosen because they presuppose a certain level of understanding of shop floor relationships and entities. And an existing understanding of how to make use of user contexts. Without the understanding of these things, it is very hard for participants to successfully complete the last two tasks in time.

7.3 Overview

The following section gives an overview of the study and step by step process of each evaluation session as designed in chapter 5.7.

Participants

The study group consisted of five participants. The number five is chosen, because "observing four or five participants allows practitioners to discover 80% of a product's usability problems, observing additional participants reveals fewer and fewer new usability problems" [TLN06]. Form these five participants, two are already familiar with other shop floor structures, while three have no previous knowledge about the entities on the shop floor. All of the participants either have computer science as a major or as a minor subject in their course of studies. Figure 7.4 shows a table with basic information of all participants.

#	Age	Gender	Education	Previous knowledge
P1	23	Male	Bachelor	Yes
P2	27	Male	Master	No
P3	26	Male	Bachelor	No
P4	21	Female	Bachelor	No
P5	29	Male	Master	Yes

Figure 7.4: From 5 particpants, 2 have previous knowledge of the shop floor structure.

Data Set

The dataset used for the study included 621 entities in total and 665 relationships between individual entities. It is based on an ontology made up of 3971 triples and uses 20 different entity attributes. This dataset is generated using the shop floor generator tool described in chapter 6.6 5.7.

Available Tools

Figure 7.3 displays the working environment provided for each participant. It consists of a desktop machine with two 20”, full HD monitors with a keyboard and a mouse. The desktop machine has a second Generation i7 processor and 16 GBs of RAM. In the study, the user dashboard, which ran in the Firefox browser, is laid over both screens.

Before the evaluation starts a web browser is opened, displaying an empty dashboard view.

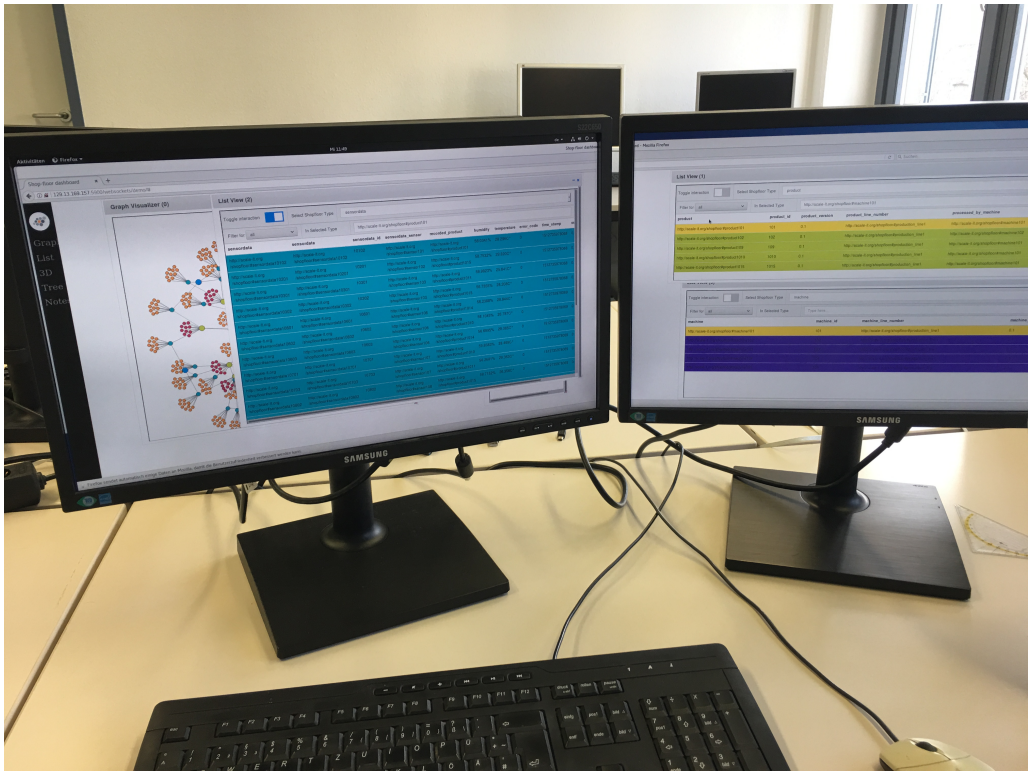


Figure 7.5: The user environment setup used by each participant consists of a desktop machine with two 20”, full HD monitors.

To solve three task previously described, the participants have to use three visualization apps, the list view app, the tree view and the graph view app. The 3D model view app

is not included in the study to limit the available options of the user and to encourage new ways of combining the given visualization apps by using their user context. It may be included in future evaluations.

For the tasks, participants could use single apps or they could combine the functionalities of multiple apps within the user context.

For each task, the user has to decide which combination of apps to use. Different tasks favour different combinations of apps, depending on the problem or on personal preferences and experience of the participant. Before the participant starts her task first task, possible approaches and helpful app-interactions are presented to him by using the example task. The following text explains a sample workflow including usage of multiple visualization apps.

Sample Workflow

Every app is launched in the dashboard, by clicking the name of the app in the left navigation bar. To enable apps to react to the user context the blue slider button has to be enabled. An interaction between apps can look like the following.

The list view app and the tree view app are started by clicking their names in the navigation bar. Then the user context slider is activated in the tree app. In the next step, a specific sensor data observation is selected in the list app by clicking on it. Now every app, which has reactions enabled and also contains a possible representation of the clicked object will show this representation and execute a predefined custom action. In this case, a representation of the observation will be displayed in the tree view app.

Before a participant gets confronted with the three tasks, she will first take a look at the documentation, which will guide her through a basic example workflow. The documentation also contains all features of each app listed with the associated user actions and a short introduction text to explain the type of tasks in which the app can be used. In the list view app, for example, a user can type a search term in the search bar and select a shop floor entity type in the shop floor type selector to list all entities of this type containing the typed searched term. After a user read the documentation she will go through the basic example workflow involving every app. The task in this example is to look up all ids of product components from product '10101'.

1. Add a list view app instance and a graph view app instance from the dashboard using by clicking the Graph button and the list button on the side navigation bar.
2. Enable the user context in the graph view app by clicking the blue slider button.
3. Select the product type category and search for the product with id '101' in the list view app and select it by clicking on it.
4. Now the product should be highlighted in the graph view app.
5. Now by clicking the neighbouring product components. You can read their ids if you click the info button in the graph view app.
6. Now open the tree view app. Enable the user context inside the tree view app.
7. In the graph view app, start exploring other nodes and spend some time by reading their tree representations.

Now the user starts the three tasks with the warm-up task as described in chapter 5.7.

7.4 Results

This section presents the results of the evaluation measures, without discussing them, starting with the SUS questionnaire.

Results of SUS

Kortum discovered in an empirical study, that the average SUS score is 69.7 per study [BKM08]. The result of the study is a scale categorizing SUS scores into actual usability. This scale is used in figure 7.6 to frame the SUS result of this evaluation.

The result of the SUS questionnaire is 72, which is between an ok score and a good score. From all scores, the lowest is 50 and the highest 85. A detailed composition of the SUS result is display in figure .

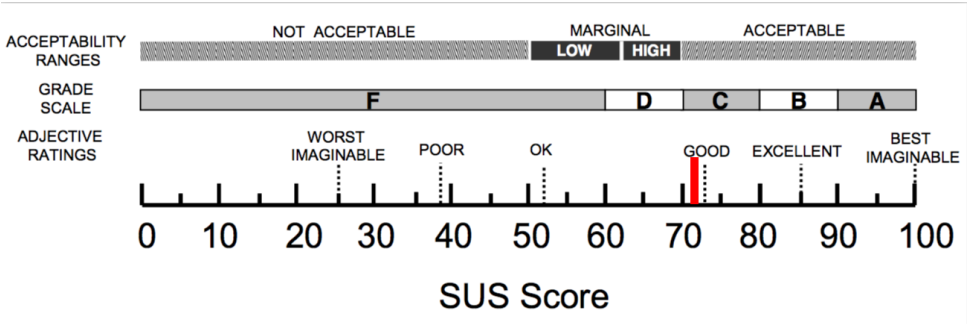


Figure 7.6: SUS results presented in the scale created by Kortum et al. [BKM08]. The result of the study is represented by the red bar at 72 points on the scale.

SUS criteria	1	2	3	4	5	Avg.
I think I would like to use this application frequently	0	1	0	2	2	4.0
I found this Application was unnecessarily complex	0	2	1	2	0	3.0
I thought the system is easy to use	0	0	0	4	1	4.2
I think that I would need assistance to be able to use this application	1	1	0	3	0	3.6
I would imagine that most people would learn to use this application very quickly	0	0	0	3	2	4.4
I found the various functions in this application well integrated	0	0	1	1	3	4.4
I thought there is to much inconsistency in the system	2	3	0	0	0	1.6
I found this application very cumbersome to use	1	2	2	0	0	2.2
I felt very confident using this application	0	1	2	1	1	3.4
I needed to learn a lot of things before I could get going with this application	1	4	0	0	0	1.8

Figure 7.7: SUS results - Table header: 1 for “strongly disagree” and 5 for “strongly agree”. Table body: Number of participants, which seletected this answer.

Results of UEQ

UEQ results, the six UEQ scales, attractiveness, perspicuity, efficiency, dependability, stimulation and novelty, range each between -3 and 3. Schrepp categorizes scale values over 0.8 as a positive user experience and below -0.8 as a negative user experience [LHS08].

The results of UEQ scales of this study are displayed in figure 7.8. Figure 7.9 compares the results with 246 other studies. As an important notice, because of technical reasons,

the twenty-first item clarity is not recorded and therefore has not been used to calculate the value of perspicuity. Figure 7.8 displays the values of UEQ scales, where the highest value is attractiveness with 1.667 and the lowest dependability with 0.8.

UEQ scales	
Attractiveness	1.667
Perspicuity	1.133
Efficiency	1.550
Dependability	0.800
Stimulation	1.400
Novelty	1.000

Figure 7.8: The six UEQ scales of the user evaluation, where attractiveness has the highest value with 1.667 and dependability the lowest with 0.8.

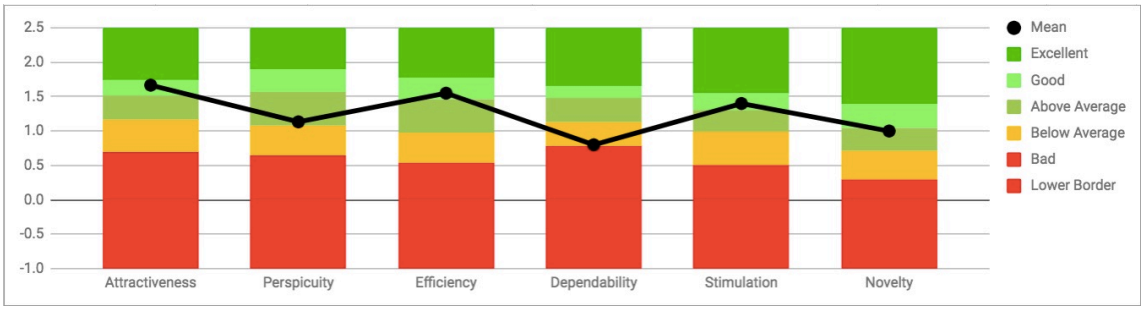


Figure 7.9: The UEQ scale values of the user evaluation compared against 246 other studies.

The value of the UEQ scale can be divided into three groups. The first contains the attractiveness scale, which is a pure valence dimension. The second group contains goal-directed(pragmatic quality) scales: Perspicuity, efficiency and dependability. The third group contains non-goal-directed (hedonic quality) aspects: Stimulation and novelty. The average value of each group is displayed in figure 7.10.

Pragmatic and Hedonic Quality	
Attractiveness	1.67
Pragmatic Quality	1.16
Hedonic Quality	1.20

Figure 7.10: Attractiveness,Pragmatic and Hedonic Quality values from the UEQ results.

Results of Custom Questions In the following, a selection of answers to the custom questions is presented, preceded by the outgoing question.

Q1 - What did you most like about the app?

- Column view, with multiple synchronized views very suitable for ontologies.
- Option to move and create windows. Interaction
- Graph increases overall clarity
- Speed and re-activeness of the selection

Q2 - What did you most dislike about the app? If so how would you improve/change it?

- I had to manually count the entries, a function to do that would be nice
- UI, not self-explaining yet.
- Not clear, which window to toggle for synch functionality

Q3 - Which feature did you miss when using the app?

- Possibility to sort columns in the list view app
- Possibility to directly re-use the filter result in a list view app.
- SQL-query generated from selection in GUI.
- Coloring on the graph when a node is selected, no user feedback when search yields empty results.
- Counting elements based on rules
- Dynamic display of tool-tips, when hovering over specific areas of an app

Q4 - Which feature did you find unnecessary or superfluous? - No answers

Q5 - Did you miss to fulfil any tasks? If so, why? - No answers, all tasks could be completed by all participants.

Answers from questions **Q6** to **Q7**, contained the same answers already mentioned in the questions **Q1** to **Q5**.

Recorded Heat Maps and Statistics

The table displayed in figure 7.11 shows the number of selection events recorded in the graph and list view app for every participant. The selections in the tree view app are not recorded because the tree view app only shows information about the currently selected element, which makes selections of other elements impossible.

The highest number of total selections is 310 and the lowest 79. Overall the list view app is the most used app by all participants.

#	**Total clicks**	**List view clicks**	**Graph view clicks**
P1	290	289	1
P2	310	184	126
P3	206	191	15
P4	120	118	2
P5	79	46	33
Mean	201	165,6	35,5

Figure 7.11: The distribution of the number of selections between the graph view and the list view differs between participants.

The heat map in figure .7 shows the regions, where a participant clicked most often in the graph view app during an evaluation session.

Figure 7.5 presents a heat map of the list view recorded during an evaluation session. The right areas on the heat map show high click rates. For example, the most clicked area of the app is located around the 'Toggle interaction' button.

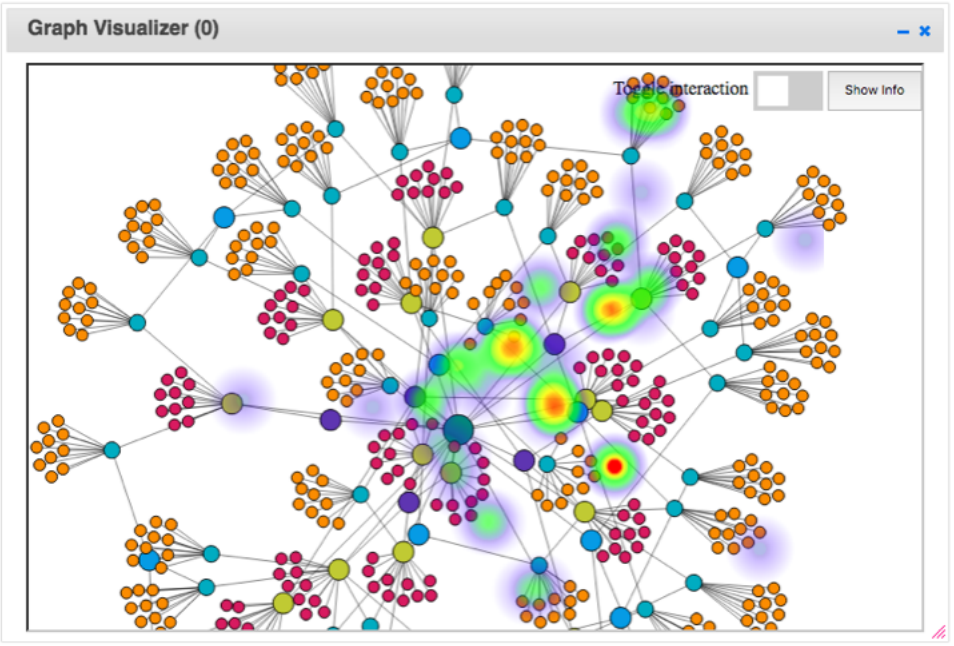


Figure 7.12: Graph view with overlaid heat map of mouse clicks.

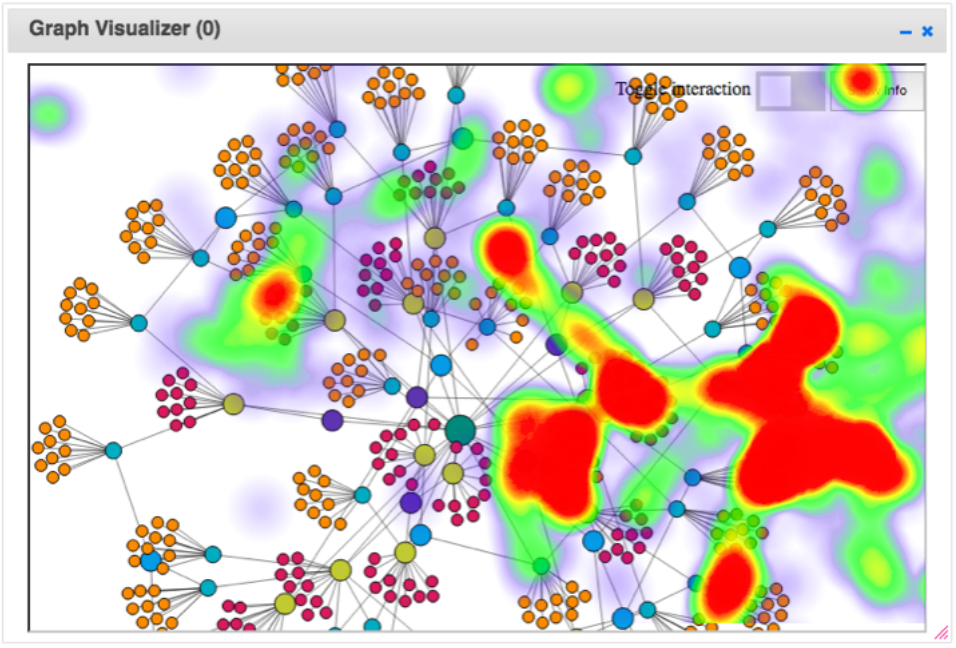


Figure 7.13: Graph view with overlaid heat map of mouse hover.

7.5 Discussion

In this section, the results presented in the previous section are discussed. The discussion includes possible interpretations of the results and suggestions of causes for this results.

SUS

The result of SUS with 72 points indicates the system is usable. Starting points for further improvement can be derived from the custom questionnaire answers. This is stated below

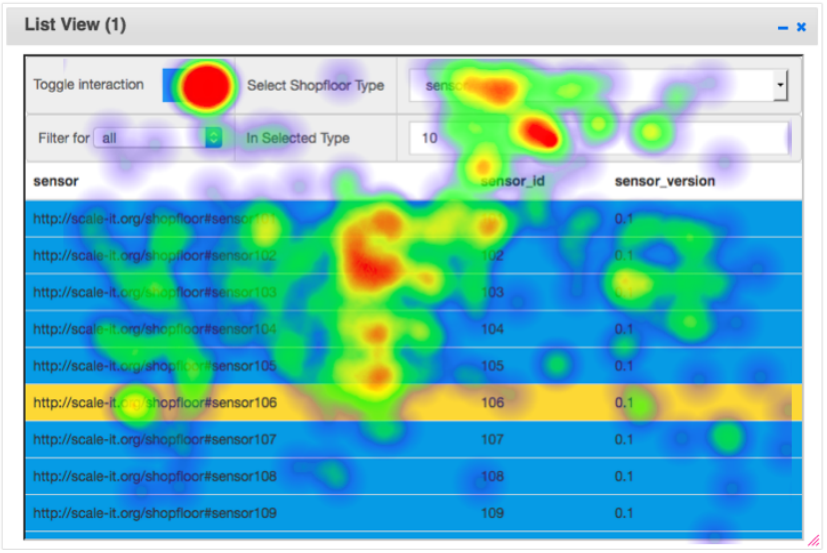


Figure 7.14: List view with overlaid heat map of mouse clicks.

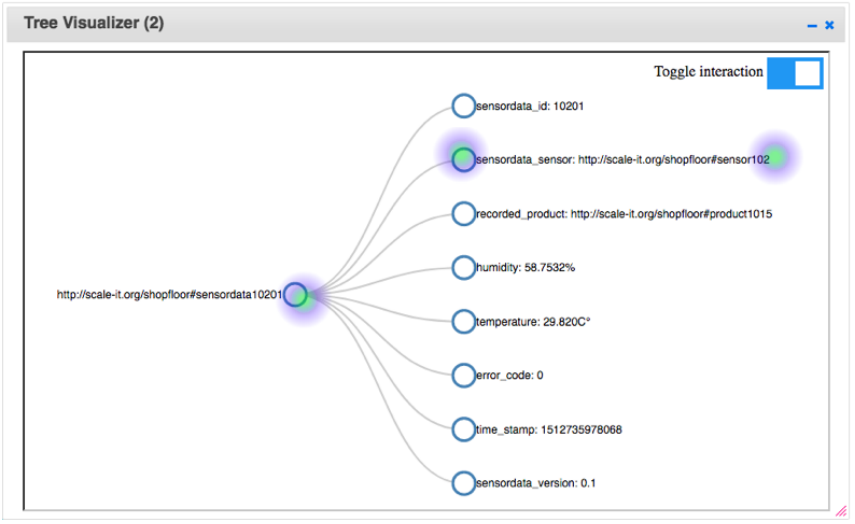


Figure 7.15: Tree view with overlaid heat map of mouse clicks.

in the section custom questions.

In comparison to the SUS scores of 82 and 77.5 of related work presented in chapter 4, the system scores lower.

UEQ The lowest UEQ scale is dependability with a value of 0.8, which is below a good result by 0.2 points. The dependability of the system can be improved by adding additional information text to different tools, when hovering over their access areas, but also by improving or replacing the app window management solution, which right now requires high precision when arranging app windows.

Custom Questions

Possible improvements derived from the custom question answers are:

1. Show information of important elements, when hovering them.
2. In list view app: Sort after selected column, combine filters to allow searching for more than one characteristic.
3. In graph view app: Keep the colour of the selected node and mark him by an outer circle.

Recorded Heat Maps and Statistics

The recorded heat maps recorded during participant sessions provide insight into how different applications have been used. For example, the heat map displayed in figure 7.5 shows the list view app interface with an overlaid heat map of mouse clicks. The warm regions indicate the most actively used areas of the app. They are around the context toggle button and the filter text input area. The fact that the context button toggle button is covered by the warmest heat map area implies a high rate of subscribing and un-subscribing commands to the user context of app issued by the user.

The reason for that could be the usage of several list view apps in a workflow, where the current selection in a few of these apps is intended to be preserved in certain steps of the workflow. A possible feature that could help here is the linking of apps, which restricts the set of apps that react to event-messages sent by an app.

In addition to heat maps displayed in figure and there are also cases where heat maps provide less meaningfulness. One example for this is the heat map displayed in figure . Additional heat maps can be found in the 8.3 of this document.

The statistics presented in figure 7.11 about the usage of different apps by different participants allow several conclusions. At first, the list view app is used by a factor of 4.7 more often at average than the graph view app. This implies that most of the work in the tasks are done by using the list view app. To further investigate these facts a study will be made using tasks favouring both apps equally. Supplementary to this study, participants will be an interview about their personal preferences, when selecting apps for particular shop floor task workflows.

7.6 Conclusion

To evaluate the system designed and implemented in previous chapters, a study has been conducted. For the study, three tasks are chosen. To perform the tasks, participants have to utilize the tools provided by the system to understand relationships on the shop floor.

To measure the evaluation process, multiple measures have been chosen. For user experience and usability, SUS and UEQ have been used. For individual feedback, custom questions have been created, to identify possible weaknesses of the system, upon which the system can be improved. And at last, statistical information has been collected about the user behaviour during the study, in form of the visualization app usage ratio and the mouse usage in terms of click and hover movements.

As an overall evaluation result, the system was tested to be usable. But for day-to-day use, some improvements must be made to guarantee a higher level of dependability. This can be achieved by realizing feedback of participants and adding of more visualization apps, complementing the visualization styles in the visualization apps already developed.

8. Conclusion and Future Work

This chapter looks back on the work described in this document to classify its relevance and impact. The architecture includes all parts of the data flow chain on the shop floor, from raw sensor data acquisition over Linked Data generation to a human-readable visualization for different shop floor parties.

8.1 Conclusion

The architecture design of the system addresses two major challenges on the shop floor. First, the acquisition and access of sensor data and machine meta-data on the shop floor and second, the availability and readability of information about relationships between IIoT devices.

To master these challenges a proof-of-concept architecture has been developed. The goal of the architecture is to acquire shop floor data, process it to generate Linked Data and expose it in a way that makes it accessible and readable for different parties on the shop floor. The design of the architecture is based on a distributed micro-services architecture combined with Linked Data and IoT principles like decentralization, digital twins and device inter-connectivity. The architecture is deployed with a containerized infrastructure.

The challenge of unequal creation and access of sensor data and machine meta-data is addressed by a system design that allows for automatic generation of Linked Data from acquired raw sensor data. In the generation process, the raw data is parsed and enriched with provenance information to create Linked Data, as described in chapter 5.5.

To provide a more generic solution that can be adapted to different scenarios with different data and data storage types the Linked Data adapter design pattern has been used. This pattern allows to create a system of decoupled apps, each individually replaceable to adapt to current needs of the scenario without changing other services, that perform the task of Linked Data generation.

This generic approach of raw sensor data acquisition and Linked Data generation can also be used to solve the challenge of integration of existing data from non Linked Data sources. If successfully applied in manufacturing environments, it enables automatic recording of occurring events on the shop floor. If the recording of events can take place automatically, a holistic information view of the system is created. This holistic view is the starting point for the solution of the second challenge addressed by this architecture, the availability and readability of information about relationships between IIoT devices.

This solution combines Linked Data enabled user interfaces, called visualization apps, through common user contexts. Inside these common user contexts, users can create complex search and retrieval use cases, that give controllers and maintenance workers a better view of the shop floor. User contexts also allow searching for information on the shop floor by navigating different representations. This is done by performing certain events in other visualization apps in the same user context.

The architecture based on this visualization solution is evaluated in a user study. During the study participants had to perform different tasks of shop floor maintenance. To successfully perform these tasks a participant has to learn how to use the visualization system to examine different entities and their relationships. The results of the evaluation are, that the system has a good usability with a SUS score of 72 points. It also confirms the fulfillment of the requirements set in the architecture design. The results of the evaluation state that the system can improve a users understanding of relationships on the shop floor.

8.2 Future Work

This section suggests possible starting points for future work that is based on the work done in this thesis.

Extending the Evaluation Scope

The conducted evaluation is focused primarily on usability and effectiveness of the system. In the future evaluations should also focus on other areas relevant for usage on the shop floor. This could be the improvement of structural understanding of the advantages of interlinked information wikis in combination with shop floor apps.

To reproduce or use the system, one can conduct sources mentioned in chapter 1.4.

Additional Visualization Apps

Several Apps can be added to the system to allow the user to perform a new range of complex workflows. A few suggestions are the following.

1. Timeline app allows replaying of every recorded action taking place on the shop floor in a 3D model viewer.
2. A Wiki, explaining all necessary terms for the user to properly use the system, while also providing a collection of best practices for every task editable directly inside the app by a qualified user.
3. Log-file viewer for machines.

Augmented Reality on the Shop Floor

Augmented reality can be used to enrich the visual field of users that operate on the shop floor with Linked Data. This can be combined with existing visualization apps on the shop floor trough user contexts. Visualization apps in the same user context can be used to navigate the augmented shop floor environment easily. It is planned to develop such an application in the future. This application will combine the already existing visualization apps with an augmented reality of the shop floor environment. The goal of the application is to explore new ways of information exploration on the shop floor.

8.3 Outlook

Simulated Shop Floor Environment

The idea of a simulated shop floor environment is to record every event contributing to the shop floor to run a simulation of complex shop floor processes.

By combining the recorded Linked Data, the information stored about existing entities on the shop floor and machine learning methods to simulate a shop floor. The factors playing a role on the shop floor are huge, but so are the always increasing amounts of Linked Data collected on them.

If more information and meta-information becomes available about everything happening on the shop floor, it may be possible in the future to create complex shop floor simulations behaving similar to real-world environments, which they are simulating. This would allow to test more shop floor systems with less effort and as an implication remove hurdles in the process of developing such systems.

Outlook on Future Shop Floor Systems

Future shop floor systems will gain more and more autonomy from their human operators. This will increase the development of machine-to-machine communication in manufacturing environments. But also the degree of which machines are able to perform monitoring and maintenance tasks and analysis completely without human intervention.

There already exist good machine learning algorithms allowing more autonomous operation of maintenance tasks. For example, the solution for the problem of fault diagnosis as described in [GDZX04] solves the problem of fault diagnosis in sheet metal stamping processes by applying support vector machines.

Bibliography

[Ala13] A. P. R. A. Alaa F. Sheta, *Business Intelligence and Performance Management*. Springer, 2013.

[Aue14] S. Auer, "Introduction to lod2," in *Linked Open Data–Creating Knowledge Out of Interlinked Data*. Springer, 2014, pp. 1–17.

[BFKT14] J. Bonér, D. Farley, R. Kuhn, and M. Thompson, "The reactive manifesto," 2014.

[BHBL09] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *International journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22, 2009. [Online]. Available: http://eprints.soton.ac.uk/271285/

[BKM08] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the system usability scale," *Intl. Journal of Human–Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008.

[Bro96] J. Brooke, "SUS - A quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996. [Online]. Available: http://hell.meiert.org/core/pdf/sus.pdf

[Bur16] B. Burns, "Design patterns for container-based distributed systems," *USENIX Workshop on Hot Topics in Cloud Computing*, 2016.

[Com12] O. Community, "Oslc primer," *The Gerontologist*, vol. 52, no. 6, pp. NP–NP, 2012. [Online]. Available: http://open-services.net/primer/index.php{#}primer{_}main{%}5Cnhttps://academic.oup.com/gerontologist/article-lookup/doi/10.1093/geront/gns026

[EP17] M. Early and A. Program, *Microservices in . NET Core with examples in NancyFX MEAP Edition*. Manning Publications, 2017.

[GDZX04] M. Ge, R. Du, G. Zhang, and Y. Xu, "Fault diagnosis using support vector machine with an application in sheet metal stamping operations," *Mechanical Systems and Signal Processing*, vol. 18, no. 1, pp. 143–159, 2004.

[GTW10] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the web of things," in *Internet of Things (IOT), 2010*. IEEE, 2010, pp. 1–8.

[HG08] J. Hendler and J. Golbeck, "Metcalfe's law, web 2.0, and the semantic web," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 1, pp. 14–20, 2008.

[HM00] J. Hendler and D. L. McGuinness, "The darpa agent markup language," *IEEE Intelligent systems*, vol. 15, no. 6, pp. 67–73, 2000.

[Int] Interconics, "Interconics automated-optical-inspection," http://fastpcbs.com/wp-content/uploads/2016/07/interconics-Automated-Optical-Inspection-sub.jpg, accessed: 2018-01-30.

[JF14] A. Jena-Fuseki, "serving rdf data over http," 2014.

[KOM05] A. Kiryakov, D. Ognyanov, and D. Manov, "Owlim–a pragmatic semantic repository for owl," in *International Conference on Web Information Systems Engineering*. Springer, 2005, pp. 182–192.

[Kö17] O. König, "Ontology-based information flow control and visualization in an industry 4.0 scenario," https://publikationen.bibliothek.kit.edu/1000078304, 2017.

[Ler14] R. M. Lerner, "At the forge: 12-factor apps," *Linux J.*, vol. 2014, no. 245, Sep. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2682554.2682559

[LHS08] B. Laugwitz, T. Held, and M. Schrepp, "Construction and evaluation of a user experience questionnaire," in *Symposium of the Austrian HCI and Usability Engineering Group*. Springer, 2008, pp. 63–76.

[LSH+09] A. Latif, A. U. Saeed, P. Hoefler, A. Stocker, and C. Wagner, "The linked data value chain: A lightweight model for business engineers," *Proceedings of ISEMANTICS09 International Conference on Semantic Systems*, pp. 568–575, 2009. [Online]. Available: http://www.i-semantics.at/2009/papers/the{_}linked{_}data{_}value{_}chain.pdf

[MCS+16] A. Miclaus, W. Clauss, E. Schwert, M. A. Neumann, F. Mütsch, T. Riedel, F. Schmidt, and M. Beigl, "Towards the shop floor app ecosystem: Using the semantic web for gluing together apps into mashups," in *Proceedings of the Seventh International Workshop on the Web of Things*. ACM, 2016, pp. 17–21.

[Met95] B. Metcalfe, "Metcalfe's law: A network becomes more valuable as it reaches more users." *Infoworld*, vol. 17, no. 40, pp. 53–53, 1995.

[MMS14] C. Mader, M. Martin, and C. Stadler, "Linked Open Data – Creating Knowledge Out of Interlinked Data," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8661, pp. 90–107, 2014. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84927597423{&}partnerID=tZOtx3y1

[NK13] J. Newton-King, "Json .net," *Online] http://www. newtonsoft. com/json*, 2013.

[Pra01] M. J. Pratt, "Introduction to {ISO} 10303—the {STEP} {Standard} for {Product} {Data} {Exchange}," *Journal of Computing and Information Science in Engineering*, vol. 1, no. 1, pp. 102–103, 2001. [Online]. Available: http://dx.doi.org/10.1115/1.1354995

[PSHS10] H. K. Patni, S. S. Sahoo, C. A. Henson, and A. P. Sheth, "Provenance aware linked sensor data," 2010.

[RDE+07] K. Rohloff, M. Dean, I. Emmons, D. Ryder, and J. Sumner, "An evaluation of triple-store technologies for large data stores," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2007, pp. 1105–1114.

[RvdM13] J. Rivera and R. van der Meulen, "Gartner says the internet of things installed base will grow to 26 billion units by 2020," *Stamford, conn., December*, vol. 12, 2013.

[sca] "Scaleit - online documentation," http://scaleit-platform-documentation.readthedocs.io/, accessed: 2018-01-04.

[SGS+16] A. Soylu, M. Giese, R. Schlatte, E. Jimenez-Ruiz, Ö. Özçep, and S. Brandt, "Domain experts surfing on stream sensor data over ontologies," *CEUR Workshop Proceedings*, vol. 1588, pp. 11–20, 2016.

[Shi10] J. Shinavier, "Real-time# semanticweb in<= 140 chars," in *Proceedings of the Third Workshop on Linked Data on the Web (LDOW2010) at WWW2010*. Citeseer, 2010.

[sho] "Shop floor ontology," https://github.com/ScaleIT-Org/Ontology-based-InformationFlow-Industry-4.0, accessed: 2018-01-10.

[TKKF14] B. Tilahun, T. Kauppinen, C. Keßler, and F. Fritz, "Design and development of a linked open data-based health information representation and visualization system: Potentials and preliminary evaluation," *Journal of Medical Internet Research*, vol. 16, no. 10, p. e31, 2014.

[TLN06] C. W. Turner, J. R. Lewis, and J. Nielsen, "Determining usability test sample size," *International encyclopedia of ergonomics and human factors*, vol. 3, no. 2, pp. 3084–3088, 2006.

[twe] "The twelve factor app," https://12factor.net/, accessed: 2017-12-10.

[W3C] W3C, "W3c - web of things," https://www.w3.org/standards/, accessed: 2018-01-15.

[ZGPU12] J. Ziegler, M. Graube, J. Pfeffer, and L. Urbas, "Beyond app-chaining: Mobile app orchestration for efficient model driven software generation," in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. IEEE, 2012, pp. 1–8.

[ZLEKS17] N. S. Zadeh, L. Lindberg, J. El-Khoury, and G. Sivard, "Service oriented integration of distributed heterogeneous it systems in production engineering using information standards and linked data," *Modelling and Simulation in Engineering*, vol. 2017, 2017.

Appendix

Library	Version	URL
jQuery	3.3.1	https://jquery.com
jQuery UI	1.12.1	https://jqueryui.com/
D3.js	3.4	https://d3js.org/
Heatmap.js library	1.0	https://github.com/pa7/heatmap.js
Node js	8.9.4 LTS	https://nodejs.org/en/
NPM websocket package	1.0.25	https://www.npmjs.com/package/websocket
Python	3.6.4	https://www.python.org/downloads/release/
Python	2.7.14	https://www.python.org/downloads/release/python-2714/
Docker-py	2.7	https://docker-py.readthedocs.io/en/stable/
Python simplehttpserver	2.7	https://docs.python.org/2/library/simplehttpserver.html
DOT NET core	2.0	https://github.com/dotnet/core
Newtonsoft.Json	10.0.3	https://www.nuget.org/packages/Newtonsoft.Json
DotNetRdf	2.0.1	https://www.nuget.org/packages/dotNetRDF/
RomanticWeb	1.0.0-rc1	https://www.nuget.org/packages/RomanticWeb/1.0.0-rc1
Nancy fx framework	1.4.4	https://github.com/NancyFx/Nancy/
OWIN	1.0	http://owin.org/html/spec/owin-1.0.html
SPARQL	1.1	https://www.w3.org/TR/sparql11-query/
Fuseki	2.0	https://jena.apache.org/documentation/fuseki2/
Jena TDB	1.1.0	https://jena.apache.org/documentation/tdb/index.html
Docker	17.12	https://docs.docker.com/

Figure .1: Software libraries used in implementation, version numbers during usage and source URLs.

Heat Maps

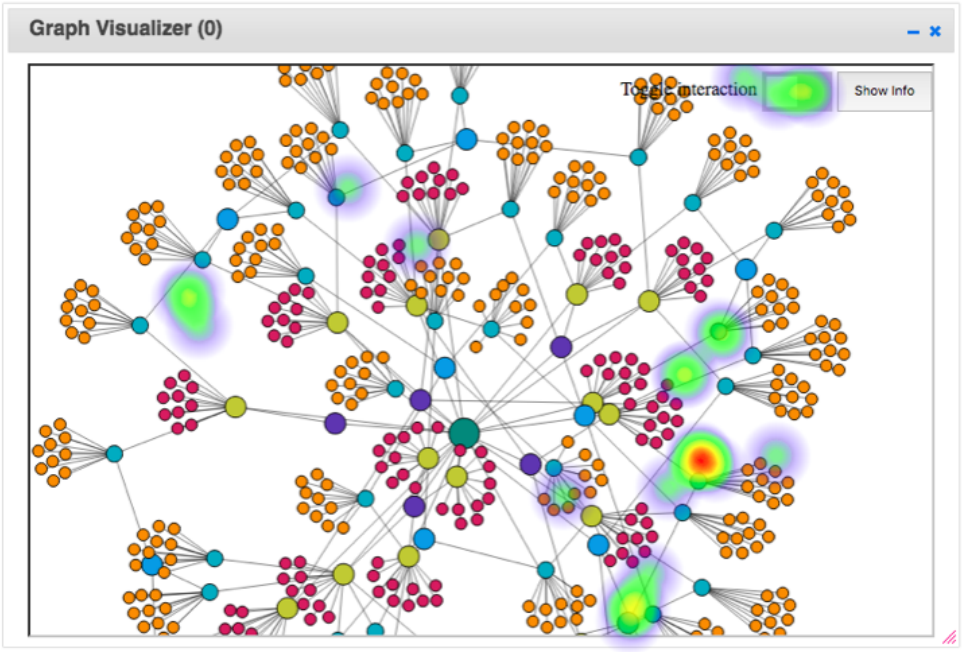


Figure .2: Graph view with overlaid heat map of clicks.

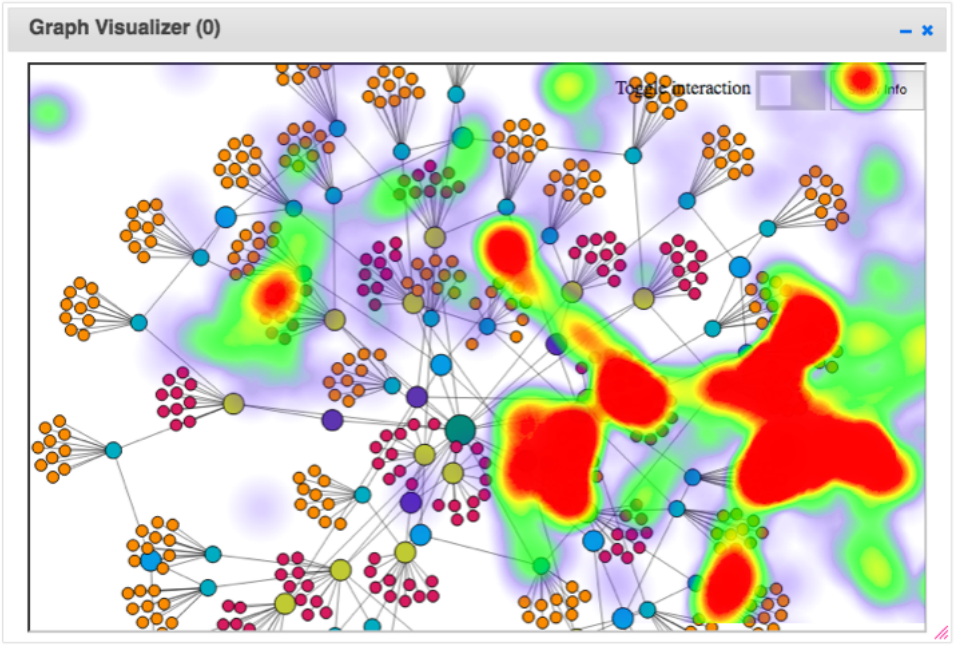


Figure .3: Graph view with overlaid heat map of hover movements.

Figure .4: List view with overlaid heat map of mouse hover movements.

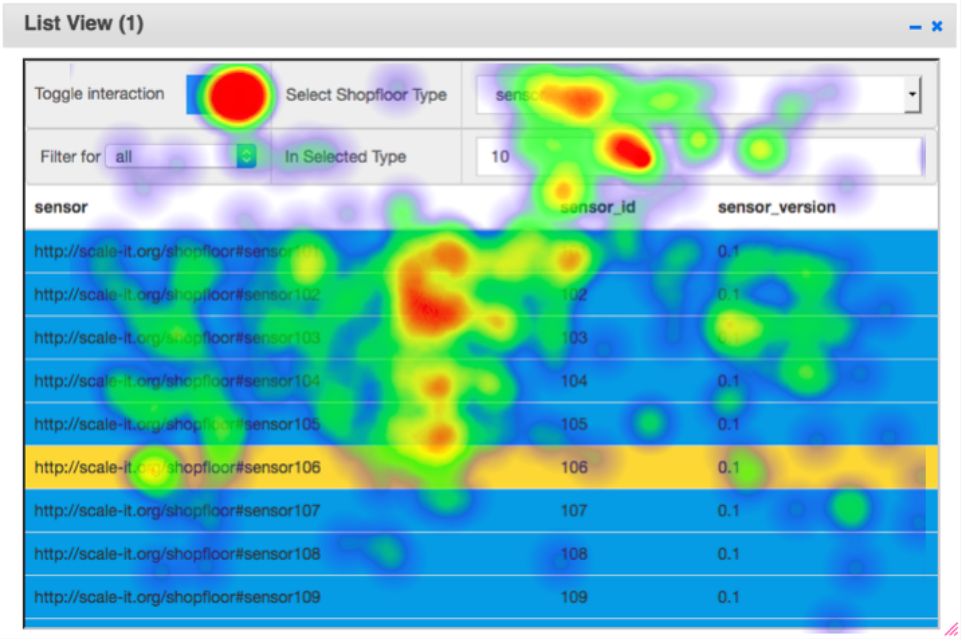


Figure .5: List view with overlaid heat map of mouse hover movements.

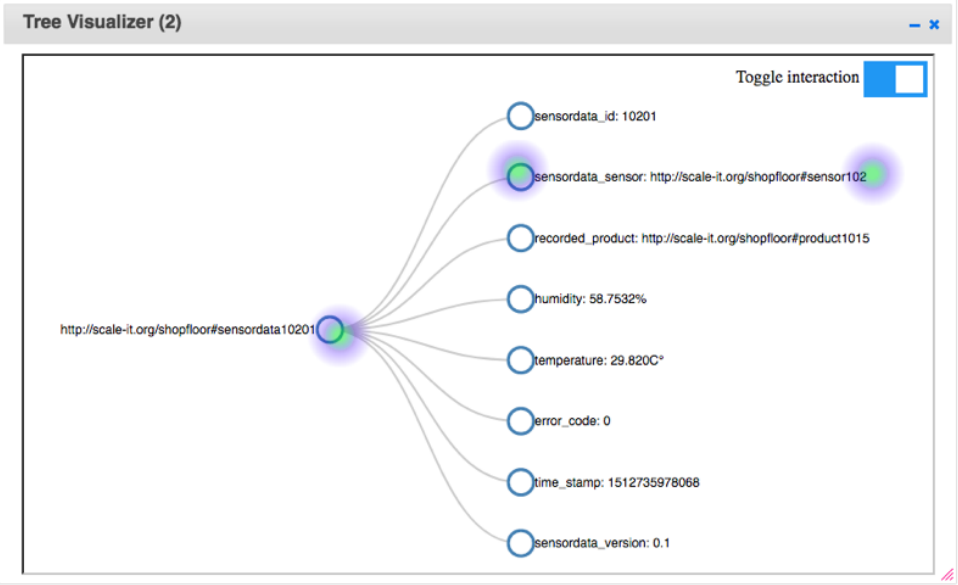


Figure .6: Tree view with overlaid heat map of mouse clicks.

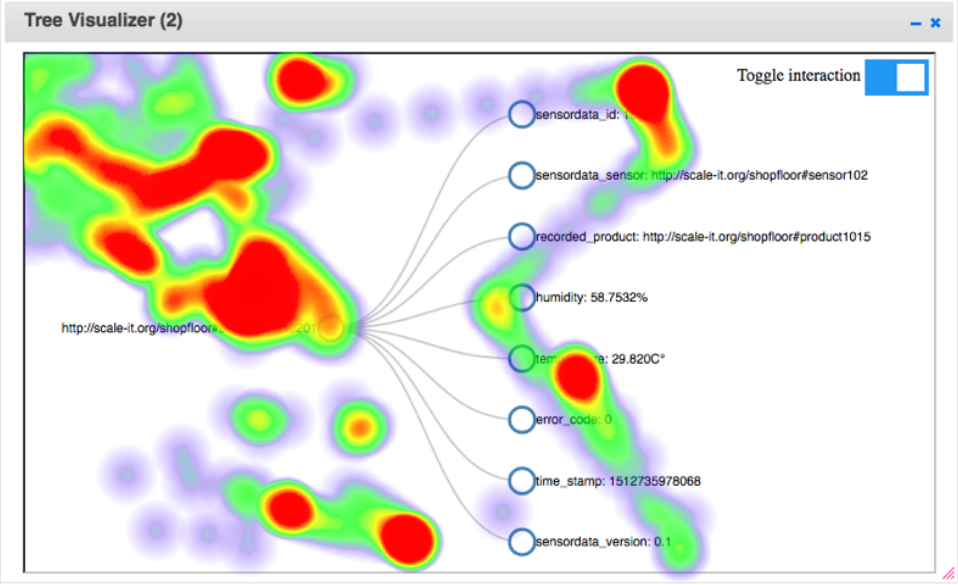
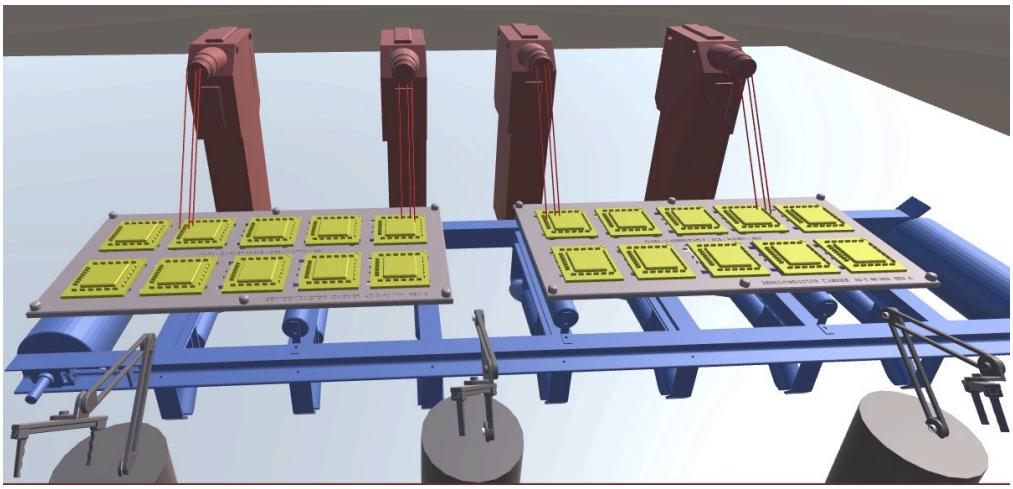


Figure .7: Tree view with overlaid heat map of mouse hover movements.

Linked Data Graph – Shop Floor Scenario:



Shop floor scenario:

Scenario – challenges of the shop floor:

On this shop floor, printed circuit boards (PCB) are soldered and baked, every PCB consists of a board with ten chip components(capacitors, transistors, resistors). After every production step sensors observe the board and its components to ensure everything is working properly, and to detect,if a machine starts to produce errors. Each production line (*blue*) consist of several machines (*grey*), sensors (*red*) and products (light grey) with components each (*yellow*).

Video of PCB production:

Production lines, machines and sensors:

The shop floor is divided in different production lines. Along each line, sensor and machines are placed. Sensors record information about product at different positions in the production line.

Products and product components:

Each product(PCB) consists of a board with ten chip components(capacitors, transistors, resistors).

Sensor data and sensor data values:

Each record of a product is represented as sensor data – it contains general information about a sensor record like error codes or temperature of the board. Each sensor data also has ten sensor data value representing sensor record information about single board components.

You are a shop floor operator:

You work on the shop floor and have the responsibility of

delivering the highest quality PCBs. This is achieved through monitoring the production and identifying error sources, which can then be fixed by your staff.

Your tasks:
You have to perform three different task to help the shop floor manager solve errors occurring during production on the shop floor. He needs specific information about the currently occurring errors in his production lines. You have to find information about the errors and information about other effects happening on the shop floor to understand the connection between changes applied to different stages of the production process on an assembly line.

The particular tasks are the following: 1. Warm Up Task: How many sensor data instances exist for the product with the id 104 2. Find the machine processing the most error-free products(the error code not equal to zero). 3. Find the machine processing the most error products.

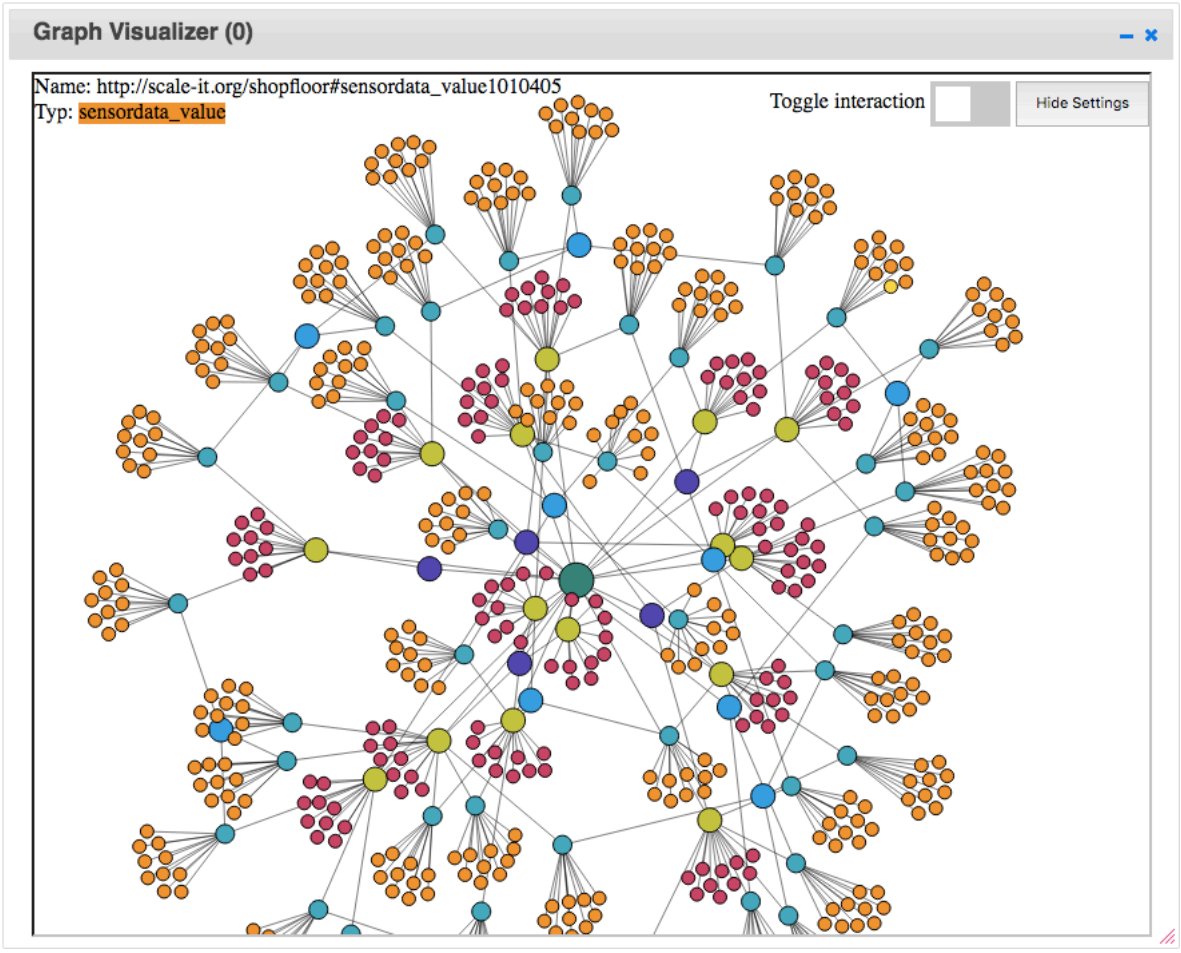
Your Tools:

To solve this tasks the shop floor manager provides you different apps specialized to work with shop floor data. For each task, you have to use a specific combination of them to achieve the desired result.

The apps are:

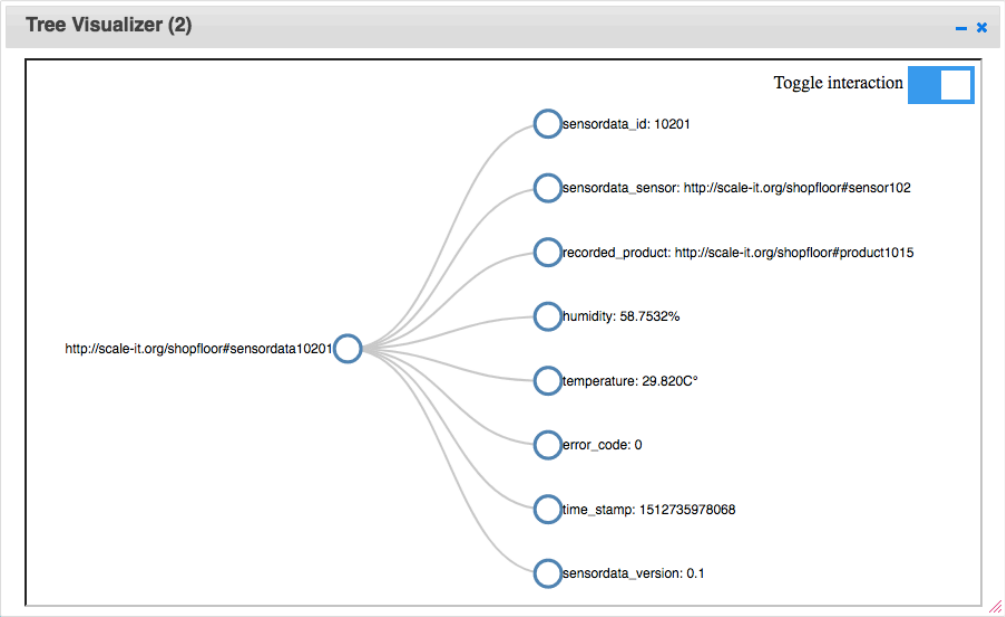
1. The graph view app:
The graph view app displays an undirected graph, representing all instances of the shop floor as nodes and their relationships as links. It is possible to hide node and link

labels but also specific entity types to allow focusing only on the relevant entities.



2. The tree view app:

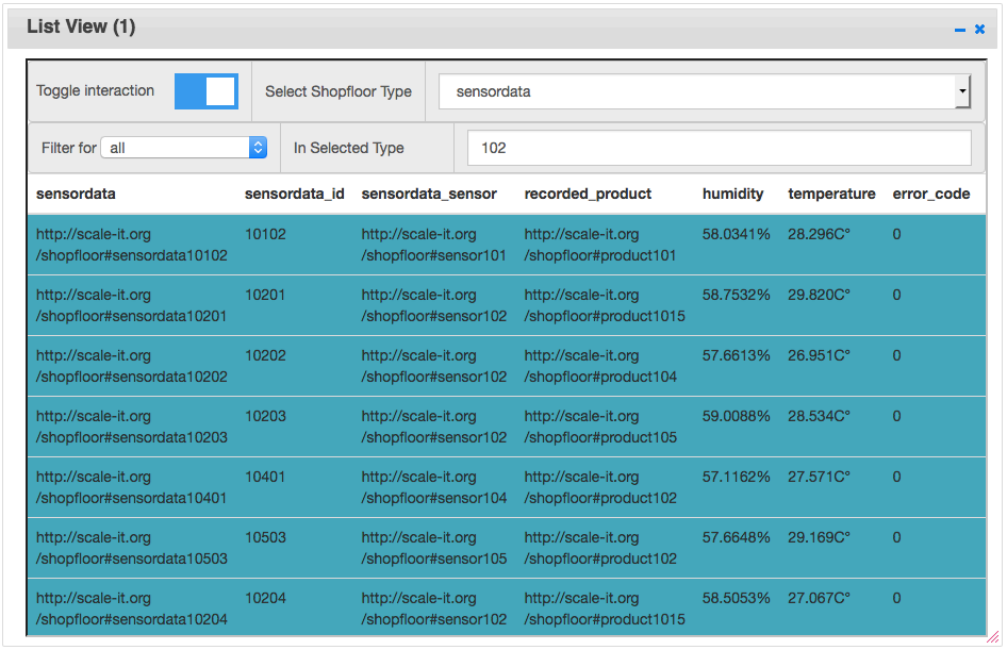
The tree view displays detailed information about a specific entity on the shop floor like the error codes of a sensor data value.



List view App:

3. The List View:

The list visualizer provides searchable lists of all entities. Each entity is represented as a row.

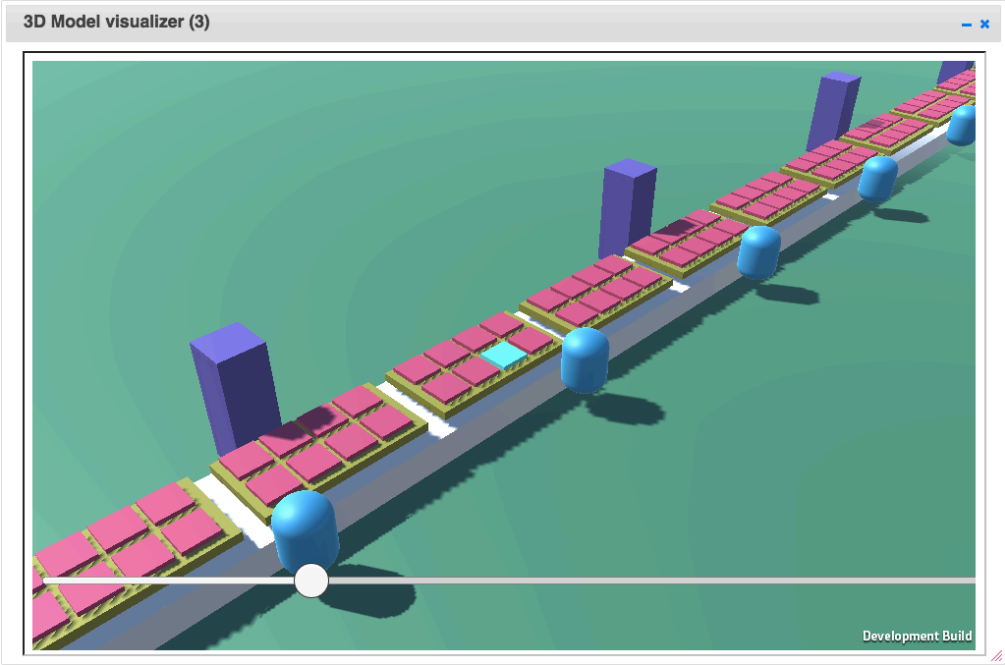


List view App:

4. The note view app: (Write your results here):

Use the note view to record the solution of every task. You

can use the copy-selected-to-clipboard-button (+ pressing ctrl-C and enter) to copy the currently selected app to the clipboard to easily paste it in your results.



List view App:

5. The 3D view app: (not used during this scenario)

The 3D model view app shows a three-dimensional representation of the shop floor and all its physical(!) entities. The user can navigate along the production lines and examine single entities by clicking on them.

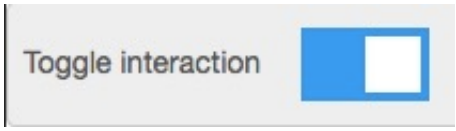
Combining tools:

Each selection of an entity in one app sends a message to all the other apps and allows them to act accordingly. For example, if the list and the 3D model visualizer is opened – clicking on a product in the list will select the product in the 3D model and centres the camera on it if an interaction is toggled in the app. To toggle interaction press the grey slider:

1. Toggle interaction turned off:

Toggle Off

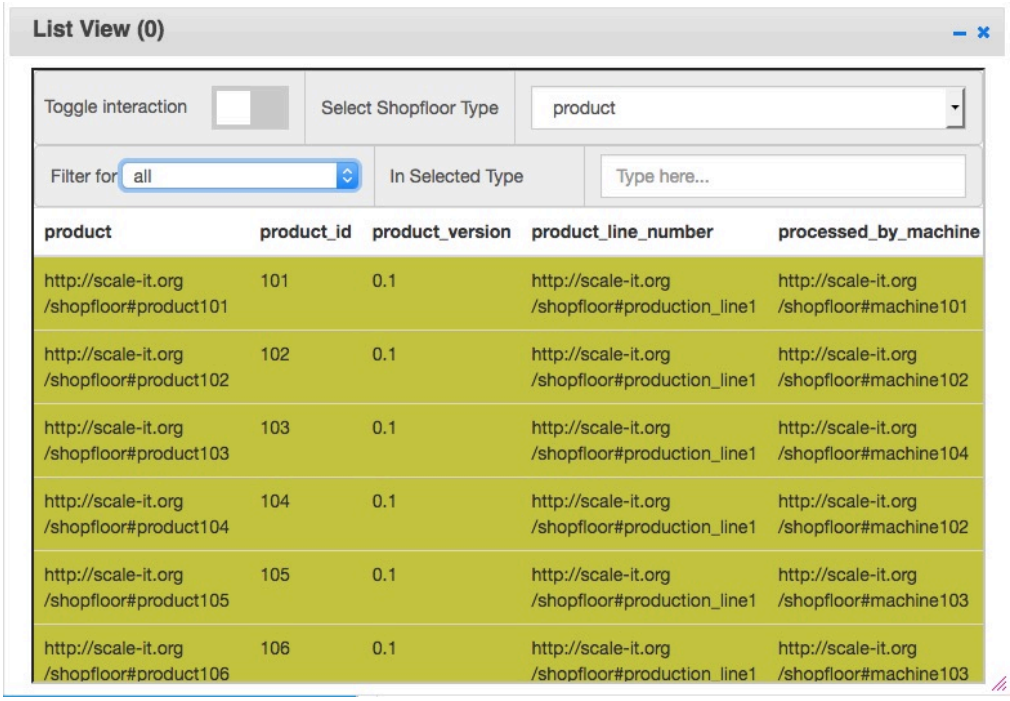
2. Toggle interaction turned on:



Toggle On

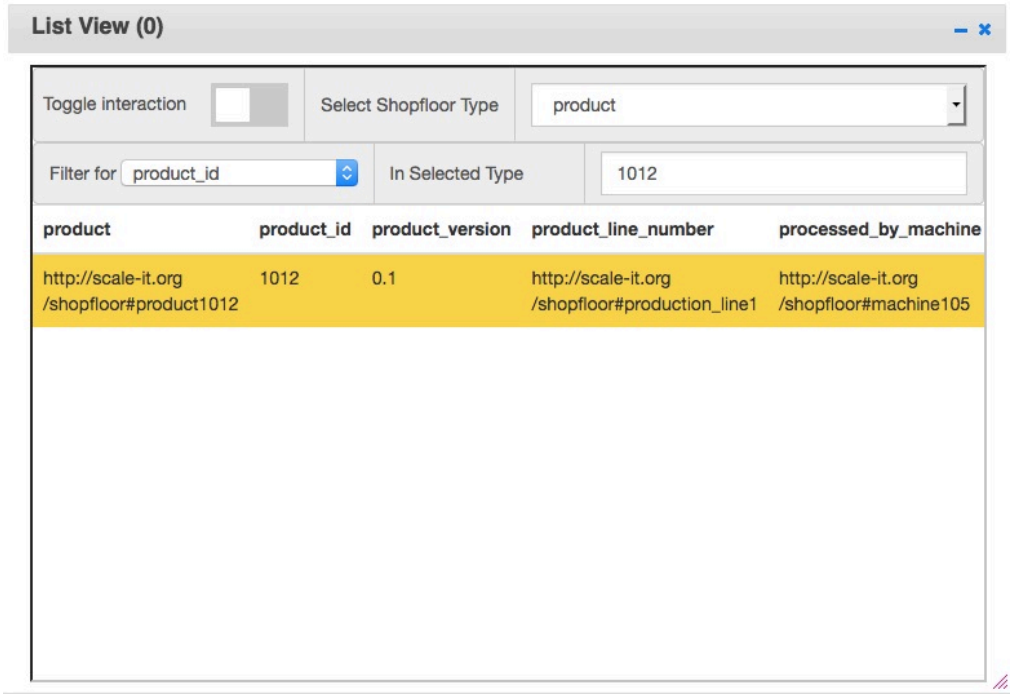
Example workflow for looking up product components ids of product with id 1012:

1. Open a list view app instance



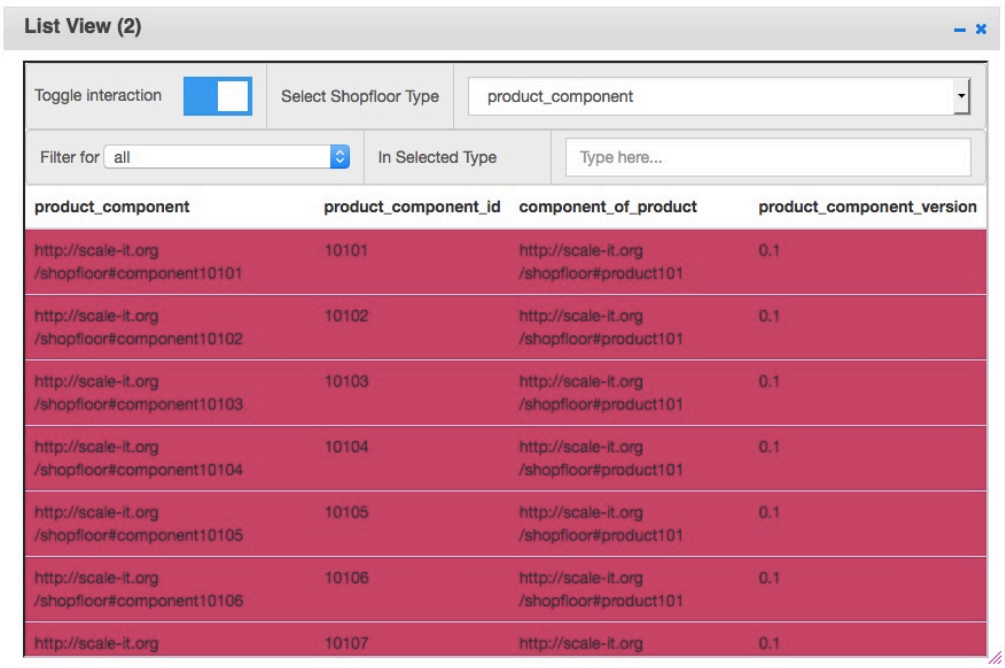
Open a ListView app:

1. Select filter by id and type the id ‘1012’ in the search box.



Select filter by id and type the id 1012 in the search box:

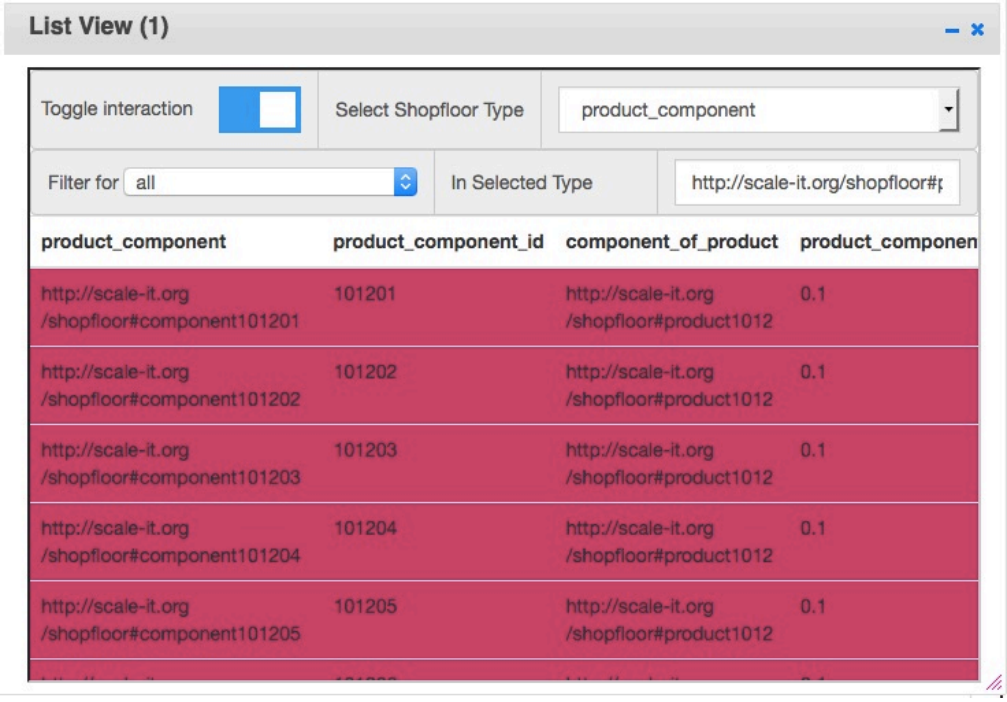
1. Open a second list view app instance and toggle its toggle interaction slider.



Open a second Listview app instance and slide the toggle interaction slider:

1. With a click on the search result in the first list view

app instance you now get all needed product components listed.



With a click on the search result in the first Listview instance you now get all needed product components listed:

To start the tasks click this link to open the tool dashboard

Tasks:

1. Example task: Look up all product component ids of product ‘product1012’.
2. Warm up task: How many sensor data recordings exist for product ‘product104’?
3. Find a machine that processed the most products, for which no errors have been recorded.
4. Find a machine that processed the most products, for which one or more errors have been recorded.

After your done with all tasks: Click here to start the questionnaires.