Master Thesis

Generic Frontend for Exploring Sensor and Information Services

Author:
M.Sc. Uliana Andriieshyna
Matrikel-Nr: 3828303

Supervisors:
Dr.-Ing. Josef Spillner
Prof. Dr. rer. nat. habil.
Dr. h. c. Alexander Schill

April 10, 2014

Declaration of Authorship

I, Uliana Andriiehyna, declare that this thesis, titled "Generic Frontend for Exploring Sensor and Information Services", and the work presented in it have been done on my own without assistance. All information directly or indirectly taken from external sources is acknowledged and referenced in the bibliography.

Dresden, XX.XX.2013 ____________________

Contents

Chapter 1

Introduction

The increasing numbers of sensor devices has increased the number of sensor-specific protocols, platforms and software. The definition "sensor" consists not only a phisical device that can be an actuator or measuring device, but also a contest-depending information that are available through the Internet, e.g. Facebook, RSS Feeds, Weather Forecast and etc. As a result various approaches have been proposed to interconnect, maintain and monitor various type of data sources[1, 2, 3].That specifically focused on a platform development, protocol definition and software architecture for the concrete user-oriented requirements and for a narrowly focused areas of usage instead of defining a common system approach. Mainly in proposed approaches was discovered such type of questions as security and privacy betwenn sensor's protocol and platform, platform development itself, online monitoring of different data sources, social dimensions[4], that almost not focused on requirements and criteria of a user-friendly interface, that for user nowadays become one of the mnost important thing. Therefore the area of research of this master thesis is dedicated to define generic frontend for exploring data sources, thwt can be easily and universaly interconnected to an any platform or system without any sort of Graphical User Interface(GUI). The following sections ground the motivation for the chosen research field, define the central research questions and goals of this master's thesis, and describe the overall structure of the work.

1.1 Motivation

In the recent years with the technological progress in the web technologies, information systems, and in particular sensor data systems, have become an essential part in daily life of the modern society. More and more aspects of human life is shifted to the Web. This allows fast and easy way to get information from any point on earth. In the same time increased number of platform and interfaces. People have started to use them more often not only for manufacture, business, education but also for private reasons. Currently, most of the research is concerned with the protocol and middleware levels, whereas the potential of a generic interactive access to sensor and information services needs to be explored in a best comfortable way for user and developer also. In this master thesis, a first universal web-based frontend with a main focus on a GUI is to be created. Users should be able to explore not just the information provided by Web services, but also from a real sensors around them. The system architecture of such a concept should be distributed in such a way, that already implemented projects could easily interconnect with it.

Creating composite third-party services and applications from reusable components is an important technique in software engineering and data management. Although a large body of research and development covers integration at the data and application levels, weak work has been done to facilitate it at generic level. This master thesis discusses the existing user interface frameworks and component technologies used in presentation integration, illustrates their strengths and weaknesses, and presents some opportunities for future work.

1.2 Addressed Use Cases

In order to define concrete research questions, it is important to determine which kind of concept is going to be implemented as well as to clarify the common terms which are used throughout the thesis:

- **Application class.** This thesis is focused on a general multi-tier web-based prototype that delivers dynamic content to end-users via RESTful API.
- **Platform.** Implies a virtualized hardware architecture and a software stack required for running an application of the respective class.
- **Infrastructure.** Assumes a set of web-based protocols and sensors streaming resources used to maintain services of the aforementioned application class.

1.3 Research Questions and Goals

As mentioned in the previous section, there are already exist many solutions for creating sensor-aware applications. But these platforms focuses on a single area of usage and they are not commonly suitable to support the dynamic and adaptable composition and usage of different type of sensors in one dashboard.

Within this thesis the following research questions should be answered in order to design, implement and evaluate the generic frontend for retrieving data sources:

- Which architecture should the generic frontend have?
- What type of data sources are have to be retrieved and what is the universal interface for collaboration between backend and frontend system?
- How can be the general GUI designed and implemented?
- Which software components might be applied to the concept in order to be applicable to the most available data sources and platform?

Therefore, this thesis is aimed at the development of a concept that provides users a possibility to personalize their current environment indepently from any type and kind of devices.

1.4 Structure

After the introduction in *Chapter 1* the thesis is structured in the following way:

Chapter 2 defines the background of the master's thesis describes the basic used terminology and the foundation platforms. A reference scenario and the requirements to a concept that has to be developed are also introduced in this chapter.

Chapter 3 is devoted to the state of the art analysis. The related research works in the areas of the sensor-retrieving approaches, the component based groupware systems, the browser based and non-browser based systems are investigated and evaluated against the defined requirements.

Chapter 4 focuses on the concept of the generic Frontend for exploring sensor and Information services, considering possible approaches, strategies, frameworks and necessary criteria, defined in *Chapter 3*.

Chapter 5 provides the implemented functionality of the concept and describes evaluation of results. Important methods and libraries and interfaces are described in details.

Chapter 6 concludes the master's thesis underlining and evaluating achieved goals and providing prospects for the possible future work. Two test cases are proposed, described and performed in order to cover the fulfillment of the defined requirements by the developed solution. Afterwards the results of the evaluation are discussed.

Chapter 2

Foundations and Requirements Analysis

The main goal of the following chapter is a definition of requirements to generic frontend for exploring sensor and information services. To achieve this, fundamental terms in the presented research area are formulated at the beginning of the chapter. Basic aspects concerning data sources definition, types of information services in the web that have to be retrieved are described in the section 2.2. According to aforementioned parts of an approach, summary section underlines the chapter by combining needs of frontend and data sources and defines requirements to a common concept. Such a concept described in details in the chapter 4.

2.1 Frontend's Requirements

In computer science, the frontend is responsible for collecting input in various forms from the user and processing it to conform to a specification the backend can use. The frontend is an interface between the user and the backend[5] and the separation of software systems into front- and backends simplifies development and separates maintenance. Therefore need to be distinguished what are the main requirements to a generic frontend for exploring data sources, i.e. :

- *Loose coupling:* each of systems components has, or makes use of, little or no knowledge of the definitions of other separate components. Where the main goal is to avoid dependencies between new components, modules and easy deployment of future enhancement.

- *Fine-grained structure:* split the system into a small parts (Logic Modules, universal interfaces to any type of data format, Information resources in Web), such that it can be distributed across internet, and can be apllied to any resource and mobile device everywhere. It is also include ease of expansion and integration with backend system, independently from platform used on a backend and web frameworks used on a frontend

- *Multi-user binding:* defining to every user according to their type and rights visibility rules and concrete interface view targeting by using JID. Every account have to be unique binded to any data source, contracts, personalized preferences. After that, GUI can be targeted according to it.

- *Cross-platforming or multi-platform:* a possibility of application to be run on any type of device(e.g., smartphone, notebook, tablet) without special preparation or changes. this gives an possibility to be flexible.

- *Responsive web design:* an ability of a GUI automatically adapt to any size of device screen, by provisioning high usability performance.

- *Usability:* web-based interface, that can be easily explored in social platfroms and connects any type of news feeds.

2.1.1 Fine-grained structure

The fine-grained structure of a sytem provides a possibility to distribute task between responsible modules of a system. It guarantees performing tasks in parallel, moreover, exchange, enhance and add new part of a system, without influence to an any another module. Thus individual tasks are relatively small in terms of code size and execution time. The data is transferred among processors frequently in amounts of one or a few memory words. Also it garantees necessary performance in response time. In order to attain the best parallel performance, the best balance between load and communication overhead needs to be found. If the granularity is too fine, the performance can suffer from the increased communication overhead. On the other side, if the granularity is too coarse, the performance can suffer from load imbalance.

Hereby, generic frontend have to be splitted to a modules, responsible for a separated task, that in the same time gives a possibility to expand the system.

2.1.2 Loose Coupling

The goal of loose coupling is to reduce dependencies between systems. And this term mostly related to an internal software structure of the system, e.g. classes, patterns, asynchronous requests. But to make it clear, why generic approach in building dynamic frontend have to be loosely coupled it is necessary to compare tight- and loose coupling in Table 2.1[1]

[1] A High-level Comparison of Tight and Loose Coupling,`http://wiki.scn.sap.com/wiki/display/BBA/Loose+Coupling+Through+Web+Services`

Target	**Tight Coupling**	**Loose Coupling**
Physical Connection	Point-to-Point	Via mediation (a)
Communication Style	Synchronous	Asynchronous
Data Model	Complex common types	Simple common types
Service Discovery and Binding	Static	Dynamic
Dependence on Platform-Specific Functionality	Strong/many	Weak/few
Interaction Pattern	Via complex object trees	Via data-centric, self-contained messages
Transactional Behavior	Controlled by a central transaction manager (e.g., two-phase commit)	Compensation (b)
Control of Process Logic	Central control	Distributed control
Deployment	At the same time (c)	Different point in time if desired
Versioning(d)	Explicit/forced upgrades	Implicit upgrades

Table 2.1: A High-level Comparison of Tight and Loose Coupling

Where:
a. Mediation implies that some mechanism must handle communication between the composite application and the backend system, filling the role of the service contract implementation layer.
b. Compensation means that for every modifying service, a dedicated compensational service must explicitly be developed for rollback purposes. If the modifying service is part of a service chain that has to be executed as one transaction and an error occurs, the compensational service helps to undo the first modifying operation and sets the system back to its initial state.
c. For tightly coupled applications, the parts must always be deployed at the same time. Loosely coupled applications do not have this requirement (though of course the parts could be deployed at the same time). Consider an interface change for a web service that performs a write. When a tightly coupled service is called, if the interface has changed, it simply will not work. A loosely coupled service operation will write the data to the service contract implementation layer and continue. Once the new write operation is in place, it can handle all buffered calls.
d. Consider versioning of services. If a service provider changes the interface of a service, it is probable that not all consumers can update their applications at the same time. In case of tight coupling, all consumers must explicitly update their applications as well. With loose coupling, the provider offers separate versions at the same time so that consumers don't have to update their applications. Alternatively, the provider supports the new functionality

behind the old interface and fills new parameters with default values (an implicit upgrade that does not affect the consumer).

Table 2.1 shows how important is to realize loose coupling in generic sytems, namely frontend, to make it dynamic, independent and distributed.

The degree of the loose coupling can be measured by noting the number of changes in data elements that could occur in the sending or receiving systems and determining if the computers would still continue communicating correctly[6, 7]. These changes include items such as:

- adding new data elements to messages
- changing the order of data elements
- changing the names of data elements
- changing the structures of data elements
- omitting data elements

Benefits of loose coupling include flexibility and agility. A loosely coupled approach offers unparalleled flexibility for adaptations to changing landscapes. Since there are no assumptions about the landscape your application is running against, you can easily adapt the composite application as needed. This is especially important for ISVs and system integrators who develop applications once and install and configure them at diverse customer sites. The application itself stays untouched. Another aspect to consider is the probability of landscape changes during the lifetime of the application. Due to mergers and acquisitions and system consolidations, the landscape underneath the application is constantly changing. Without loose coupling, you'll be forced to adapt your application again and again. In essence, loose coupling means reducing the number of assumptions to a bare minimum. The goal of loose coupling is to minimize dependencies between systems. However, note that loose coupling comes at a price—every form of loose coupling has disadvantages (especially with regard to complexity) that must be considered.

2.1.3 Multi-User Binding

In order to provide better user experience needed to be distinguish main user types of a system itself:

- Backend developer, that needs an acceptable interface for backend and adaptable modules for its data structure, to easily integrate already existed backend with new frontend. Comprehensive and detailed service descriptions is a first entry point for developers, who starts to work with web service. Well or poorly described service measures the amount of efforts, made by developers
- Administrator of a system, that have to add new functionality and modules/blocks to GUI or change already existed. Administrator have to have an experience with defined

- User of an application, that wants to see and control sensors and statistic

According to defined types of users, needed to be specified what are the main requirements from a user prospective to the researched frontend approach.
Backend developer, who starts to work with the web service, needs comprehensive and detailed service descriptions, because it is a first entry point for him. Well or poorly described service measures the amount of efforts, made by developers. Therefore generic frontend have to support common standards in a field of interface, protocols, approaches of web services.
Administrator plays a role of experienced user, that knows the purpose of a system, how it works, which frameworks and platforms have been used for implementation and how to help developer with integration. Also administrator have an opportunity to change frontend GUI and system structure intself from an inside.
After user login into a system, without any knowledge about a system, should be clarified what should be shown on a main screen and how to define usage scenario in an understandable for user way. In case of user task to sign contracts between him and resource provider, frontend have to provide examples of data, such that user can decide what is usefull for him and what is not.

In context of a generic frontend the term multi-user binding means to satisfy personalized users' requirements with corresponding credentials, preferences and personal settings. To use client-side data binding and dependency injection to build dynamic views of data that change immediately in response to user actions.

2.1.4 Cross-Platfroming

Because of the competing interests of cross-platform compatibility and advanced functionality, numerous alternative web application design strategies have emerged. Such strategies include:

Graceful degradation
Graceful degradation attempts to provide the same or similar functionality to all users and platforms, while diminishing that functionality to a "least common denominator" for more limited client browsers. For example, a user attempting to use a limited-feature browser to access Gmail may notice that Gmail switches to "Basic Mode", with reduced functionality. Some view this strategy as a lesser form of cross-platform capability.

Separation of functionality
Separation of functionality attempts to simply omit those subsets of functionality that are not capable from within certain client browsers or operating systems, while still delivering a "complete" application to the user(maybe separation of concerns).

Multiple code base
Multiple code base applications present different versions of an application depending on the specific client in use. This strategy is arguably the most complicated and expensive way to fulfill cross-platform capability, since even different versions of the same client browser (within the same operating system) can differ dramatically between each other. This is

further complicated by the support for "plugins" which may or may not be present for any given installation of a particular browser version.

Third-party libraries
Third-party libraries attempt to simplify cross-platform capability by "hiding" the complexities of client differentiation behind a single, unified API.

2.1.5 Responsive web design

Responsive web design (RWD)[8, 9] is a Web design approach aimed at crafting sites to provide an optimal viewing experience—easy reading and navigation with a minimum of resizing, panning, and scrolling—across a wide range of devices (from mobile phones to desktop computer monitors).

A site designed with RWD[10, 11]adapts the layout to the viewing environment by using fluid, proportion-based grids,flexible images, and CSS3 media queries, an extension of the @media rule.

- The fluid grid concept calls for page element sizing to be in relative units like percentages, rather than absolute units like pixels or points.
- Flexible images are also sized in relative units, so as to prevent them from displaying outside their containing element.
- Media queries allow the page to use different CSS style rules based on characteristics of the device the site is being displayed on, most commonly the width of the browser.
- Server-side components (RESS) in conjunction with client-side ones such as media queries can produce faster-loading sites for access over cellular networks and also deliver richer functionality/usability avoiding some of the pitfalls of device-side-only solutions.

2.1.6 Usability

Usability is the ease of use and learnability of a human-made object. In human-computer interaction and computer science, usability studies the elegance and clarity with which the interaction with a computer program or a web site (web usability) is designed. According to Jakob Nielsen, "Studies of user behavior on the Web find a low tolerance for difficult designs or slow sites. People don't want to wait. And they don't want to learn how to use a home page. There's no such thing as a training class or a manual for a Web site. People have to be able to grasp the functioning of the site immediately after scanning the home page—for a few seconds at most."[12] Otherwise, most casual users simply leave the site and browse elsewhere.

ISO defines usability as "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use." The word "usability" also refers to methods for improving ease-of-use during the design process. Usability consultant Jakob Nielsen and computer science professor Ben Shneiderman have written (separately) about a framework of system acceptability, where usability is a part of "usefulness" and is composed of[13]:

- Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
- Efficiency: Once users have learned the design, how quickly can they perform tasks?
- Memorability: When users return to the design after a period of not using it, how easily can they re establish proficiency?
- Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
- Satisfaction: How pleasant is it to use the design?

In scope of this master thesis will be covered such chracteristics of usability as: learnability, efficiency and satisfaction. Rest of it together with possible enhancement will be proposed in Chapter 6 in section Future Work.

2.2 Types of Data Sources

In order to distinguish what type of data sources can be retrieved, all data sources was splited to physical and software sensors.

2.2.1 Physical Sensors

A sensor is a converter that measures a physical quantity and converts it into a signal which can be read by a observer. After receiving this signal backend have to reconvert it to a signal that can be retrieved by frontend. The main task of frontend in this case to determine what are the main types of physical sensors, how to structurize and differentiate it according to their functionality and make a standard data format, by using which, backend and frontend can always easily integrate new sensors.

Together with software sensors the proposed data standard is justified and explained in Section 4.3.3

2.2.2 Software Sensors

Nowadays more and more physical information move to a Web by simplifying the way of sharing information. Already implemented thousands of public services such as: GPS coordination, weather forecast with all arising from this data, e.g. outside temperature, humidity, pressure etc., news feeds, traffic feeds, social networks. People got used to it, and today has become an inseparable part of their lives. Software sensor is not only an information from the web but all that have already implemented software support that can be used through the standard inteface.

An important research question is: how to manupulate data, made by different vendors, by using single dashboard. To define most common approach, used by vendors was made categorization[14].

Dimension	**Categories**	**Question to be answered**
Type	Web Crawler, Customizable Crawler, Search Engine, Pure Data Vendor, Complex Data Vendor, Matching Vendor, Enrichment Tagging, Enrichment Sentiment, Enrichment Analysis, Data Market Place	What is the type of the core offering?
Time Frame	Static/Factual, Up To Date	Is the data static or real-time?
Domain	All, Finance/Economy, Bio Medicine, Social Media, Geo Data, Address Data	What is the data about?
Data Origin	Internet, Self-Generated, User, Community, Government, Authority	Where does the data come from?Who is the author?
Pricing Model	Free, Freemium, Pay-Per-Use, Flat Rate	Is the offer free, pay-per-use or usable with a flat rate?
Data Access	API, Download, Specialized Software, Web Interface	What technical means are offered to access the data?
Data Output	XML, CSV/XLS, JSON, RDF, Report	In what way is the data formatted for the user?
Language	English, German, More	What is the language of the website? Does it differ from the language of the data?
Target Audience	Business, Customer	Towards whom is the product geared?
Trustworthiness	Low, Medium, High	How trustworthy is the vendor? Can the original data source be tracked or verified?
Size of Vendor	Startup, Medium, Big, Global Player	How big is the vendor?
Maturity	Research Project, Beta, Medium, High	Is the product still in beta or already established?

Table 2.2: Categorization of Data Vendors

The facts about the data vendors were gathered by means of a Web search. As every vendor or marketplace has a website, this publicly available information was used to determine how to categorize each vendor. After having done that with the initial set of vendors, it was checked how many entries a category had to justify its existence. When a cat- egory had only few entries, a new Web search for more data suppliers falling into that category was started in order to make sure no important vendors were omitted. If more companies were found, the list was extended iteratively, and the new companies were analyzed regarding the other dimensions.

Data Access The data access dimension describes through which means end-users receive their data from vendors. The main categories identified and presented in Fig- ure 6 are:

- API: An API (application programming in- terface) is used to provide a language- and platform-independent programmatic access to data over the Internet.
- Download: Traditional download of files is the easiest way to access a data set, because anyone can use such a service with only a Web browser.
- Specialized Software: Some vendors have im- plemented a specialized software client to con- nect with their Web service. While this ap- proach does have downsides (implementation and maintenance expense, dependency issues, etc.), there are some scenarios in which the con- cept is worthwhile, for example, providing the customer with an easy-to-use graphical user in- terface as an out-of-the-box solution that needs no further customization, or granting access to real-time streams of data.
- Web Interface: In a Web interface, the data is displayed to the customer directly on a website.

The flexibility and modularity of APIs have made these the most popular of all access methods. More than 70% of all vendors offer an API. However, less than 30% of all vendors have an API as their only way to access data. Most vendors offer an API next to other methods. For example, Web interfaces or file downloads are used to give previews of the dataset, to make it easier and more accessible for the customer to see what the actual data looks like, e. g., Factual.com has an extensive Web frontend that renders tables or geodata. The concept of specialized software does not seem to stand very well on its own.This approach lacks flexibility, because customers are restricted in the way they can use the data by the functionality of the provided software. Nevertheless, most customers who want data do not want any restrictions on how they can access and process the data. From a theoretical point of view, it seems to be the best approach for a vendor to offer all the aforementioned means of access to his data, because that allows customers to choose their preferred way of access.

Besides aforementioned properties of data format, the main issue for generic frontend is to dynamically retrieve information provided by data sources. It has to be done independently from the data type by using only one channel for all of them. Data will change its value during some period of time and as soon as it happens, user have to receive it automatically. Frontend is responsible for collecting and aggregating data, keeping connection with it, notifying user and visulization on GUI. Doesn't metter how often the data will be changed, but all of it can be transfered as streaming. The term streaming data, usually applied to such type of data as video, audio, or map of values. But usual approaches as HTTP GET/POST become usefull in retrieving static data value. And the same approach for real-time data doesn't work. Based on that, one of the most important requirement to concept, that support any type of data, is to configure interface between backend and frontend, by using streaming channel. It provides possibility for frontend universally transfer any kind of data.

2.3 Summary

This chapter gives an overview of a basic requirements to a generic frontend. Starting from main properties in Section 2.1 namely, loose couling, fine-grained structure, multi-user binding, cross-platforming, responsive design and usability. And continuing in Section 2.2 with definition of data sources, typization and separation to a physical and software sensors. It helps to define main channels of retrived data in which user can be interested in and based on that build a system architecture. An important part of data retrieval, without dependency on data type, is to guarantee real-time streaming. This approach provides an universal way to derive information services and possible solutions for implementation described and explained in detaile in the Chapter 5.

Chapter 3

State of the Art

The following chapter covers an overview and analysis of the existent solutions in the related research areas of web-based third-party applications, which were designed specially for retrieving different type of sensed data to the user. At the beginning of the chapter in the section 3.1 the most familiar research works are studied. It covers public and private projects, that mainly focused on data retrieving through the Web.

In the section 3.2 the prominent approaches of frontends development are studied and evaluated against the requirements described in the section 2.1 with a purpose to clarify their capabilities and properties.

All solutions studied in the following chapter are evaluated against the requirements described in the section 2.1.

3.1 Web of Sensors

Since the thesis is targeted at the creation of a generic user-friendly data stream interaction system and currently every project focused on a specific area of realization and concrete sensor types, such as urban environment[3]. The real-time environmental monitoring portal or geospatial infrastructure for effectively or efficiently collecting and serving vast field data over the web. Internet based urban environment observation system that can real-time monitor environmental changes of temperature, humidity, illumination or air components in urban area. It provides web-based platform, where end-user can simply monitor urban area in his/her city. In environmental monitoring, the field server for constructing outdoor sensor network is a Web-based field observation device, which detects field environmental parameters and publishes them on Internet in real-time.

Smart city[1]tool using information technology and communication (ICT) to help local government to monitoring what currently happened in the city. application for monitoring city in single dashboard to help summarize the current condition of city. The architecture system use network sensor consisting of sensor nodes that has function to capture city condition like temperature, air pollution, water pollution, traffic situation. Also we can add another information socio-economic situation like public health service, economic indicator, energy supplies, etc. We have successfully developed the prototype of the smart city dashboard the give more accurate information of Bandung City, one of big cities in Indonesia. This study has implemented prototype system of smart city dashboard at Bandung. It consists of network sensor, server, application and also communication protocol which is used for city monitoring. The summary information of city can be displayed in single view to help people watching, analyzing and action to what being happen at the real-time in the city.

*Microsoft SensorMap[15]*has proposed a system to monitor and present physical sensors in the real world. SensorMap allows owners of sensor networks to register their physical sensors and publish their data on SensorMap. They use GeoDB to store the sensor network information, DataHub to retrieve the new sensor data to enable real time services, and the aggregator to summarize sensor data in a specific area to clients.

LiveWeb Portal[16] presents the architecture, design, and application of a sensorweb service portal, where sensorweb is a global observation system for varied sensory phenomena from the physical world and the cyber world. This system has been used to represent and monitor real-time physical sensor data and cyber activities from ubiquitous sources. LiveWeb meets its goal of providing an efficient and robust sensor information oriented web service, enabled with real-time data representation, monitoring and notification.LiveWeb has the following properties: the system enables sensorweb service accessible from anywhere, makes sensor network data readable by anyone, sensor network sharing, the system make sensor data format transparent to data users, real-time data display suits sensor data properties, an offline alert system strengthens real-time features.

Internet of Things[2] where presented a service platform based on the Extensible Messaging and Presence Protocol (XMPP) for the development and provision of services for Internet of Things(IoT) mainly focusing on the integration of things based on service technologies, scenarios in domains like smart cities, automotive or crisis management require service platforms involving real world objects, backend-systems and mobile devices. And argued necessary usage of XMPP client as protocol for unified, real-time communication and introduce the major concepts of our platform. Based on two case studies we demonstrate real-time capabilities of XMPP for remote robot control and service development in the e-mobility domain.

Dynvoker Portal[17] a generic human-driven ad-hoc usage approach, by including rapid service testing and dynamic inclusion of services as plugins into applications. Dynvoker consists of a relatively small application core which can be run as a servlet, a web service or a command-line application. explore method-centric and resource-centric services alike, output forms in various formats or integrate GUI services to provide a richer user experience. The generic design of many parts of Dynvoker has yielded a lightweight architecture which is freely available to any interested person as an open source project.

Sensor Web Enablement project[18] is focused on developing standards to enable the discovery of sensors and corresponding observations, exchange, and processing of sensor observations, as well as the tasking of sensors and sensor systems. Open Geospatial Consortium, Inc. members specifies interoperability interfaces and metadata encodings that enable real time integration of heterogeneous sensor webs into the information infrastructure. Developers will use these specifications in creating applications, platforms, and products involving Web-connected devices such as flood gauges, air pollution monitors, stress gauges on bridges, mobile heart monitors, Webcams, and robots as well as space and airborne earth imaging devices. In this publication by OGC was defined such an important XML-based standards as: Sensor Model Language (Sensor ML), Sensor Observation Service (SOS), Web Notification Service (WNS) etc. As subproject calls SANY(Sensors Anywhere) focuses on interoperability of in-situ sensors and sensor networks. The goal for the SANY architecture is to provide a quick and cost-efficient way to reuse data and services from currently incompatible sensors and data sources in future environmental risk management applications. By developing a standard open architecture and a set of basic services for all kinds of sensors, sensor networks, and other sensor-like services, the SANY IP supports and enhances both GMES (Global Monitoring for Environment and Security, a major European space initiative) and GEOSS (Global Earth Observation System of Systems) in the area of in-situ sensor integration. Though the SANY work enhances interoperability for monitoring sensor networks in general, the application focus is on air quality, bathing water quality, and urban tunnel excavation monitoring.

VICCI Project(Visual and Interactive Cyber-physical Systems Control and Integration)[19, 20]. The scope includes smart home environments and supporting people in the ambient assisted living, considers the software-technical side of so-called “Cyber-physical systems” (CPS). This term includes complex, embedded systems, which connect the virtual and the physical world with each other (IoT)in different application scenarios. The main uses of CPS are in logistics, traffic optimization, in the use of robots in the industrial and domestic sectors, in modern energy networks (Smart grid), in the building and factory automation (Smart factory), as well as in the field of intelligent office installations (Smart Office). The aim of project VICCI is the creation of software engineering principles that are necessary for the development of complex cyber-physical systems. Firstly, CPS should be made understandable and accessible by means of a comprehensive control center. Secondly, platforms that enable the development and marketing of software for complex CPS through a pure control panel are to be developed. A domestic environment is considered a sample scenario in which a person with reduced mobility is supported by sensors, actuators and a service robot, which is currently seen as a complex cyber-physical system. No concrete frontend or any kind of user-friendly have been not yet developed.

A series of articles devoted to integrate sensed data into a Cloud. Special attention is given to privacy-relevant or otherwise sensitive information that stores in Cloud. *Sensor-Cloud*[21], a cloud design for user-controlled storage and processing of sensor data proposed security architecture enforces end-to-end data access control by the data owner reaching from the sensor network to the Cloud storage and processing subsystems as well as strict isolation up to the service-level. In this paper authors implement transport security mechanisms for communication with the Cloud, applies object security mechanisms to outbound data items, and performs key management for authorized services. *CloudRemix[22]* a Personal and Federated Cloud Management Cockpit, an interactive cockpit to manage personal clouds and their federations. Is a new techniques for users to perform asset discovery, exchange and management in Cloud area. The CloudRemix prototype demonstrates its utility to manage personal clouds in both social and market-driven environments. The goal of CloudRemix is to be open, user-centric regarding the manageable assets, and flexible regarding their free or commercial exchange, with or without explicit contract negotiation. CloudRemix is an open-source web-based cockpit application with support for multiple users. Each user gets to see an aggregated list of both local and remote services of each of the asset types.

SensorMap[23] is a portal web site for real-time realworld sensor data. SensorMap allows data owners to easily make their data available on the map. The platform also transparently provides mechanisms to archive and index data, to process queries, to aggregate and present results on a geo-centric web interface based on Windows Live Local. In this position paper described the architecture of SensorMap, key challenges in building such a portal, and current status and experience. Is an ongoing project with the goal of creating an online searchable portal of live data from the physical world. Example services provided by such a portal include: a Parking Space Finder service, for directing drivers to available parking spots near their destination; a Bus Alert service, for notifying a user when to head to the bus stop; Waiting Time Monitors, for reporting on the queuing delays at post offices, food courts, etc.; a Lost and Found service, for tracking down lost objects; and a Person Finder service, for locating colleagues or monitoring children playing in the neighborhood. The GUI is based on Windows Live Local, and therefore shares its attractive features such as zooming,panning, street maps, satellite images, etc. In addition, it lets end-users to pose queries on available sensors. SensorMap currently supports three types of queries: i) geographic queries specified by drawing geometric shapes (e.g., a region, a route) directly on the map, ii) type queries specified by sensor types within the viewport, and iii) free text queries specified by keywords describing sensors. It overlays the results returned from the agregator on Windows Live Local. The SensorMap lets users query data sources and view results on the map. The SensorMap describes first constructive idea how to interconnect different types of sensors available in the Web throgh API and provides sytem architecture. But within this project authors don't consider questions as: retrieval of real sensors in a room, sensor control, live data streaming, adaptive GUI and integration with another platform and systems without spesificly added libraries.

Xively Cloud Services[1] is the Public Cloud specifically built for the Internet of Things, by giving developers standards-based services and tools, elastic scalability, and intuitive lifecycle management capabilities. With Xively, user can gain the agility and efficiency needed to rapidly meet market demands for compelling connected products and solutions. Xively's Platform as a Service (PaaS) provides the tools and services needed to create compelling products and solutions on the Internet of Things.

Optique[2] is a scalable semantic access to Big Data for effective data analysis and value creation. Optique will bring about a paradigm shift for data access: by providing a semantic end-to-end connection between users and data sources; enabling users to rapidly formulate intuitive queries using familiar vocabularies and conceptualisations; seamlessly integrating data spread across multiple distributed data sources, including streaming sources; exploiting

[1]Xively ,`https://xively.com/`
[2]Optique ,`http://www.optique-project.eu/`

massive parallelism for scalability far beyond traditional RDBMSs and thus reducing the turnaround time for information requests to minutes rather than days[24, 25, 26]. According to research publications, no reearch was done in a field of visualization and presentation or separation of concerns. So impemented Frontend with adaptive and cross-browser GUI was developed specificly for this project by using up-to-date technologies (jQuery, HTML and CSS).

All research projects are presented in Table 3.1 according to the such characters as: status(active or inactive in nowadays), availability of Frontend as full functional sytem with logic and GUI, Mockup(planed GUI), type of project(public or private).

Target	Status	Frontend	Mockup	Type of Project
Smart City	active	yes	yes	public
Microsoft Sensor Map	inactive	yes	-	private
LiveWeb Portal	inactive	no	no	public
Sensor Web Enablement	inactive	no	no	public
SensorCloud	inactive	no	no	public
CloudRemix	active	yes	-	public
VICCI Project	active	no	yes	public
Dynvoker Portal	inactive	no	no	public
Internet of Things	active	no	no	public
SensorMap	inactive	no	no	public
Xively	active	yes	-	private
Optique	active	yes	-	private

Table 3.1: State of the Art Summary

Concluding all researched projects above, it is become clear that the necessity of a generic Frontend, that can be easily integrated with any type of data-driven platform is high. To do so, needed to be clarify what are the main methodologies to build frontend are existent nowadays, specify types of data that have to retrieved and interface for collaboration between all modules of system architecture.

3.2 Frontend Development Approaches

In computer science, the frontend is responsible for collecting input from user and processing it to a backend system and another direction - collecting data from backend, namely sensor data steam, and processing it to the user-friendly interface. Therefore, on the one side, generic frontend has to satisfy architecture requirements from backend, such as: fine-grained structure, cross-platforming, loose coupling; and on the other side, implement a dynamic user-friendly interface to a end-user with a client-side multi-user data binding. And to satisfy aforementioned requirements it is necessary to compare all available web-based solutions.

To retrieve data from different resources in one web-based interface was specified next existent approaches :

- portal with portlets,
- mashup[3],
- browser based systems
- non-browser based system

3.2.1 Portal

The main concept is to present the user with a single web page that brings together or aggregates content from a number of other systems or servers.

Usually, each information source gets its dedicated area on the page for displaying information (a portlet); often, the user can configure which ones to display. The extent to which content is displayed in a "uniform way" may depend on the intended user and the intended purpose, as well as the diversity of the content. Very often design emphasis is on a certain "metaphor" for configuring and customizing the presentation of the content and the chosen implementation framework and/or code libraries[27, 28]. In portal technologies end-user can customize number of retrieved data sources, but for that he has to be aware what is it and how to integrate it in portal. User interface in portals have fixed layout, style and location on the web page. To make changes in it, end-user needs to have a deep knowledge of the system architecture and of whole portal entirely. The portal allows the administrator to define specific sets of applications, which are presented to the user in a single page context. The Portlets themselves are more than simple views of existing Web content. A Portlet is a complete application having multiple states and view modes, plus event and messaging capabilities.

Portlets run inside the Portlet container of a portal server, similar to a servlet running on an application server. The Portlet container provides a runtime environment in which portlets are instantiated, used, and finally destroyed. Portlets rely on the portal infrastructure to access user profile information, participate in window and action events, communicate with other portlets, access remote content, look up credentials, and store persistent data.

A portal may use a search engine API to permit users to search intranet content as opposed to extranet content by restricting which domains may be searched. Apart from this common search engines feature, web portals may offer other services such as e-mail, news, stock quotes, information from databases and even entertainment content. Portals provide a way for enterprises and organizations to provide a consistent look and feel with access control and procedures for multiple applications and databases, which otherwise would have been different web entities at various URLs. The features available may be restricted by whether access is by an authorized and authenticated user (employee,member) or an anonymous site visitor.

Examples of early public web portals were AOL, Excite, Netvibes, iGoogle, MSN, Naver, Indiatimes, Rediff, Sify and Yahoo![4]. See for example, the "My Yahoo!" feature of Yahoo!

[3] http://www.programmableweb.com/applications
[4] Yahoo!,http://pipes.yahoo.com/pipes/

which may have inspired such features as the later Google "iGoogle" (soon to be discontinued). The configurable side-panels of, for example, the modern Opera browser and the option of "Speed Dial" pages by most browsers continue to reflect the earlier "portal" metaphor.

Main features Integration — Ability to integrate with your current tools or the possibility of adding new tools. You have your outlook calendar and email integrated within intranet.

Security — Enable user or group based security to secure documents and sites throughout the intranet portal.

Customization — Software that is flexible to allow for organization. Web Parts can be used to create custom modules which can make interaction easier with the site. Ability for users to customize tools and resources they use most often.

Collaboration — People are now able to collaborate their work with each other. Example would be multiple people working on one document.

Communication Channels — Allows corporations to promote corporate culture and present information in a more interactive way than before.

User Friendly — Application must be easy to use and understand due to a wide range of technical abilities.

Remote Access — Ability for users to access content away from the office.

Targeted Content — Business portal administrators can target content by business group area, e.g., HR, Marketing, Legal, Corporate Executives, etc.

Portal technology has proven powerful but complex. Mashups offer the other extreme - simplicity, but maybe not as much power.

3.2.2 Mashup

Mashup is a web page, or web application, that uses content from more than one source to create a single new service displayed in a single graphical interface. The term implies easy, fast integration, frequently using open application programming interfaces (API) and data sources to produce enriched results that were not necessarily the original reason for producing the raw source data. The term mashup originally comes from pop music, where people seamlessly combine music from one song with the vocal track from another-thereby mashing them together to create something new. The main characteristics of a mashup are combination, visualization, and aggregation. It is important to make existing data more useful, for personal and professional use. To be able to permanently access the data of other services, mashups are generally client applications or hosted online. Both commercial products and research prototypes have a broad range of features that simplify a mashups design process, and provide mashups storage and publication. But to customize retrived resources end-user have no option, as use only predefined type and numbers of applications, that was created by application or platform developer. The mashup application is a composite Web 2.0 application that aggregates and integrates heterogeneous web resources offered in a form of available Web APIs and sources for creating a new service. Mashups differ from traditional component-based applications in providing more situational character of these applications[29]. In general there distinguish the following three types of mashups[30]. Customer mashups are aimed at the combination and reformation data from various public sources according to users' needs. Data mashups aggregate similar types of resources from diverse sources into a new single data representation. And business, or enterprise, mashups define composite applications that are focused on solving heterogeneous business problems by supporting collaborative activities[31]. The architecture of a mashup is divided into three layers:

- *Presentation / user interaction:* this is the user interface of mashups. The technologies used are HTML/XHTML, CSS, Javascript, Asynchronous Javascript and XML (Ajax).
- *Web Services:* the product's functionality can be accessed using API services. The technologies used are XMLHTTPRequest, XML-RPC, JSON-RPC, SOAP, REST.
- *Data:* handling the data like sending, storing and receiving. The technologies used are XML, JSON, KML.

Concerning architectural styles of mashup applications, server-side and client- side mashups are distinguished. In the server-side mashups a content aggregation is realized on a server[32]. The server plays a role of a proxy between the mashup application and other web sites that involved in this application. The opposite client-side mashups aggregate content on a client, typically, at a client's web browser[33].

Server-side mashup is similar to traditional Web applications using server-side dynamic content generation technologies like Java servlets, CGI, PHP or ASP. The data from multiple sources are aggregated at the server side and the final results are rendered at the client's browser. In the client-side mashup, content mashed can be generated directly within the client's browser using client-side scripts (such as JavaScript or Applets). Mashups following the client-side style are often referred as Rich Internet Applications (RIAs). The benefit of client-side mashup includes the prompt response to user interactions because the data is pre-processed at the client's browser by leveraging Ajax techniques. For example, a page can be updated for portions of its content without having to refresh the entire page. Often a mashup uses a combination of both the server-side and the client-side style to achieve the data aggregation[34].

Portal vs Mashup

Mashups and portals are both content aggregation technologies. Portals are an older technology designed as an extension to traditional dynamic Web applications, in which the process of converting data content into marked-up Web pages is split into two phases: generation of markup "fragments" and aggregation of the fragments into pages. Each markup fragment is generated by a "portlet", and the portal combines them into a single Web page. Portlets may be hosted locally on the portal server or remotely on a separate server.

Portal technology is about server-side, presentation-tier aggregation. It cannot be used to drive more robust forms of application integration such as two-phase commit.

Mashups differ from portals in the following respects:

	Portal	**Mashup**
Classification	Older technology, extension to traditional Web server model using well-defined approach	Using newer, loosely defined "Web 2.0" techniques
Philosophy/approach	Approaches aggregation by splitting role of Web server into two phases: markup generation and aggregation of markup fragments	Uses APIs provided by different content sites to aggregate and reuse the content in another way
Content dependencies	Aggregates presentation-oriented markup fragments (HTML, WML, VoiceXML, etc.)	Can operate on pure XML content and also on presentation-oriented content (e.g., HTML)
Location dependencies	Traditionally, content aggregation takes place on the server	Content aggregation can take place either on the server or on the client
Aggregation style	“Salad bar” style: Aggregated content is presented ‘side-by-side’ without overlaps	“Melting Pot” style – Individual content may be combined in any manner, resulting in arbitrarily structured hybrid content
Event model	Read and update event models are defined through a specific portlet API	CRUD operations are based on REST architectural principles, but no formal API exists
Relevant standards	Portlet behavior is governed by standards JSR 168, JSR 286 and WSRP, although portal page layout and portal functionality are undefined and vendor-specific	Base standards are XML interchanged as REST or Web Services. RSS and Atom are commonly used. More specific mashup standards such as EMML are emerging.

Table 3.2: Portal vs Mashup Technologies

The portal model has been around longer and has had greater investment and product research. Portal technology is therefore more standardized and mature. Over time, increasing maturity and standardization of mashup technology will likely make it more popular than portal technology because it is more closely associated with Web 2.0 and lately Service-oriented Architectures (SOA). New versions of portal products are expected to eventually add mashup support while still supporting legacy portlet applications. Mashup technologies, in contrast, are not expected to provide support for portal standards.

Another possibility is that the overall concept of portals is replaced by new technologies like Ajax widget libraries and even JavaFX. Many of the limitations of JSR-168 portlets seem quaint now that we are in the age of the interactive Web 2.0 or even Web 3.0.

3.2.3 Non-Browser Based Systems

Another synonym of non-browser based sytem is native application, that is targeted to create software specificly for operation system used by device. It is become popular when usual mobile phone started to have parts of computer functionality, e.g. smartphones, tablets. Of course enormous numbers of portable devices increased number of operation systems, requirements and restrictions. But native application always provide best user experience and possibility to use all resources of mobile device. Android and Apple's iOS have, by far, the greatest market share, but there are others, including the Blackberry and Windows Phone operating systems. Developing native apps involves targeting one or more of these platforms, each of which has its own software development kit (SDK)[5]. A native mobile app is a smartphone application that is coded in a specific programming language, such as Objective C for iOS and Java for Android operating systems. Native mobile apps provide fast performance and a high degree of reliability. They also have access to a phone's various devices, such as its camera and address book. In addition, users can use some apps without an Internet connection. However, this type of app is expensive to develop because it is tied to one type of operating system, forcing the company that creates the app to make duplicate versions that work on other platforms.

Rather than being accessed via the Web, native apps are mainly deployed through app marketplaces that are also mostly targeted at particular platforms. These markets allow apps to be downloaded for free or commercially, with the app store taking a percentage cut of sales revenue.

Native apps enjoy a number of natural advantages over Web apps for certain types of tasks. Native user interfaces provide an interaction level and quality that currently cannot be achieved through a Web app running in a browser. In addition, native app processing can employ mobile device hardware features, such as GPS and other localization facilities, accelerometers and touchscreens. As HTML5 develops, Web apps will also be able to exploit some or all of these features. But for now, these bells and whistles are mostly exclusive to native apps.

A native app also has the ability to use offline data storage. Again, the advance of Web technologies, such as HTML5, will begin to close this gap because Web apps will be able to store data for offline use as mobile caching models continue to improve.

The number one disadvantage, or at least consideration, for native apps is the amount of resources businesses require to invest in the development process. Each platform has its own framework, and to target more than one involves multiple programming languages - not to mention an understanding of the different application frameworks. In addition to the initial development project, maintenance of native apps is an ongoing concern, as the platforms they are designed to work with are constantly changing.

[5]Native App,http://www.techopedia.com/2/28134/development/web-development/native-app-or-mobile-web-app

Depending on the app, there also may be significant costs related to distribution and promotion. The official app stores take a cut of each app sale. These have become so overcrowded that for new apps, getting noticed is no mean feat. For commercial projects, apps can be sold for a set, one-off fee or downloaded for free, with many apps using in-app advertising as an alternative source of monetization.

Another potential disadvantage of native apps is the level of control the app stores and platforms exert. This applies to apps and user data. Marketplace policies vary, but for some, particularly Apple's Mac App Store, content is subject to a strict approval system. In some cases, apps have been quickly withdrawn from the store, leaving with little recourse for developers. Since the official stores are the main sources of distribution for the major platforms, your apps are totally dependent on them and at the mercy of their decisions. In addition, there is the possibility that these apps may be removed from user devices, in which case users could permanently lose their data.

also need to consider the kind of data that your application manipulates and where it is stored, and how the users will feel about that. People are obviously okay having their facebook profile data stored on a webserver, but they might feel differently if you're writing a finance application like MYOB and you want to store all their personal financial details on your server. You might be able to get that to work, but it would require a lot of effort to implement the required security and to assure the userbase that their data is safe. In that situation you might decide that the overall effort is lower if you go with a native GUI app.

3.2.4 Browser Based Systems

An application that runs within the Web browser (such as Firefox, Internet Explorer or Chrome, for example). The instructions, typically written in a combination of HTML and JavaScript, are embedded within the Web page that is downloaded from a Web site. The advantage of browser-based applications is that they can run in a Windows PC, Mac or Linux machine, since all Web browsers are required to render HTML and execute JavaScript in the same manner, no matter their environment. In practice, there are minor differences in page rendering, which are generally tolerable. See Web application, JavaScript and HTML.

One of the main benefits of browser-based applications is that there are no downloads necessary in order to make them run. And no need anymore to reinstall and update software manually. This means that, generally, even users behind firewalls can benefit from using these types of tools[6].

Web applications optimized for mobile use also offer significant benefits for certain projects. This is an area that is set to undergo enormous change over the next few years, particularly through technologies such as HTML5 and jQuery Mobile, not to mention improvements in network connectivity. Although it is difficult to accurately predict time frames for these changes, it is clear that, in terms of functionality, they will greatly impact the ability of Web apps[7] to compete with native apps.

[6] Browser based sytem definition,`http://www.pcmag.com/encyclopedia/term/61816/browser-based-application`

[7] WEB APPLICATIONS,`http://www.w3.org/2008/webapps/`

The major advantage of using Web apps to deliver services is the simple fact that only one application needs to be developed. Of course, a successful Web app is tested and refined to cope with browser, operating system and hardware differences, but the bulk of application processing remains accessible from any mobile user environment. Mobile browsers are advancing at a fast pace, and the functionality gap between them and their desktop counterparts is gradually narrowing.

To satisfy one of the main requirement about dynamic user-friendly interface, adaptable to any kind of device, mashup architecture should be enhanced with a HTML5 Technology. Based on various design principles, that truly embody a new vision of possibility and practicality[35].

- Compatibility(inherit all previous techniques and standards)
- Utility
- Secure by Design(origin-based security model that is not only easy to use but is also used consistently by different APIs.)
- Separation of Presentation and Content(CSS3)
- Interoperability(Native browser ability instead of complex JavaScript code; a new, simplified DOCTYPE;simplified character set declaration; powerful yet simple HTML5 APIs)
- Universal Access(support users with disabilities by using screen readers; media independence-HTML5 functionality should work across all different devices and platforms; support for all world languages)

Some organizations have adopted a range of flexible techniques to provide a native app experience while minimizing platform-specific development requirements. Many native apps are essentially Web application interfaces, so some developers are looking to maximize server side processing. However, in such cases, issues caused by network connectivity remain significant hurdles.

3.3 Summary

The chapter has introduced main approaches for building web-based dashboards for different types of sensed data. The section 3.1 provides list of projects devoted to retrieve sensed data from the web and another resources such as: temperaure, humidity, traffic sensors. Was studied not only scientific research projects but also private customer-oriented solutions for Big Data managment. In total 13 projects was structured and characterized according to properties of generic frontend concept.

The section 3.2 consists main technologie that was defined to implement frontend: portal with portles, mashup, native applications and non-browser based sytems. As result, mashup technologies based on browser satisfy all necessary requirements to create generic frontend for exploring sensor and information services. Benefits of browser-based interface:

- Easier to manage: no installation required on user machines, upgrades need only be performed on server side and are immediately available to all users. Data backup can be performed on a single machine as data won't be spread out across multiple clients.
- Application can be accessed from any machine with a browser.
- Can easily support multiple platforms consistently.
- Memory and CPU requirements may be considerably less on the client side as intensive operations can be performed on the server.
- Increased security: data is stored on a single server instead of multiple client machines and access can be better controlled.
- Many other benefits of a centralized environment including logging, data entered from multiple sources can immediately be available from other clients, etc.
- In my experience, it is often easier to debug and faster to develop web-based solutions.
- They require no upgrade procedure since all new features are implemented on the server and automatically delivered to the users;
- Web applications integrate easily into other server-side web procedures, such as email and searching.
- They also provide cross-platform compatibility in most cases (i.e., Windows, Mac, Linux, etc.) because they operate within a web browser window.
- With the advent of HTML5, programmers can create richly interactive environments natively within browsers. Included in the list of new features are native audio, video and animations, as well as improved error handling.
- Modern web applications support greater interactivity and greatly improved usability through technologies such as AJAX that efficiently exchange data between the browser and the server.
- Web applications allow for easier introduction of new user devices (e.g. smartphones, tablets) because they have built-in browsers.
- Benefits of GUI-based interface:
- May be easier to design a more responsive, fluid interface.
- Can take advantage of OS-specific functionality that may not be available via a browser.
- Doesn't necessarily require network access.
- Don't need to worry about browser compatibility issues.
- No single point of failure if server goes down or becomes unavailable.

Based on this deciosion in the chapter 4, defined system architecture and described responsibility of every module.

Chapter 4

Concept

The chapter poses and describes a concept of a user-friendly generic frontend for exploring sensor data, through designing a software architecture and a mockup of a web-based user interface that in the same time controlled and provisioned by end users request. The concept is developed based on the analysis of the current state of the art, up-to-date technologies and usability characteristics. According to defined requirements in chapter 2 to the third-party services and applications and knowledge gained from the studied related works (chapter 3), was proposed concept based on 3-tier architecture and in details described in section 4.2 - 4.4.

4.1 3-tier Architecture

Since the concept of a generic frontend should be distributed as much as possible, it is necessary to determine software architecture to a 3-tier architecture, in which presentation, application processing, and data management functions are logically separated. Figure 4.1 shows this architecture:

- Data Tier: data from different types of sensors
- Application Tier: registry and datahub
- Client Tier: web-based UI

Client Tier hosts the presentation layer components. The main function of the interface is to translate tasks and results to graphical user interface that can be easily understandable and explorable from any kind of device. That satisfy requirements of the usability(section 3.4).
Application Tier includes business logic, logic tier and data access tier. It controls an application's functionality by performing detailed processing, transformation of one type data to another one, defines an interface of interconnection between client tier and data tier.
Data Tier consists source of data that have to be retrieved by application tier to a Client Tier, by request from a end user. This tier keeps data neutral and independent from application

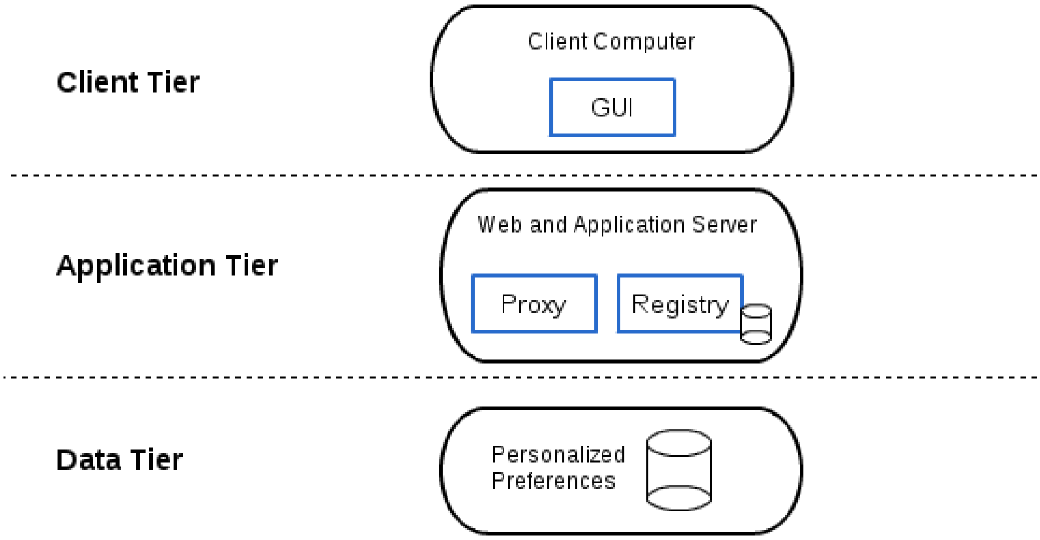

Figure 4.1: 3-tier Architecture

server or business logic. Giving data its own tier also improves scalability.
The three tiers architecture may seem similar to the model-view-controller (MVC) concept. However, topologically they are different. A fundamental rule in a three tier architecture is the client tier never communicates directly with the data tier; in a three-tier model all communication must pass through the middle tier. Conceptually the three-tier architecture is linear. However, the MVC architecture is triangular: the view sends updates to the controller, the controller updates the model, and the view gets updated directly from the model. From a historical perspective the three-tier architecture concept emerged in the 1990s from observations of distributed systems (e.g., web applications) where the client, middleware and data tiers ran on physically separate platforms. Today, MVC and similar model-view-presenter (MVP) are Separation of Concerns design patterns that apply exclusively to the presentation layer of a larger system. In simple scenarios MVC may represent the primary design of a system, reaching directly into the database; however, in most scenarios the Controller and Model in MVC have a loose dependency on either a Service or Data layer/tier. This is all about Client-Server architecture.
In the next section gives an detailed explanation about structure modules in 3-tier architecture.

4.2 Client Tier

The client tier or another name is presentation tier is a layer which users can access directly such as a web page by using browser. It is the first that user see. This tier consists logic and GUI by using which, user can communicate with sensor in a way that is presented on Figure 4.2. Also Client Tier responsible to be adapted to any kind of mobile or desktop devices that user can use. Therefore as a cross-platforming approach was chosen web-based solution, where all communication flows through the browser.

Figure 4.2 presents simple content layout in which user can be interested in. It consists:

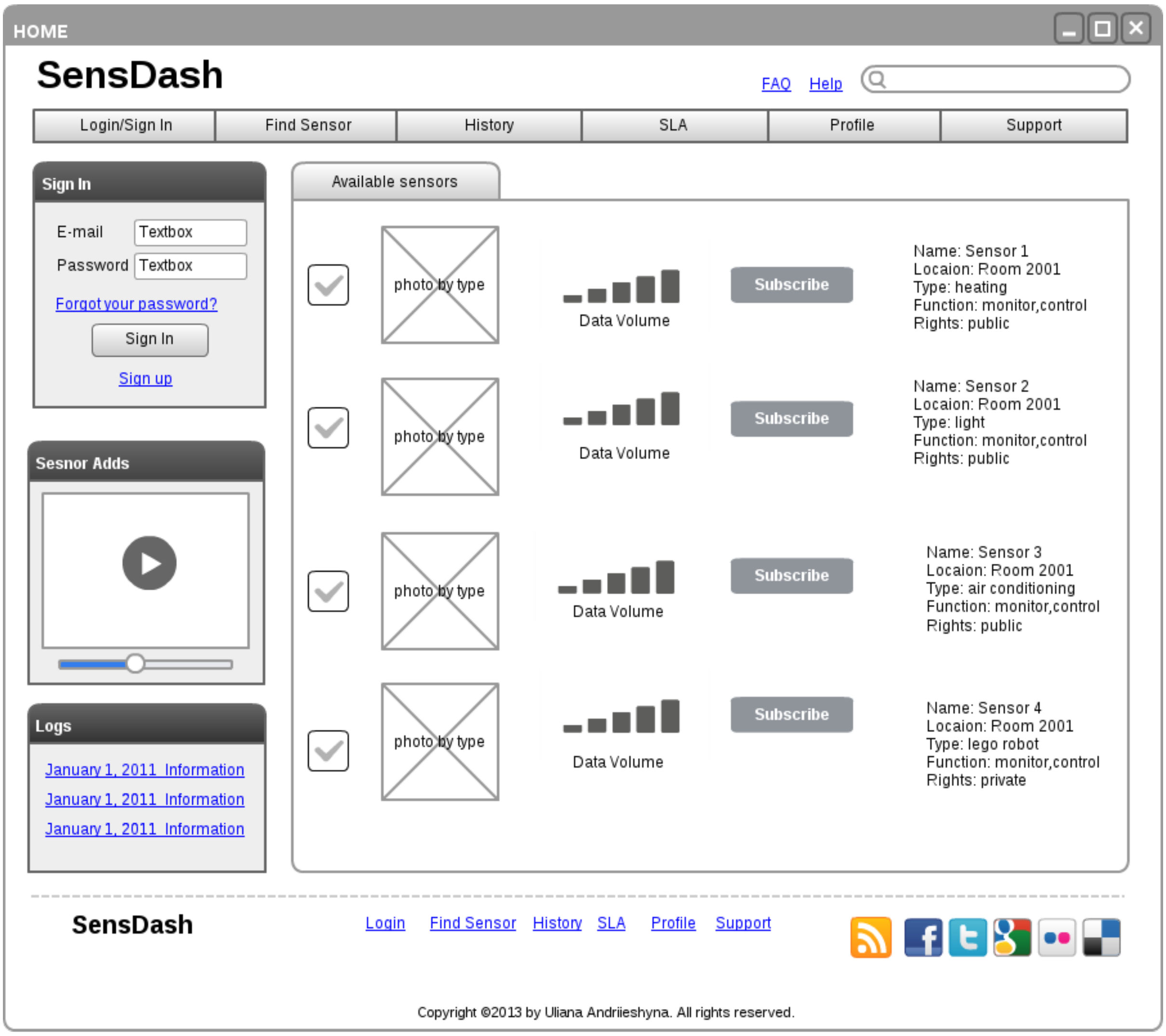

Figure 4.2: GUI Mockup

- Login form with user name and password fields. After user logged in, the system defines his/her rights and applies visibility rules according to credentials. User that have an admin rights receives an opportunity to control and manipulate sensors. Simple user will receive an opportunity to get a statistic and information from sensors.

- Sensor icon defines what is the current type of sensor, e.g. light, temperature, heating, robot lego, etc. In such a way user can easily, even in seconds, understand and catch what is the main function of a sensor in the list.

- Availability or unavailability to see alive statistics. User can subscribe only to the available services. If some services become unavailable it will be automatically marked as inactive and after refreshing will be deleted from the list of available sensors and moved to a history tab.

- Data Volume icon shows what is the average data stream volume needed to retrieve sensor data(Kb/s). The dashboard should automatically adapt quality of streaming data to necessary and possible bandwidth of available connection(Wi-Fi, 3G, GPRS)

The common use case from a end-user point of view shown on Figure 4.3. User can use any type of mobile device and his favorite browser to receive information from the sensors by using web-site as a dashboard.

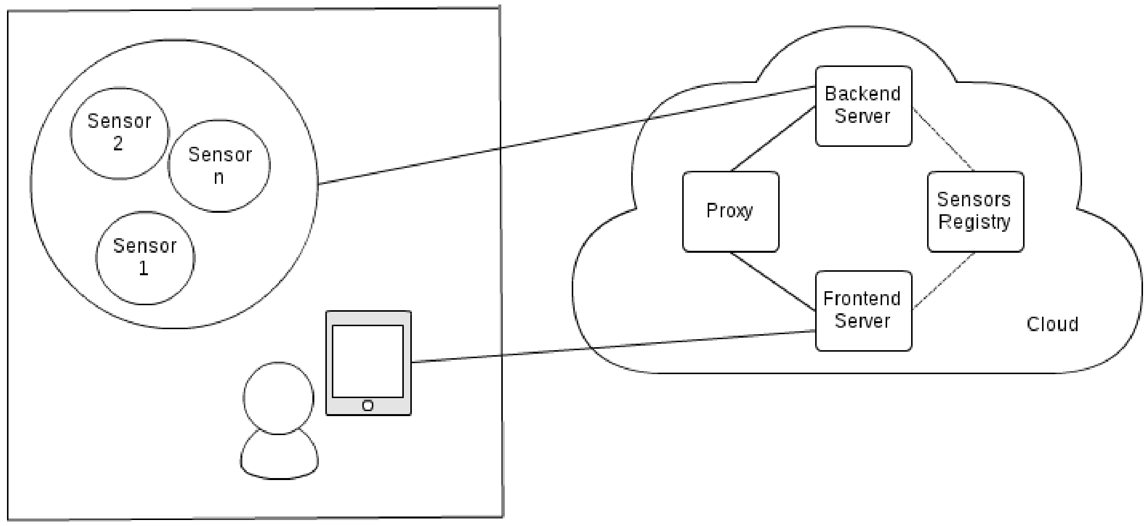

Figure 4.3: Use Case

4.3 Application Tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.
Application tier consists all logical modules as: Web server, Sensor registry, DataHub and Web-based Frontend. All these modules connects to each other as shown on a Figure 4.4. Detailed information about functionality of every module described in sections above.

4.3.1 Web server

The primary function of a web server is to deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to text content.
In proposed concept Web server responsible for dynamic distribution of tasks between various modules of common system to serve efficiently static files. Such specific operation logic like authentication of user, registration of sensors and users are delegated to an external components such as Registry, DataHub and Frontend. these external components can be interchange without dependency to the system itself.

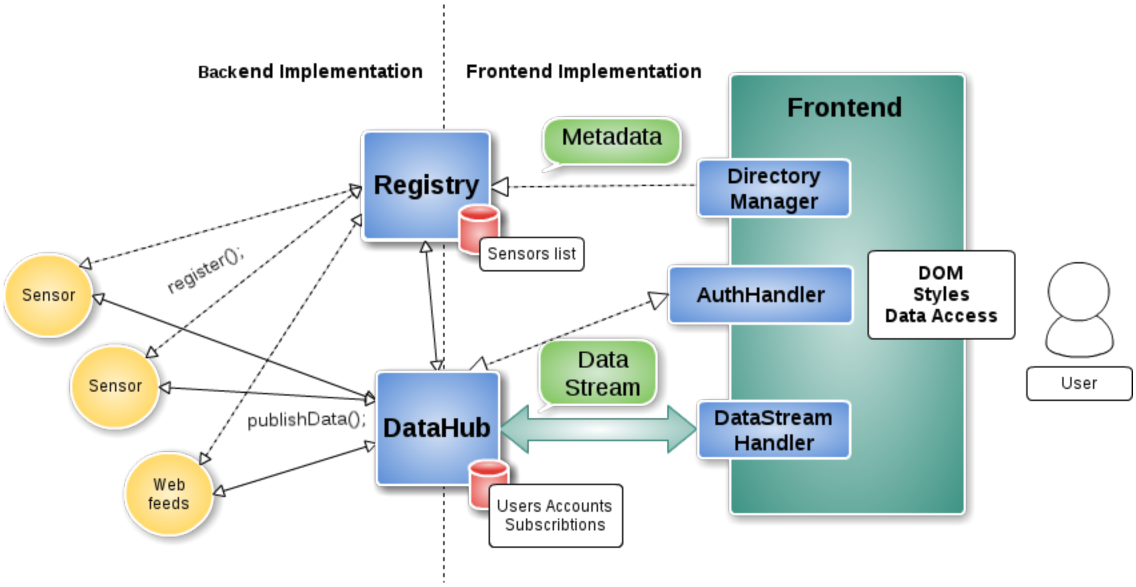

Figure 4.4: System Architecture

4.3.2 Sensor Registry

Sensor Registry is a module responsible for keeping all required metadata of available sensors, for connecting arbitrary sensors. Frontend gets info from Registry about registered sensors and their availability. By using simple RESTful API, so that any Registry that implements suggested API and returns valid JSON, which consits such info as: id of sensor, availability(true or false), SLA, necessary bandwidth for retrieving data, title and type. By using RESTful API, the concept provides a possibility to dynamically connect and disconnect different Registries, that are required in a different context of usage. Before publishing data to SensorMap, a data publisher must first register the sensor by providing its static description. This meta data describes sensor name, sensing type, loca- tion, data type, units, URL, as well as a free text de- scription of the sensors, and is used in searching sen- sors for a given user query. In SensorMap, we use a Sensor Description Markup Language (SDML) to encode these properties. Unlike SensorML [2], SDML only describes sensor data interfaces rather than the internal structure of sensors. Thus, it is much simpler and lighter-weight than SensorML. However, SDML syntax is similar to SensorML's; we can incorporate SensorML's features into SDML as we need and Sen- sorMap can incorporate SensorML when it matures.

4.3.3 DataHub

Since concept of generic frontend is meant to support any type of sensor, regardless of its interface, DataHub is responsible for mapping interface of particular sensors data stream into JSOn message, delivered through XMPP protocol. Requirements to DatahUb are:

- convert sensor data into particular JSON supporting scheme(specification*)
- implement XMPP protocol to provide exchange message with XMPP server

- get and parse sensor data

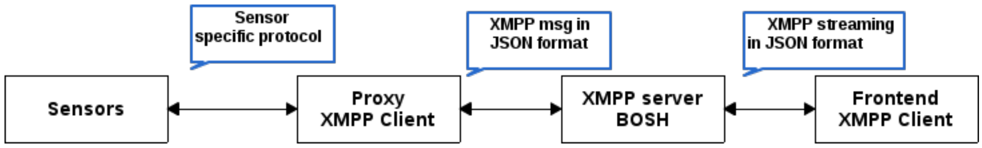

Figure 4.5: Protocol flow

4.3.4 Web-based Frontend

Interfaces

According to RESTful API and simple structure of registry data, communication between Frontend and Registry will done by using HTTP get request and JSON format.
According to requirements to retrieve streaming data from the sensors, it is necessary to use XMPP protocol[36]. XMPP powers a wide range of applications including instant messaging, multi-user chat, voice and video conferencing, collaborative spaces, real-time gaming, data synchronization, and even search. Although XMPP started its life as an open, standardized alternative to proprietary instant messaging systems like ICQ and AOL Instant Messenger, it has matured into an extremely robust protocol for all kinds of exciting creations.
The core of XMPP is the exchange of small, structured chunks of information. Like HTTP, XMPP is a client-server protocol, but it differs from HTTP by allowing either side to send data to the other asynchronously. XMPP connections are long lived, and data is pushed instead of pulled. Because of XMPP's differences, it provides an excellent companion protocol to HTTP. XMPP-powered web applications are to AJAX what AJAX was to the static web site; they are the next level of interactivity and dynamism. Where JavaScript and dynamic HTML have brought desktop application features to the web browser, XMPP brings new communications possibilities to the Web. XMPP has many common social web features built in, due to its instant messaging heritage. Contact lists and subscriptions create social graphs, presence updates help users keep track of who is doing what, and private messaging makes communication among users trivial. XMPP also has nearly 300 extensions, providing a broad and useful range of tools on which to build sophisticated applications.
XMPP, like all protocols, defines a format for moving data between two or more communicating entities. In XMPP's case, the entities are normally a client and a server, although it also allows for peer-to-peer communication between two servers or two clients. Many XMPP servers exist on the Internet, accessible to all, and form a federated network of interconnected systems. Data exchanged over XMPP is in XML, giving the communication a rich, extensible structure. Many modern protocols forgo the bandwidth savings of a binary encoding for the more practical feature of being human readable and therefore easily debugged. XMPP's choice to piggyback on XML means that it can take advantage of the large amount of knowledge and supporting software for dealing with XML. One major feature XMPP gets by using XML is XML's insensibility. It is extremely easy to add new features to the protocol that are both backward and forward compatible. This extensibility is put to great use in

the more than 200 protocol extensions registered with the XMPP Standards Foundation and has provided developers with a rich and practically unlimited set of tools. XML is known primarily as a document format, but in XMPP, XML data is organized as a pair of streams, one stream for each direction of communication. Each XML stream consists of an opening element, followed by XMPP stanzas and other top-level elements, and then a closing element. Each XMPP stanza is a first-level child element of the stream with all its descendant elements and attributes. At the end of an XMPP connection, the two streams form a pair of valid XML documents. The Extensible Messaging and Presence Protocol (XMPP) is the IETF's formalization of the base XML streaming protocols for instant messaging and presence developed within the Jabber community starting in 1999. This page provides a brief chronology of Jabber/XMPP technologies from the perspective of standardization[37].

- *Decentralization*
  The architecture of the XMPP network is similar to email; anyone can run their own XMPP server and there is no central master server.

- *Open standards*
  The Internet Engineering Task Force has formalized XMPP as an approved instant messaging and presence technology under the name of XMPP (the latest specifications are RFC 6120 and RFC 6121). No royalties are required to implement support of these specifications and their development is not tied to a single vendor.

- *History*
  XMPP technologies have been in use since 1999. Multiple implementations of the XMPP standards exist for clients, servers, components, and code libraries.

- *Security*
  XMPP servers can be isolated from the public XMPP network (e.g., on a company intranet), and strong security (via SASL and TLS) has been built into the core XMPP specifications.

- *Flexibility*
  Custom functionality can be built on top of XMPP; to maintain interoperability, common extensions are managed by the XMPP Standards Foundation. XMPP applications beyond IM include groupchat, network management, content syndication, collaboration tools, file sharing, gaming, remote systems control and monitoring, geolocation, middleware and cloud computing, VoIP and Identity services.

The XMPP network uses a client–server architecture (clients do not talk directly to one another). However, it is decentralized—by design, there is no central authoritative server, as there is with services such as AOL Instant Messenger or Windows Live Messenger. Some confusion often arises on this point as there is a public XMPP server being run at jabber.org, to which a large number of users subscribe. However, anyone may run their own XMPP server on their own domain. Every user on the network has a unique Jabber ID (usually abbreviated as JID). To avoid requiring a central server to maintain a list of IDs, the JID is structured like an email address with a username and a domain name (or IP address[16]) for the server where that user resides, separated by an at sign (@), such as username@example.com.

Authentication Stub

Frontend MVC

In the design shown in Figure 1 on page 8, Model represents the application object that implements the application data and business logic. The View is responsible for formatting the application results and dynamic page construction. The Controller is responsible for receiving the client request, invoking the appropriate business logic, and based on the results, selecting the appropriate view to be presented to the user. The Model represents enterprise data and the business rules that govern access to and updates to this data. Often the Model serves as a software approximation to a real-world process, so simple real-world modeling techniques apply when defining the Model. A View renders the contents of a Model. It accesses enterprise data through the Model and specifies how that data should be presented. It is the View's responsibility to maintain consistency in its presentation when the Model changes. This can be achieved by using a push Model, where the View registers itself with the Model for change notifications, or a pull Model, where the View is responsible for calling the Model when it needs to retrieve the most current data. A Controller translates interactions with the View into actions to be performed by the Model. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in a Web application, they appear as GET and POST HTTP requests. The actions performed by the Model include activating business processes or changing the state of the Model. Based on the user interactions and the outcome of the Model actions, the Controller responds by selecting an appropriate View.

4.4 Data Tier

As was mentioned in section 4.1 Data Tier consists source of data that have to be retrieved by application tier to a client tier. Where is source of data is a data provided by sensors, in a specific type by using specific protocol, that will be handled by Application Tier. The main focus of this section determine possible characteristics of streamed data that can be retrieved by Client Tier in a lightweight scenario for mobile devices.
Streaming media is multimedia that is constantly received by and presented to an end-user while being delivered by a provider. Its verb form, "to stream", refers to the process of delivering media in this manner; the term refers to the delivery method of the medium rather than the medium itself. In general, media files can be delivered in one of three ways, via streaming, progressive download, or adaptive bitrate streaming. Each has its purpose.
Streaming involves delivering the media to the client via a server process using specific

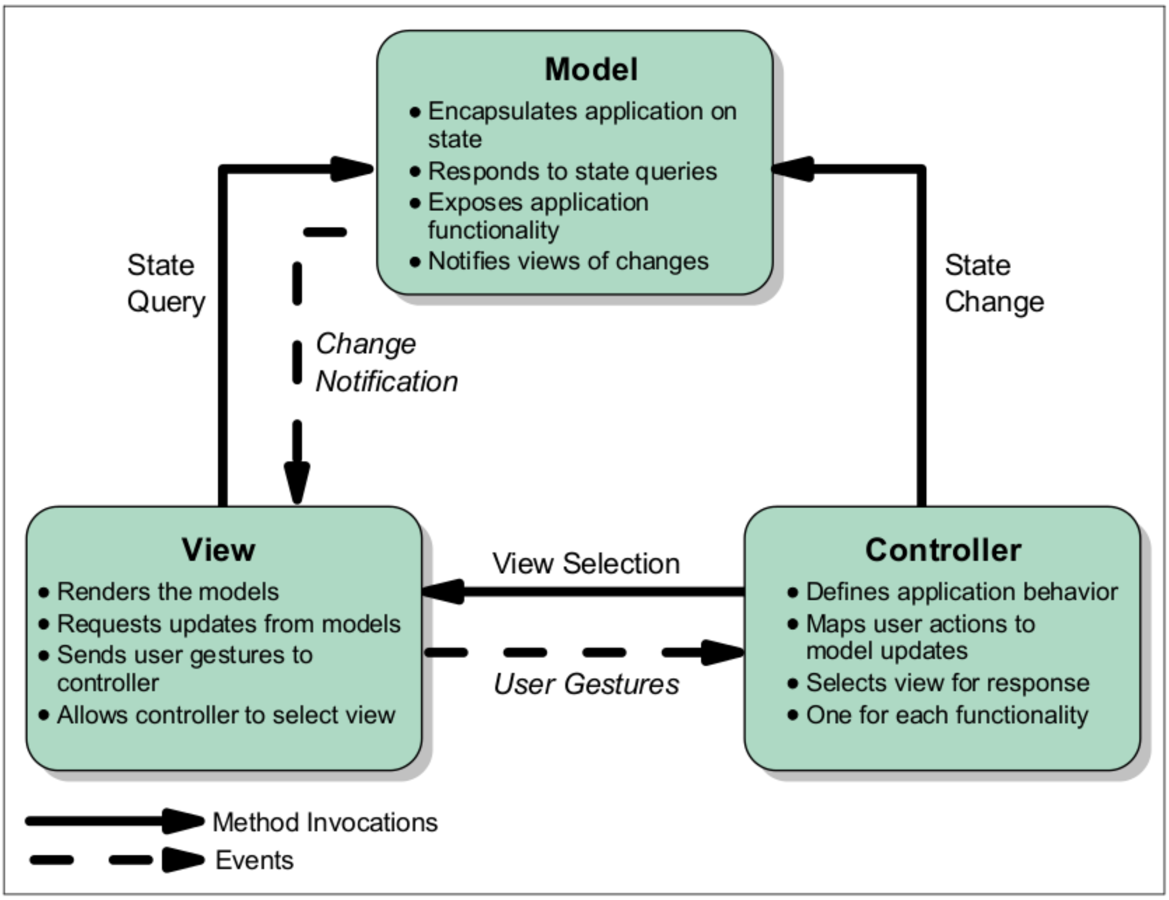

Figure 1 The Mode-View-Controller design pattern

Figure 4.6: MVC Pattern

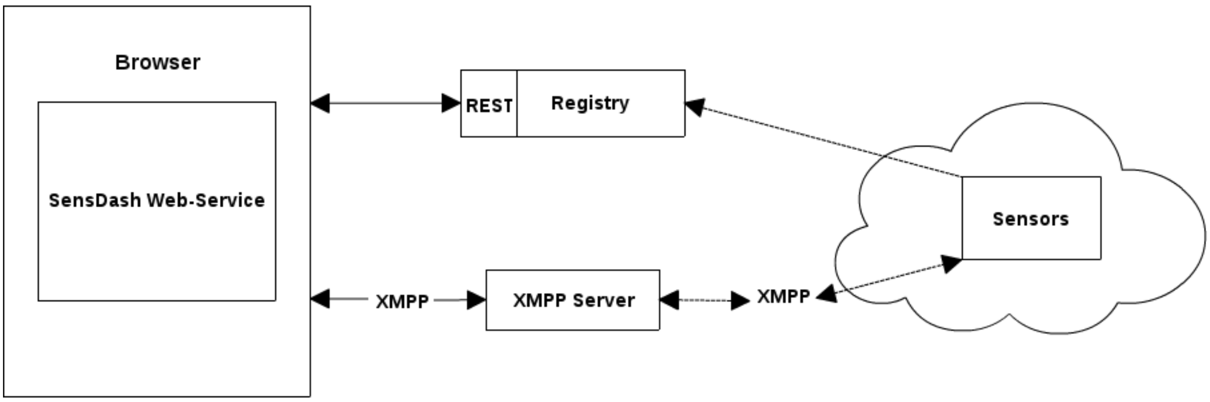

Figure 4.7: Interface

streaming protocols (such as XMPP). Video playing begins almost immediately, especially if the video file was encoded at a data rate similar to the effective bandwidth of the target viewer. Streaming video is also often not cached by the client so a local copy of the video is not held in its entirety on the client machine. While it is not impossible for an enterprising person to capture and hold a copy of the stream, it takes more effort than the casual viewer may be willing to take on. To adapt for the slowest common denominator in regard to end-user bandwidth, streaming videos are often encoded at lower quality and data rates.

Progressive download simply delivers a media file via traditional web server technologies. The file begins playing on the client as soon as enough data has been buffered to provide a smooth uninterrupted viewing experience. Progressive downloaded files are easier to capture since an entire copy of the file is downloaded to the local device. Also, the quality of the file can be higher simply because a user on a slower connection will just have to wait longer for the viewing to begin.
Adaptive bitrate streaming is a kind of best of both worlds. As the name implies, adaptive bitrate is a streaming technology and generally requires a dedicated streaming server. In this case, media files are transcoded into multiple bitrates with the appropriate streaming being delivered to the user based on their available bandwidth. Adaptive streaming servers can also dynamically change the bitrate as network conditions dictate[38].
This explanation shows that adaptive bitrate streaming is the most valuable and suitable for concept of a generic frontend. But it is necessary to go deeply in details to define limits and understanding of "good quality", "bad quality", "excellent quality". All three delivery methods are forms of Adaptive Bit Rate Streaming. This delivery method will have a massive impact on every aspect of Internet video delivery because it allows the stream to actually adapt the video experience to the quality of the network and the device's CPU.
In other words, the video stream can increase or decrease the bit rate and resolution of the video (its quality) in real time so that it's always streaming the best possible quality the available network connection can support. The better the network connection, the better the video image quality. The fact that the stream handles all of this complexity means the mobile video viewer doesn't have to do anything; everything is left to the stream and the player.
So how does this all work? To prep your video content for HLS, you start off with a high quality version of your video and encode multiple copies of it using MPEG-4 H.264. These copies are at various bit rates and resolutions ranging from lower quality renditions appropriate for slower 3G connections, up to extremely high quality renditions suitable for fast devices on fast networks. The renditions are then wrapped into MPEG-2 Transport Streams and chopped up into 10 second segments or chunks. It's these segments that are eventually streamed to an HTML5 Video Player on a mobile device, browser or set-top box, and because the player receives the video in 10 second chunks and can detect the quality of the network connection, it can switch to a higher or lower quality video segment every ten seconds if bandwidth conditions change.
Mobile platform such as iOS/Mobile Encoding supports at least two video types: 3GP + MPEG-4 for less sophisticated devices, and H.264 + MP4 for smartphones. One output video can cover all of smartphone users – iPhone/iPad/iPod, Android, and (for the most part) Blackberry too. Toss in PSP, PS3, and Xbox 360 for good measure. Mobile devices well using a handful of standard encoding profiles. Start with the Universal Smartphone Profile for wide compatibility; add in an Advanced Smartphone Profile version for the more advanced devices; and round out mobile list with a legacy profile for widest compatibility – either our Legacy Smartphone Profile (below), or even a 3GP video for even wider compatibility. The following defaults are the starting point for these profiles. Default these settings by default, but you can replicate them easily enough in whatever encoding tool you're using. Defaults: Video: H.264, Level 3.0, Baseline profile Audio: AAC, 1-2 channels

4.5 Summary

In this chapter, a first web-based concept for sensor streaming services is to be created. Along with it, a light-weight scenario service registry will be needed. Users should be able to explore not just services, but also the information provided by them, and eventually be led to advanced usage patterns such as the development of third-party applications to access the information data and real-time streams.
DataHub responsibilities:

- 1
- 2
- 3

Registry responsibilities:

- 1
- 2
- 3

Web-server responsibilities:

- 1
- 2
- 3

Frontend responsibilities:

- 1
- 2
- 3

Chapter 5

Implementation and Evaluation

The chapter contains practical part of the work, describing implementation of suggested prototype in the section 2.3. The prototype implements the major aspects proposed in the concept (chapter 4). The implementation consists the major aspects proposed in the concept, according to 3-tier architecture. Namely next components:

- **Client Tier** presents adaptive to different screens GUI, dynamically changed content and multi-user access
- **Application Tier** consists Apache Web-server, XMPP server and guarantee appropriate interface of collaboration between tiers via JSON format and defined structure
- **Data Tier** consists cookies for authorization

To make evaluations real, system will use data from the VICCI(Visual and Interactive Cyber-physical Systems Control and Integration) project at the Faculty of Computer Science of the Dresden University of Technology. The scope includes smart home environments and supporting people in the ambient assisted living. Also it was connected by using Data Hub as a Proxy, that is the part of Master Thesis of Luiz Alberto Borges, "Data Hub for Adaptive Data Services".

5.1 Development Environment

As a programming languages for implementation of this work jQuery[1] and HTML5[2] together with CSS3[3] are chosen. jQuery is a fast, small, well documented, easy and widely used and feature-rich JavaScript library. In addition it has such an important properties as: chaining, easy-to-use AJAX, event handlers, CSS selectors, pluins. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. It enables the

[1] jQuery programming language, `http://jquery.com/`
[2] HTML specification, `http://www.w3.org/wiki/HTML/Specifications`
[3] CSS specification, `http://www.w3.org/Style/CSS/specs.en.html`

project code to be portable over different platforms and provides opportunity for robust and effective development. The choice is dictated mostly by two aspects. On the one hand, the system that is being developed is distributed by it's nature. On the other hand, jQuery has low entry barrier, and the code written in this language is extremely readable, laconic and understandable. These facts make further support of written code much easier for other developers that have an experience with any other JavaScript library.
To show justification of choosen approach was made comprehensive comparison between main web toolkits/libraries as Dojo[4], Prototype[5], Yahoo User Interface(YUI) and ExtJS[6] that shown in the Table 5.1.

Target	jQuery	Dojo	Prototype	YUI	ExtJS
License	MIT	BSD & AFL	MIT	BSD	GPL and Commercial
Size	32 KiB	41 kB	46–278 kB	31 kB	84–502 kB
Source language	JavaScript	JavaScript + HTML	JavaScript	Javascript + HTML + CSS	JavaScript
Grid	yes	yes	yes	-	yes
DOM wrapped	yes	yes	yes	no	yes
Other data retrieval	XML, HTML	XML, HTML, CSV, ATOM	-	yes	XML
DOM wrapped	yes	yes	yes	no	yes
Server push data retrieval	yes	yes	-	via Plugin	yes
GUI page layout	with Plugin	yes	yes	-	yes
Touch events	with Plugin	yes	yes	-	yes

Table 5.1: Comparison of JavaScript frameworks

Also the most important part is a version of browser support. jQuery[7], Dojo[8], Prototype[9] , YUI[10], ExtJS[11]

[4]Dojo documentation, `http://dojotoolkit.org/features/`
[5]Prototype documentation, `http://prototypejs.org/`
[6]ExtJS documentation,`http://docs.sencha.com/extjs/4.2.2/`
[7]jQuery browser support, `http://jquery.com/browser-support/`
[8]Dojo browser support,`http://livedocs.dojotoolkit.org/releasenotes/1.4`
[9]Prototype browser support, `http://prototypejs.org/doc/latest/Prototype/Browser/index.html`
[10]YUI browser support, `http://yuilibrary.com/yui/environments/`
[11]ExtJS browser support, `http://www.sencha.com/products/extjs/`

Target	jQuery	Dojo	Prototype	YUI	ExtJS
Chrome	1+	3	1+	-	10+
Opera	9+	10.50+	9.25+	10.0+	11+
Safari	3+	4	2.0.4+	4.0	4+
Mozilla Firefox	2+	3+	1.5+	3+	3.6+
Internet Explorer	6+	6+	6+	6+	6+

Table 5.2: Browser Support

5.2 Web-based Framework Analysis

- **Bootstrap**
  Bootstrap is the most popular and widely used framework, nowadays. It's a beautiful, intuitive and powerful web design kit for creating cross browser, consistent and good looking interfaces. It offers many of the popular UI components with a plain-yet-elegant style, a grid system and JavaScript plugins for common scenarios.

  It consists of four main parts: Scaffolding – global styles, responsive 12-column grids and layouts. Bear in mind that Bootstrap doesn't include responsive features by default. If design needs to be responsive this functionality have to be done manually. Base CSS – this includes fundamental HTML elements like tables, forms, buttons, and images, styled and enhanced with extensible classes. Components – collection of reusable components like dropdowns, button groups, navigation controls (tabs, pills, lists, breadcrumbs, pagination), thumbnails, progress bars, media objects, and more. JavaScript – jQuery plugins which bring the above components to life, plus transitions, modals, tool tips, popovers, scrollspy (for automatically updating nav targets based on scroll position), carousel, typeahead (a fast and fully-featured autocomplete library), affix navigation, and more.

- **Foundation**
  Foundation is a powerful, feature-rich, responsive front-end framework. With Foundation user can quickly prototype and build websites or apps that work on any kind of device, with tons of included layout constructs, elements and best practices. It's built with mobile first in mind, utilitizes semantic features, and uses Zepto instead of jQuery in order to brings better user experience and faster performance.
  Foundation has a 12-column flexible, nestable grid powerful enough to create rapidly multi-device layouts. In terms of features it provides many. There are styles for typography, buttons, forms, and various navigation controls. Many useful CSS components are provided like panels, pricing tables, progress bars, tables, thumbnails, and flex video that can scale properly your video on any device. And, of course, JavaScript plugins including dropdowns, joyride (a simple and easy website tour), Magellan (a sticky navigation that indicates where is the user on the page), orbit (a responsive image slider with touch support), reveal (for creating modal dialogues or pop-up windows), sections (a powerful replacement for traditional accordions and tabs), and tooltips.

- **GroundworkCSS**

  GroundworkCSS is a new, fresh addition to the front-end frameworks family. It's a fully responsive HTML5, CSS and JavaScript toolkit built with the power of Sass and Compass which gives the ability to rapidly prototype and build websites and apps that work on virtually any device.

  It offers an extremely flexible, nestable, fraction-based, fluid grid system that makes creating any layout possible. GroundworkCSS has some really expressive features like tablets and mobile grids which maintain the grid column structure instead of collapsing the grid columns into individual rows when the viewport is below 768 or 480 pixels wide. Another cool feature is a jQuery ResponsiveText plugin which allows to have dynamically sized text that adapts to the width of the viewport: extremely useful for scalable headlines and building responsive tables. The framework includes a rich set of UI components like tabs, responsive data tables, buttons, forms, responsive navigation controls, tiles (a beautiful alternative to radio buttons and other boring standard form elements), tooltips, modals, Cycle2(a powerful, responsive content slider), and many more useful elements and helpers. It also offers a nice set of vector social icons and a full suite of pictographic icons included in FontAwesome. To see the framework in action user can use the resizer at the top center of the browser window. This way user can test the responsiveness of the components against different sizes and viewports while exploring the framework's features. GroundworkCSS is very well documented with many examples, and to get user started quickly the framework also provides several responsive templates. The only thing as a weakness is the missing of a way to customize download.

- **Gumby**

  Gumby is simple, flexible, and robust front-end framework built with Sass and Compass.

  Its fluid-fixed layout self-optimizes the content for desktop and mobile resolutions. It support multiple types of grids, including nested ones, with different column variations. Gumby has two PSD templates that get user started designing on 12 and 16 column grid systems. The framework offers feature-rich UI Kit which includes buttons, forms, mobile navigation, tabs, skip links, toggles and switches, drawers, responsive images, retina images, and more. Following the latest design trends the UI elements have Metro style flat design but can use Pretty style with gradient design too, or to mix up both styles. An awesome set of responsive, resolution independent Entypo icons, is completely integrated into the Gumby Framework. Gumby has also a very good customizer with color pickers which helps to build your custom download with ease.

- **Kube**

  Lastly, if user need a solid, yet simple base without needless complexity and extras, for your new project, Kube can be the right choice. Kube is a minimal, responsive and adaptive framework with no imposed styling which gives to user the freedom to create. It offers basic styles for grids, forms, typography, tables, buttons, navigation, and other stuff like links or images.

  The framework contains one compact CSS file for building responsive layouts with ease and two JS files for implementing tabs and buttons in your designs. If user is looking for maximum flexibility and customization, user can download developer version which includes LESS files, with variables, mixins and modules.

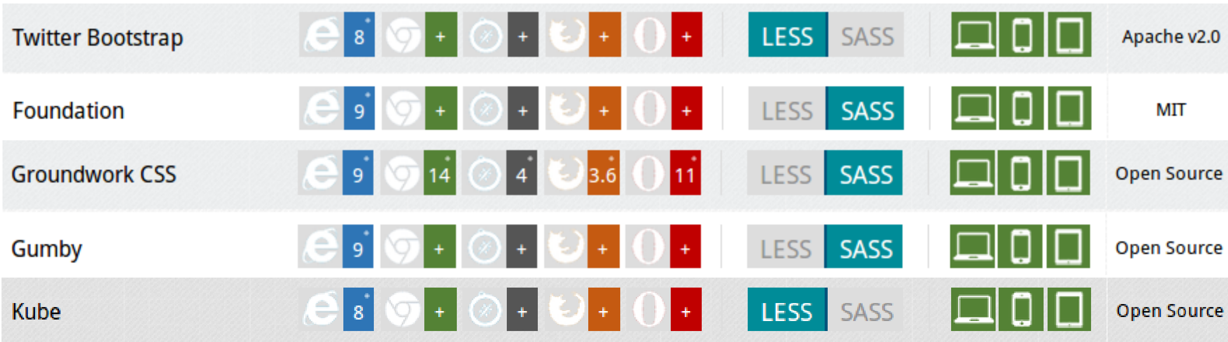

Figure 5.1: Framework Comparison[12]

5.3 Data Flow Model

Data Flow model describes the

5.3.1 XMPP implementation

XMPP Stanzas

Work is accomplished in XMPP by the sending and receiving of XMPP stanzas over an XMPP stream. Three basic stanzas make up the core XMPP toolset. These stanzas are <presence> , <message> , and <iq> . Each type of stanza has its place and purpose, and by composing the right kinds of quantities of these stanzas, sophisticated behaviors can be achieved. Remember that an XMPP stream is a set of two XML documents, one for each direction of communication. These documents have a root <stream:stream> element. The children of this <stream:stream> element consist of routable stanzas and stream related top-level children. Each stanza is an XML element, including its children. The end points of XMPP communication process input and generate output on a stanza-by-stanza basis.The following example shows a simplified and short XMPP session:: In this example, Elizabeth created an XMPP stream by sending the opening <stream:stream> tag. With the stream open, she sent her first stanza, an <iq> element. This <iq> element requested Elizabeth's roster, the list of all her stored contacts. Next, she notified the server that she was online and available with a <presence> stanza. After noticing that Mr. Darcy was online, she sent him a short <message> stanza, thwarting his attempt at small talk. Finally, Elizabeth sent another <presence> stanza to inform the server she was unavailable and closed the <stream:stream> element, ending the session.

```
<stream:stream>
  <iq type='get'>
    <query xmlns='jabber:iq:roster'/>
  </iq>

  <presence/>

  <message to='darcy@pemberley.lit'
           from='elizabaeth@longbourn.lit/ballroom'
           type='chat'>
    <body>I cannot talk of books in a ball-room; my head is always full of
      something else.</body>
  </message>

  <presence type='unavailable'/>
</stream:stream>
```

Figure 5.2: Stanzas example

Server

The set of XMPP servers that can mutually communicate forms an XMPP network. The set of public XMPP servers forms the global, federated XMPP network. If a server does not speak the server-to-server protocol, it becomes an island, unable to communicate with external servers. An XMPP server will usually allow users to connect to it. It is, however, also possible to write applications or services that speak the server-to-server protocol directly in order to improve efficiency by eliminating routing overhead. Anyone can run an XMPP server, and full-featured servers are available for nearly every platform. Ejabberd, Openfire, and Tigase are three popular open source choices that will work on Windows, Mac OS X, or Linux systems. Several commercial XMPP servers are available as well, including M-Link and Jabber XCP.

Connection

Before any stanzas are sent, an XMPP stream is necessary. Before an XMPP stream can exist, a connection must be made to an XMPP server. XMPP includes some sophisticated support for establishing connections to the right servers. Typically clients and servers utilize the domain name system (DNS) to resolve a server's domain name into an address they can connect to. Email services in particular use mail exchange (MX) records to provide a list of servers that handle mail for a given domain so that one well-known server address does not have to handle every service. Email, being an early Internet application, got special treatment in DNS. These days, service records (SRV) are used to provide a similar function for arbitrary services. The first thing an XMPP client or server does when connecting to another XMPP server is to query the appropriate SRV record at the server's domain. The response may include multiple SRV records, which can be used to load balance connections across multiple servers. If an appropriate SRV record cannot be found, the application tries to connect to the given domain directly as a fallback. Most libraries also allow you to specify a server to connect explicitly.

Long Polling

Even with AJAX, data was still being requested, or polled, at timed intervals. Servers can be crippled if too many clients poll too fast.However, to get quick updates, the polling interval needs to be quite small; the lowest latency possible is the length of the polling interval. Another issue with polling is that most poll requests do not receive new data. In order to see changes within a reasonable time frame of when they occur, the polling interval must be quite short, but the actual data may not change very often. For example, if there is new data ready on the server, the server answers immediately. If there is not new data, the server keeps the connection open, holding any reply. Once new data arrives, it finally responds to the request. If no new data arrives after some period of time, the server can send back an empty reply, so as not to hold too many open connections at once. Once a request is returned, the client immediately sends a new one, and the whole process starts over. Because each polling request is potentially open for a long period of time, this technique is called long polling. It has many advantages over normal polling.

5.4 Browser Support

In past years a Flash-based media player in more than sufficient for streaming on the Web and this technology is still necessary to support legacy browsers. But thankfully modern standards have advanced and the inclusion of HTML5 video opens doors for dozens of new opportunities.

In this guide I'd like to offer an introduction to HTML5 video for the Web. It will take some practice to understand the native in-browser player and all its functionality. When you're working with a flash video player it's all too common to associate all video formats in .flv. While this does work, most flv files cannot retain quality anywhere near the more advanced file formats/codecs. There are 3 important video types which are supported by HTML5: MP4, WebM, and Ogg/Ogv. The MPEG-4 file type is generally encoded in H.264 which allows for playback in third party Flash players. This means you don't need to keep a .flv video copy to support a fallback method! WebM and Ogg are two much newer file types related to HTML5 video. Ogg uses Theora encoding which is based on the open-source standard audio file format. These can be saved with a .ogg or .ogv extension. So which of these file types do you need for your website? Well ideally all 3 would be great as they provide the full support spectrum. Yet this isn't realistic, and in fact, you can cover all the bases with only two of them. Here is a breakdown of what works for each browser:

Mozilla Firefox – WebM, Ogg Google Chrome – WebM, Ogg Opera – WebM, Ogg Safari – MP4 Internet Explorer 9 – MP4 Internet Explorer 6-8 – No HTML5, Flash Only! Most flash video players will support MP4 files as long as they're encoded in H.264. As such, each of these browsers will embed MP4+Flash as a final resort. This means you only need to create two different video formats to support all browsers. MP4 for Safari/IE9 and a choice between WebM or Ogg for the rest.

5.5 Database Model

5.6 Use Cases

5.6.1 Frontend

5.6.2 3-tier Architecture in Software Projection

5.6.3 Use Cases Realization

5.6.4 Evaluation

5.6.5 Summary

Chapter 6

Conclusion and Outlook

6.1 Conclusion

The chapter summarizes the presented thesis providing an overview of each chapter and describing the achievement of the thesis's goals defined in the section 1.2. At the end of the chapter suggestions for the future work are made. The presented thesis consists of six chapters including the current chapter. The chapter 1 Introduction states the motivation (section 1.1) of the work arguing the necessity of creation of generic frontend. The main research questions and the goals that have to be achieved in the thesis are described in the section 1.2. The description of the thesis's structure (section 1.3) completes the first chapter.

6.2 Achieved Goals

6.3 Future work

To conclude the thesis, suggestions for the future work are made. These suggestions are devoted to improve either the developed concept or the current implementation.
Visualization and interaction metaphor for the introduced access control
Enhanced user interface for application part
User interface to support the introduced dynamic composition

6.3.1 Conceptual Aspects

This subsection describes possible improvements on the concept level.

6.4 Addressed research questions

SLA

Differentiation of SLA depending on user type/rights. Typization of SLA and filtering . Easy enhancement in case of SLA changes, then come in to a picture next questions: How to re-sign SLA with user, how to nitfy about changes frontend and user itself, that have already once accepted it, how will system react in case user will not accept new SLA. How can frontend predefine all future changes that can appear according to SLA changes?

Privacy and security

Cookies stored on a mobile device can be easily be sotollen via hacking attack on browser or account in browser. To avoid this should be in detail researched another possibility to encrypt data or to make authorisation process more secure.

Introducing the interaction awareness

In principle, when a user open first time an application, different information can be interesting to him. Therefore, to introduce a convenient awareness, the following research questions should be investigated:

- What kind of interaction awareness information end users are really need?
- Does a user want to configure the received awareness information?
- In case of providing a configuration, what an appropriate visualization and interaction metaphor can be provided?

Integration with other application via Internet

The best opportunity to make application widely-used is to enhance list of supported hardware and software sensors. But a lot of system already propose their own sensors and corresponding app to it. How possible will be to create additional module for enhancement that will play arole of retranslator or proxy between two different systems?

Resource limitations: energy, bandwidth and computation

Since mobile devices face internal and external resource limitations, the need of differentiation of connection properties is important. For example, location data can be provided using GPS, WiFi, and GSM, with decreasing levels of accuracy. Compared to WiFi and GSM, continuous GPS location sampling drains the battery faster. One approach to this problem uses low duty cycling to reduce energy consumption of high-quality sensors (i.e., GPS), and alternates between high- and low-quality sensors depending on the energy levels of the device (e.g., sample WiFi often when battery level is less than 70 percent). This approach trades off ata quality and accuracy for energy.

6.5 Implementation Aspects

Due to the provided implemented background that supports statically defined applications and due to the lack of the time to extend this basic implementation, thedeveloped concept has been implemented partly. The future implementation work can be split up into the following tasks:

List of Figures

List of Tables

Bibliography

[1] S. Suakanto, S.H. Supangkat, Suhardi, and R. Saragih. Smart city dashboard for integrating various data of sensor networks. In *ICT for Smart Society (ICISS), 2013 International Conference on*, pages 1–5, 2013.

[2] Schuster Schill Ackermann Ameling Bendel, Springer. A service infrastructure for the internet of things based on xmpp. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*, pages 385–388. IEEE, 2013.

[3] Xianfeng Song, Chaoliang Wang, Masakazu Kagawa, and Venkatesh Raghavan. Real-time monitoring portal for urban environment using sensor web technology. In *Geoinformatics, 2010 18th International Conference on*, pages 1–5. IEEE, 2010.

[4] Michael Eggert, Roger Häußling, Martin Henze, Lars Hermerschmidt, René Hummen, Daniel Kerpen, Antonio Navarro Pérez, Bernhard Rumpe, Dirk Thißen, and Klaus Wehrle. Sensorcloud: Towards the interdisciplinary development of a trustworthy platform for globally interconnected sensors and actuators. *arXiv preprint arXiv:1310.6542*, 2013.

[5] Wikipedia. Front and back ends — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Front_and_back_ends&oldid=572495173`, 2013. [Online; accessed 15-November-2013].

[6] William A Firestone. The study of loose coupling: Problems, progress, and prospects. 1984.

[7] Erwin Danneels. Tight–loose coupling with customers: the enactment of customer orientation. *Strategic Management Journal*, 24(6):559–576, 2003.

[8] Wikipedia. Responsive web design — wikipedia, the free encyclopedia, 2014. [Online; accessed 2-February-2014].

[9] Zoe Mickley Gillenwater. Examples of flexible layouts with css3 media queries. In *Geoinformatics, 2010 18th International Conference on*, page 320, Dec 15, 2010.

[10] Nick Pettit. Beginner's guide to responsive web design, Aug 8, 2012.

[11] ETHAN MARCOTTE. A list apart, March 03, 2009.

[12] Wikipedia. Usability — wikipedia, the free encyclopedia, 2014. [Online; accessed 2-February-2014].

[13] JAKOB NIELSEN. Usability 101: Introduction to usability, 2012.

[14] Fabian Schomm, Florian Stahl, and Gottfried Vossen. Marketplaces for data: an initial survey. *ACM SIGMOD Record*, 42(1):15–26, 2013.

[15] Suman Nath, Jie Liu, and Feng Zhao. Sensormap for wide-area sensor webs. *Computer*, 40(7):90–93, 2007.

[16] Xiaogang Yang, Wenzhan Song, and Debraj De. Liveweb: A sensorweb portal for sensing the world in real-time. *Tsinghua Science & Technology*, 16(5):491–504, 2011.

[17] Josef Spillner, Marius Feldmann, Iris Braun, Thomas Springer, and Alexander Schill. Ad-hoc usage of web services with dynvoker. In *Towards a Service-Based Internet*, pages 208–219. Springer, 2008.

[18] Inc. Open Geospatial Consortium. A cloud design for user-controlled storage and processing of sensor data. In *Sensor Web Enablement Architecture*, pages 13–30. Open Geospatial Consortium, Inc., 2008.

[19] Institute for Software and Multimedia-Technology Technical University of Dresden. Vicci, 2012.

[20] M. Franke, C. Seidl, and T. Schlegel. A seamless integration, semantic middleware for cyber-physical systems. In *Networking, Sensing and Control (ICNSC), 2013 10th IEEE International Conference on*, pages 627–632, 2013.

[21] René Hummen, Martin Henze, Daniel Catrein, and Klaus Wehrle. A cloud design for user-controlled storage and processing of sensor data. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 232–240. IEEE, 2012.

[22] Josef Spillner, Johannes Schad, and Stephan Zepezauer. Personal and federated cloud management cockpit. *Praxis der Informationsverarbeitung und Kommunikation*, 36(1):44–44, 2013.

[23] Suman Nath, Jie Liu, and Feng Zhao. Challenges in building a portal for sensors world-wide. In *First Workshop on World-Sensor-Web*, 2006.

[24] Diego Calvanese, Magdalena Ortiz, Mantas Simkus, and Giorgio Stefanoni. Reasoning about explanations for negative query answers in dl-lite. *Journal of Artificial Intelligence Research*, 48:635–669, 2013.

[25] Ian Horrocks Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau. Is my ontology matching system similar to yours? In *8th International Workshop on Ontology Matching*, 2013.

[26] Ralf Möller, Christian Neuenstadt, Özgür L. Özçep, and Sebastian Wandelt. Advances in accessing big data with expressive ontologies. In Thomas Eiter, Birte Glimm, Yevgeny Kazakov, and Markus Krötzsch, editors, *Description Logics*, volume 1014 of *CEUR Workshop Proceedings*, pages 842–853. CEUR-WS.org, 2013.

[27] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big'web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814. ACM, 2008.

[28] Daniel Su Kuen Seong. Usability guidelines for designing mobile learning portals. In *Proceedings of the 3rd international conference on Mobile technology, applications & systems*, page 25. ACM, 2006.

[29] Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. Understanding mashup development. *Internet Computing, IEEE*, 12(5):44–52, 2008.

[30] Rabiu Ibrahim. Framework and model design for higher education mash-ups. In *Computer & Information Science (ICCIS), 2012 International Conference on*, volume 2, pages 938–943. IEEE, 2012.

[31] Volker Hoyer, Katarina Stanoesvka-Slabeva, Till Janner, and Christoph Schroth. Enterprise mashups: Design principles towards the long tail of user needs. In *Services Computing, 2008. SCC'08. IEEE International Conference on*, volume 2, pages 601–602. IEEE, 2008.

[32] Sean Brydon Ed Ort and Mark Basler. Mashup styles, part 1: Server-side mashups, May, 2007.

[33] Sean Brydon Ed Ort and Mark Basler. Mashup styles, part 2: Client-side mashups, August, 2007.

[34] Michael Michael Thomas Bolin. *End-user programming for the web.* PhD thesis, Massachusetts Institute of Technology, 2005.

[35] Ian Hickson and David Hyatt. Html5: A vocabulary and associated apis for html and xhtml. *W3C Working Draft edition*, 2011.

[36] Jack Moffit. *Professional XMPP programming with JavaScript and jQuery.* Wiley Publishing, Inc., Indianapolis, Indiana, 2010.

[37] XMPP Standards Foundation. Xmpp extensions, 1999.

[38] Irma Syarlina Hj Che Ilias, Sri Banu Munisamy, and Nandang Azryman Ab Rahman. A study of video performance analysis between flash video and html 5 video. In *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*, page 30. ACM, 2013.