Chair of Computer Networks

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, 10.04.2014 _____

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Systemarchitektur, Lehrstuhl Rechnernetze

AUFGABENSTELLUNG FÜR DIE MASTERARBEIT

Name, Vorname:	Andriieshyna, Uliana		
Studiengang:	DSE 2011	Matrikelnummer:	????
Forschungsgebiet:	Service and Cloud Computing	Forschungsprojekt:	DaaMob
Betreuer:	Dr.-Ing. J. Spillner	Externe(r) Betreuer:	
Verantwortlicher Hochschullehrer:		Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill	
Beginn am:	01.11.2013	Einzureichen am:	10.04.2014
Thema:	Generic Frontend for Exploring Sensor and Information Services		

ZIELSTELLUNG

Research on smart cities, connected vehicles, complex logistics across organisations and similar trends suggests a need for channeled forwarding of information from heterogeneous sources to creative third-party services and applications. 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. This involves their selection, mash-ups, and usage within a client-controlled interface.
In this master thesis, a first web-based prototype (portal) for such 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.

SCHWERPUNKTE

- Concept for a generic information and sensor service portal
- Development of the portal and associated dependency tools
- Demonstration using a convincing scenario

Unterschrift des verantwortlichen Hochschullehrers
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

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 physical device that can be an actuator or measuring device, but also a context-depending information that are available through the Web, e.g. Facebook, Tweets, RSS Feeds, Weather Forecast and etc as a real-time data streaming. 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 between sensor-specific protocol and platform, platform development itself, online aggregation and monitoring of different data sources, social dimensions[4], that do not cover criteria of a user-friendly interface, which is for user nowadays an essential part of opportunity to explore a system. Therefore the area of research of this master thesis is dedicated to define common approach of a generic frontend together with a Graphical User Interface(GUI) for exploring data sources, that can be easily and universally interconnected to any platform or a system, which has no GUI. The following sections ground the motivation of the chosen research field, define the central research questions and goals of this master thesis, and describe the overall structure of the work.

1.1 Motivation

In the recent years with the technological progress in the information systems, web technologies 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 are shifted to the Web and mobile applications. This allows fast and easy way to get information from any point on earth by using any web, mobile, traditional desktop clients. In the same time have increased a range of sensor specific 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 researches 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 a third-party user, for developer of a such system itself and future application consumer developer. As was already mentioned, most of the works in

this area are focused mainly only on a backend side, thus an essential goal fo this thesis is to provide simple and universal for integration web-based generic frontend with a main focus on a dynamic GUI generation. Users and developers should be able to explore not only 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 cover integration at the data and platform levels, weak work has been done to facilitate it at generic level. This master thesis discusses the existing web frameworks and component technologies used on presentation level integration, illustrates their strengths and weaknesses, and presents some opportunities for future work. Real-time web applications are connection-oriented applications, with web-based user interfaces, that display information as soon as it was published. Examples include social news aggregators and monitoring tools that continually update themselves with data from an external source.

1.2 Application Area

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

- **Application.** This thesis is focused on a generic multi-tier web-based frontend that delivers dynamic real-time content to end-users via user-friendly graphical interface. Such type of universal frontend acts as middleware between provider-specific data source and a user. By supporting a concept architecture any backend system can easily send data to the user, without needs to implment a specific frontend.

- **Infrastructure.** Implies a virtualized system architecture of a concept and an application stack required for running an application of the respective prototype, based on 3-tier architecture.

- **Tools and Protocols.** Assumes a set of web-based frameworks and Internet of Things(IoT) protocols for real-time data streaming of a wide range of resources used to explore and maintain services of the aforementioned application.

1.3 Research Questions and Goals

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

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

- Which architecture should have a concept of a generic frontend?
- How should it be designed in order to provide ease of integration with any backend system?
- What type of data sources have to be retrieved and what is the most universal interface for collaboration between backend and frontend systems?
- How can be a generic GUI designed and implemented by providing dynamic content retrieving?
- Which protocol can perform real-time data streaming independently from type of data in it?
- 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 surrounding sensors. And provide concept of a generic frontend which can be easily interconnected through the supported interface and extensions points with each other.

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 basics of used terminologies and the foundation platforms. Requirements to a concept of a generic frontend that has to be developed are also introduced in this chapter.

Chapter 3 is devoted to the state of the art analysis. Related research and consumer-based works in the area of sensor-based data retrieving approaches, such as: portal and mashup 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, web frameworks and necessary criteria, defined in *Chapter 3*.

Chapter 5 provides the implemented functionality of the concept and evaluated by using convinsing scenario. An example of use case scenario are proposed, described and performed in order to cover the fulfillment of the defined requirements by the developed solution. Important methods and extensions of a an interface are described in details.

Chapter 6 concludes the master's thesis underlining and evaluating achieved goals and providing prospects for the possible future work.

Chapter 2

Foundation 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 formalized goal, fundamental terms in the presented research area are formulated at the beginning of the chapter. Basic aspects concerning data source 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, which in details described 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 confirm 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 backend simplifies development and separates maintenance. Therefore need to be distinguished what are the main requirements of 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 separated 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 for every type of data format, Information resources in the Web), such that it can be distributed across Internet, and can be applied to any resource and mobile device everywhere. It also includes ease of expansion and integration with backend system, independently from platform used on a backend and frameworks/tools used on a frontend side.

- *Multi-user binding:* defining to every user according to their type and rights visibility rules and load personilized bookmarks/preferences, targeting by using JID and pass-

word. Every account have to be unique binded to a 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, which gives an easy to understand user-friendly presentation of a different data types. User-oriented design provides fast and simple awareness of what frontend propose and how it can be explored by user.

2.1.1 Fine-grained structure

The fine-grained structure of a system provides a possibility to distribute task between responsible modules of a system. It guarantees performing tasks in parallel, moreover, exchange, enhance and add new parts of a system, without influence to an any another module. Also it guarantees necessary performance in response time and makes a system to be highly scalable. 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 independent modules, responsible for a separated task, that in the same time gives a possibility to scale the system. Also, every module can be implemented independently from framework or language or interface of an implementation.

2.1.2 Loose Coupling

The goal of loose coupling is to reduce dependencies between system's parts. 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 as shown on the 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 an importance of realizing loose coupling in generic systems, 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/GUI elements
- 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 of a changes in content which have to be retrieved by GUI. Another aspect to consider is the probability of layout changes during the lifetime of the application. Due to mergers and acquisitions and system consolidations, the layout underneath the application is constantly changing. Without loose coupling, user will be forced to adapt 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.

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, which needs an acceptable interface for integration with a backend and adaptable modules for its data structure, to easily interconnect 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.
- COnsumer application developer, which is the main task to find necessary sensors as fast as possible, explore metadata provided by data soutce, SLA if exist and get an example of provided by sensor data format, configuration of end-points and system architecture.
- Simple 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.
Consumer application developer plays a role of a new 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 itself from an inside.
After user log in 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 useful 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-Platforming

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 web-based application may notice that application itself 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 fully-functional application to the user.

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 re-sizing, 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 characteristics of usability as: learnability, efficiency and satisfaction. Rest of it together with possible enhancement will be proposed in Chapter 6 in the 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 differentiated as physical and software sensors. Sensor in such a way not just a hardware, but everything that "senses"/provides a data.

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 structure and differentiate it according to their functionality and make a standard data format, by using which, backend and frontend can not only easily integrate new sensors, but can always rely on it and understand how to visualize and present it to the user.

Together with software sensors the proposed data standard is justified and explained in the Section 4.4

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 coordinate, weather forecast with all arising from it data, e.g. ambient temperature, humidity, pressure, traffic feeds. Not only physical sensors become available through the Web, but through Web 2.0, blogs, tweets, RSS feeds, social networks makes possible to create human-behavioral sensors. 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 everything that can be interconnected through the standard web interface by using internal software from provider.

An important research question is: how to manipulate data, made by different vendors, by using single dashboard? To get an answer to this question and define most common approach, was used categorization by vendors[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 category 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.

Such categorization clarify main important attributes in which customer can be interest in, namely: data access, data output, trustworthiness, data origin and time frame. Such typization appear as an appropriate attribute in a data standard, which described in details in the Section 4.4

As shown in the Table 2.2 the flexibility and modularity of APIs have made these the most popular of all data 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. Preview together with all data source descriptions as a metadata will be structurized and accessible through the Web API by using data standard description.

Besides aforementioned properties of a metadata 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. 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 visualize it on a GUI. Doesn't matter how often the data will be changed, but all of it can be transferred 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 useful 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 the concept, that support any type of data, is to configure interface between backend and frontend, by using streaming channel.

The data categorization on which concept of a generic frontend focused on are:

- feeds from the web, RSS, video and music streaming
- traffic information sensor network from the city ("smart city")
- home automation sensors ("smart home")
- continuously growing log files from researchers
- crowdsourced sensors, for example road quality or running speed measured in a decentralized way with smartphones, but again also non-sensor datasets and streams
- industrial/embedded sensors, robots
- virtual data sources which result from aggregation

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 coupling, 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 retrieved 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 details in the Chapter 5.

Chapter 3

State of the Art

The following chapter covers an overview and analysis of existent solutions in the related research areas of web-based third-party applications, which were designed specially for retrieving different type of sensed data. At the beginning of the chapter in the Section 3.1 the most familiar research works are studied. It covers public and private sector, 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 discovered in the this chapter are evaluated against the requirements described in the section 2.1.

3.1 Web of Sensors

This thesis is targeted at the creation of a generic user-friendly interaction system. Nowadays every project is focused on a specific area of realization and handling a narrow range of data types, such as urban environment[3] or real-time environmental monitoring portal or geospatial infrastructure to efficiently collect and serve a vast field of a data over the web[1] and much more as described above. A lot of work related to Cloud-based information resources and Internet of Things(IoT), where the main focus on information which is the result of human behaviour or usage of already implemented tools ot platforms.

Smart city[1] tool using information technology and communication (ICT) to help local government to monitoring what currently happened in the city. Web-based application for monitoring city in single dashboard to help summarize the current condition of a 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 exist a possibility to add another information e.g., socio-economic situation like public health service, economic indicator, energy supplies, etc. The results of work presents a prototype of the smart city dashboard the give more accurate information of Bandung City, one of big cities in Indonesia. 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, search and visualize spatially and geographically related data such as driving directions, directory entries, weather and traffic conditions based on Microsoft Virtual Earth and overlay with housing prices, crime rates, bus locations, and other data on top of browsable maps. SensorMap is a Web portal that 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, housing sensor metadata including the publisher's name; the sensor's location, name, and type; the data type; and freetext descriptions. DataHub to retrieve the new sensor data to enable real time services. The aggregator to summarize sensor data in a specific area to clients. And SensorMap GUI is based on Windows Live Local and therefore provides features such as zooming, panning, street maps, satellite images, and 3D views. In addition, it lets end users pose queries on available sensors. SensorMap currently supports three types of queries: geographic queries that drawing geometric shapes directly on the map specifies (for example, within a region, near a route); type queries that sensor types specify within the viewport; and freetext queries that keywords describing sensors specify.

LiveWeb Portal[16] presents the architecture, design, and approach to build a sensorweb service portal, where sensorweb is a global observation system for varied sensor phenomena from the physical world and the cyber world measured in a real-time. 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 on a 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, provides methodology of building searching engine and sensor network sharing. Based on open standards in the world of web, they propose an approach of how to build a generic system with heterogenious resources.

Internet of Things[2] presents 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 a platform. Based on two case studies demonstrated real-time capabilities of XMPP for remote robot control and service development in the e-mobility domain. And in the same time a proof of concept was made by authors in a project called ACDSense[1] that XMPP and its numerous extensions have gained momentum as a multi-purpose lightweight middleware for social computing, multi-device interaction, and communication with sensors. The concept was demonstrated with a use case of three distributed and interconnected XMPP-driven pervasive systems: standard XMPP client,

[1]ACDSense project, `http://www.rwth-aachen.de/`

sensor App, Web browser sensors dashboard. The aggregation of data as a Web application done by using multiple widgets with the help of an Inter-Widget Communication(IWC) approach based on portlets[17].

Dynvoker Portal[18] a generic human-driven ad-hoc usage approach, by including rapid service testing and dynamic inclusion of services as plugins into applications. Dynvoker is an engine, which 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[19] 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(OGC), Inc. members specifies interoperability interfaces and metadata encodings that enable real-time integration of heterogeneous sensor web 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)[20, 21]. 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 GUI 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.

SensorCloud[22], 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 presents transport security mechanisms for communication with the Cloud, applies object security mechanisms to outbound data items, and performs key management for authorized services.

CloudRemix[23] 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[24] is a portal web site for real-time real world 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: geographic queries specified by drawing geometric shapes (e.g., a region, a route) directly on the map, type queries specified by sensor types within the viewport, and 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 through API and provides system 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 specifically added libraries.

Xively Cloud Services[2] is the Public Cloud which was particularly built for Internet of Things, by giving developers standards-based services and tools, elastic scalability, and intuitive lifecycle management capabilities oriented on a business customer. 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[3] is a scalable semantic access to Big Data for effective data analysis and value creation. Optique brings about a paradigm shifted 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 conceptualizations; seamlessly integrating data spread across multiple distributed data sources, including streaming sources; exploiting massive parallelism for scalability far beyond traditional RDBMSs and thus reducing the turnaround time for information requests to minutes rather than days[25, 26, 27]. According to research publications, no research was done in a field of visualization and presentation. Implemented Frontend with adaptive and cross-browser GUI was developed specifically for this project by using up-to-date technologies (jQuery, HTML and CSS).

All research projects are presented in Table 3.1 according to such characters as: status(active or inactive in nowadays), availability of Frontend as full functional system 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
ACDSense	active	yes	yes	public

Table 3.1: State of the Art Summary

Concluding all researched projects above, become clear that the needs 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

[2]Xively ,https://xively.com/

[3]Optique ,http://www.optique-project.eu/

existent nowadays, specify types of data that have to be retrieved and possible interface for aggregation of data and collaboration between all modules defined by 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. Thus, to satisfy aforementioned requirements it is necessary to compare all available web-oriented solutions.

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

- portal with portlets,
- mashup[4],
- 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[28, 29]. 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.

[4] http://www.programmableweb.com/applications

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![5]. See for example, the "My Yahoo!" feature of Yahoo! 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.

[5]Yahoo!,`http://pipes.yahoo.com/pipes/`

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[30]. In general there distinguish the following three types of mashups[31]. 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[32]. 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 server-side mashups a content aggregation is realized on a server[33]. 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[34].

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[35].

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 a 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 a non-browser based system is native application, that is targeted to create software specificaly for operation system used on a 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 a mobile device. Android and Apple's iOS have 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)[6]. 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.

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

3.2.4 Browser Based Systems

An application that runs within the Web browser (such as Firefox, Internet Explorer or Chrome). 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 device, 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.

One of the main benefit 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[7].

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/CSS3 and jQuery Mobile, not to mention improvements in network connectivity. It is clear that, in terms of functionality, they will greatly impact the ability of Web apps[8] to compete with native apps.

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.

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.

[7]Browser based system definition,http://www.pcmag.com/encyclopedia/term/61816/browser-based-application

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

- 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.

Choosen Methodology Such a wide range of methodologies and approaches gives an opportunity to mix advantages of a mashup and non-browser based application in a best way. Thus, 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[36].

- Compatibility(inherit all previous techniques and standards)
- 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)

The mashup methodology has a clear and easy adoptable for the web three level structure, described above: presentation, web services and data.

3.3 Summary

The chapter has introduced main approaches of 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: temperature, humidity, traffic sensors. Was studied not only scientific research projects but also private customer-oriented solutions for Big Data management. In total 13 projects was structured and characterized according to formulated properties of a generic frontend concept.

The section 3.2 consists main methodologies of a design and implementation of a generic frontend: portal with portles, mashup, native applications and non-browser based systems. As result, mashup technologies based on browser satisfy all necessary requirements to create generic frontend for exploring sensor and information services.

Based on this decision in the chapter 4, defined system architecture and described responsibility of an 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 requirements formulated in chapter 2.
Section 4.1 begins this chapter with software architecture according to 3-tier architecture, which consists client tier, application tier and data tier. Next sections presents detailed descriptions of every module, responsible for providing component functionality of every tier. Summary of this chapter underlines responsibility and requirements to every part of system infrastructure, such that prototype can be realized and implemented accordingly.

4.1 3-tier Architecture

Since the concept of a generic frontend should be scalable and easy integrated to any kind of backend, become necessary to determine software architecture according to a 3-tier architecture, in which presentation, application processing, and data management functions are logically separated. Multitier architecture provides to developer abstract structure of every module and gives a possibility to define in which module of system developer is interested in. Also it describes how different parts of frontend interconnected with each other and what are extensions and integration's points for backend.

Figure 4.1 shows concept infrastructure:

- **Client Tier:** web-based GUI
- **Application Tier:** application logic, interface of collaboration between tiers, backend integration point
- **Data Tier:** data from different types of sensors

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

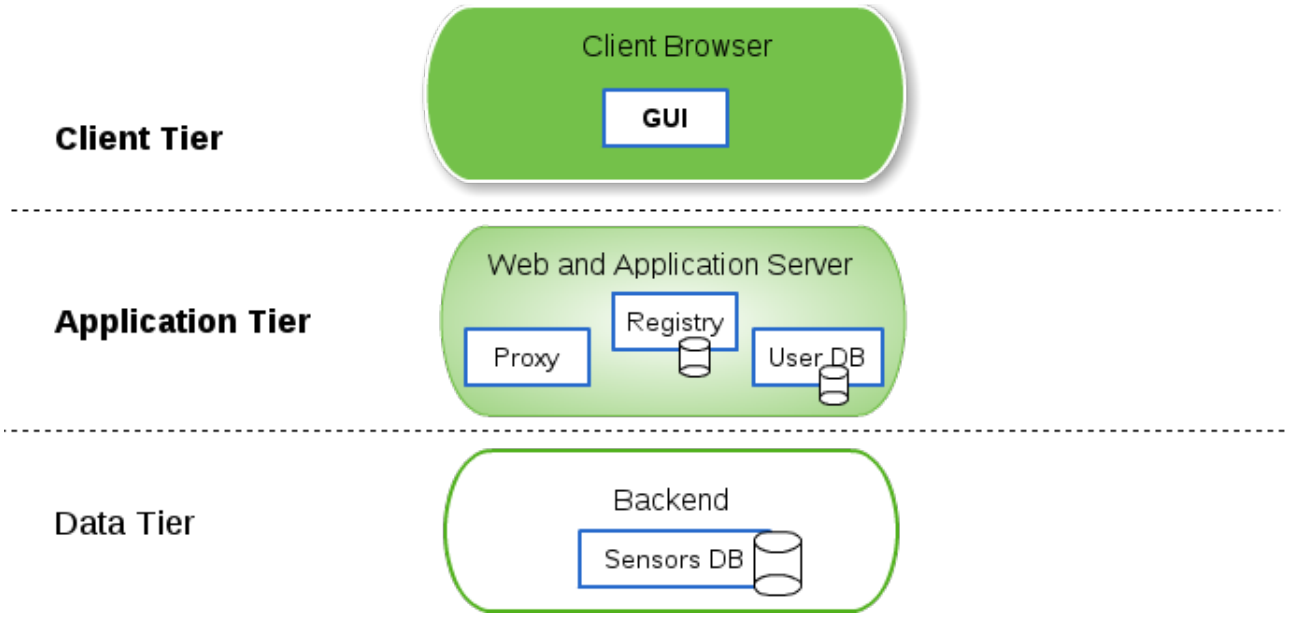

Figure 4.1: 3-tier Architecture

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 each other, defines an interface of interconnection between client tier and data tier. Besides possessing application logic between two another tiers of infrastructure, this tier also consists integration point for backend systems.
Data Tier consists source of data that have to be retrieved by application tier to a client tier, by request from a user. This tier keeps data neutral and independent from application server or business logic. Giving data its own tier also improves scalability and loose coupling.

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[37](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. Thus, to ensure highly adaptive GUI independently from data that have to be retrieved MVC pattern come into a picture. As a part of frontend logic it will be described in section 4.3.

The next section gives a detailed explanation about structure modules in every tier respectively to 3-tier architecture.

4.2 Client Tier

The client tier or another name is presentation tier is a layer which user can access directly such as a web page by using browser. It is the first that user see and comprised of widgets structured according to the responsive layout. This tier consists GUI by using which, user can communicate with sensor in a user-friendly way that is presented on the Figure 4.2. From a developer prospect of view, client tier gives an overview of a design layout, consists basic information about system architecture, provides technical details of a system such as: end-points configuration, API documentation or get sample applications. It is made in such a way, that by using only graphical interface, developer can retrieve on it personal data sources. Also Client Tier responsible for adaptation to any kind of mobile or desktop devices that can be used by user. Therefore as a cross-platform approach was chosen web-based solution, where all communication flows through the browser.

Figure 4.2 presents simple content layout that have to be presented on a web-page in order to satisfy all possible user requirements. It consists:

- Main four tabs: sensors list consists a list of all available sensors; subscriptions - show all data sources to which user subscribed; favorites tab saves favorite data sources among already subscribed and settings tab gives a possibility to user, to manage own account settings and also add new sensors through the defined channel.

- Log in form with user name and password fields. After user logged in, the system defines his/her rights and applies visibility rules according to credentials. Independently from user rights, a user can explore description of every data retrieved by system, but only after subscription to a data sensor become possible to get streaming data, provided by sensor. User that have an admin rights receives an opportunity to control and manipulate sensors. Normal user without privileges, receives an opportunity to get statistic and information from sensors and to manipulate his/her own account data.

- Sensor icon defines what is the current type of sensor, e.g. light, temperature, heating, robot lego, etc. It helps user easily, even in seconds, understand and catch what is the main function of a sensor in the list, especially if among icon he/she has already used some sensors and famous producers of data.

- 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. If user has already subscribed to any sensor, this sensor automatically added to a list/tab of subscriptions made by user. Also user has a possibility to define sequence order in which sensor information have to be displayed. It is done by using "favorite" label/tab. It helps user in a fast way receive information from a sensor.

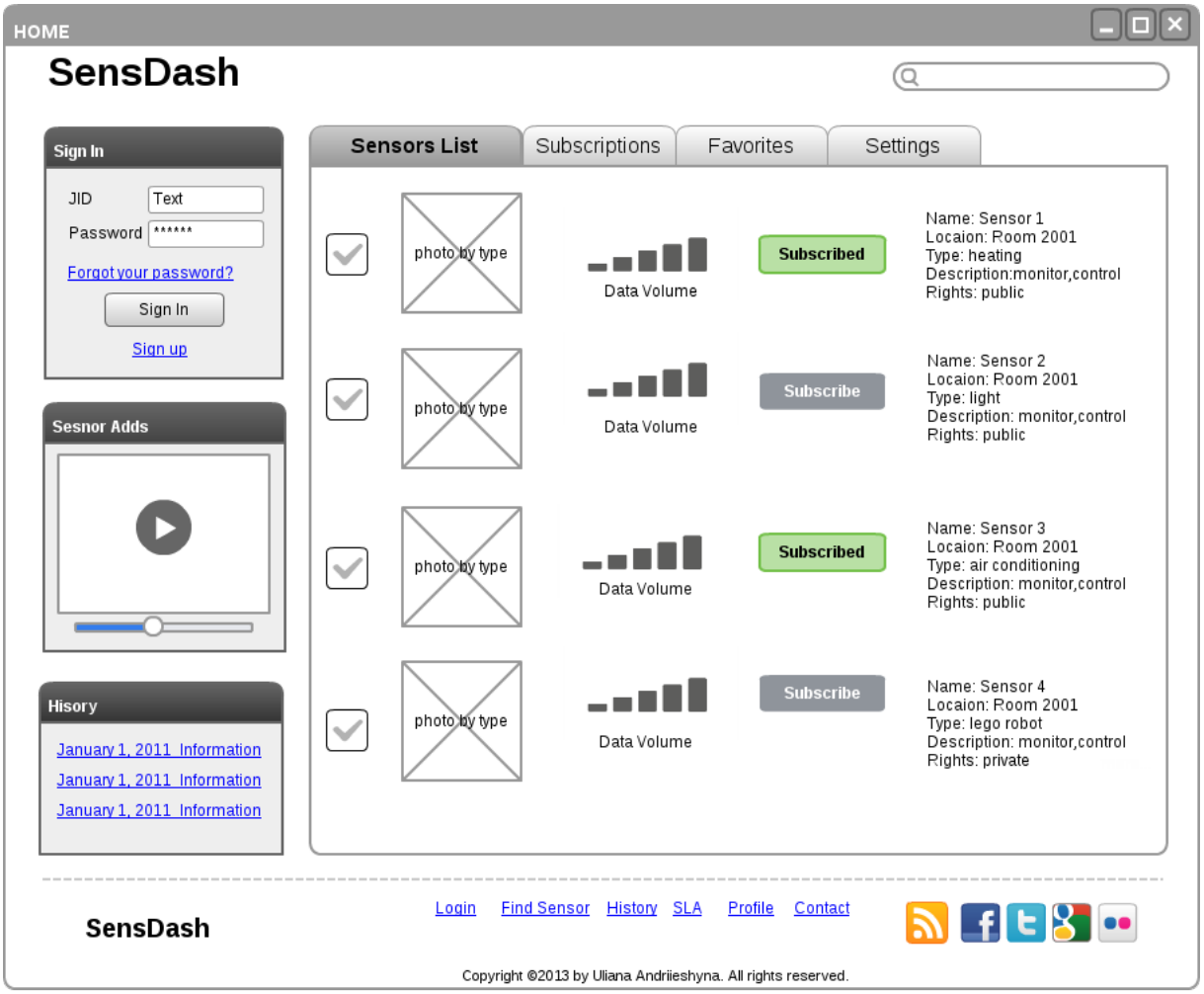

Figure 4.2: GUI Mockup

- Data Volume icon shows what is the average data stream volume needed to retrieve sensor data(Kb/s). User can self-define how possible to get such type of streaming data according to his/her available Internet connection. Ideally, the dashboard should automatically adapt quality of streaming data to necessary and possible bandwidth of available connection(Wi-Fi, 3G, GPRS), such that it will not disturb user and other application on a mobile device.

- Description and preview. To give a user full information about sensed data and to help recognize how useful will be to make a subscription, the best is give a preview or examples of data. It's not only description but also real graphics, real examples of video or audio, images etc.

- Access and providers. Data can be private or public. If public, user don't have to accept any SLA to get real data from a sensor. But for private data very important to accept SLA between subscriber and provider, before user will get any real data.

- Search panel. Have to be done by filtering available sensors only based on information available for Frontend. Without any queries to Backend and waiting for answer.

The common use case shown on Figure 4.3. User can use any type of mobile device and his favorite browser to receive information from data sources(sensors) by using web-page as a dashboard. Once a user log in to the dashboard, he/she can explore all available sensors around him. If he/she already has used this dashboard at another location and subscribed to some sensors, it will be automatically available in another tab named subscriptions.

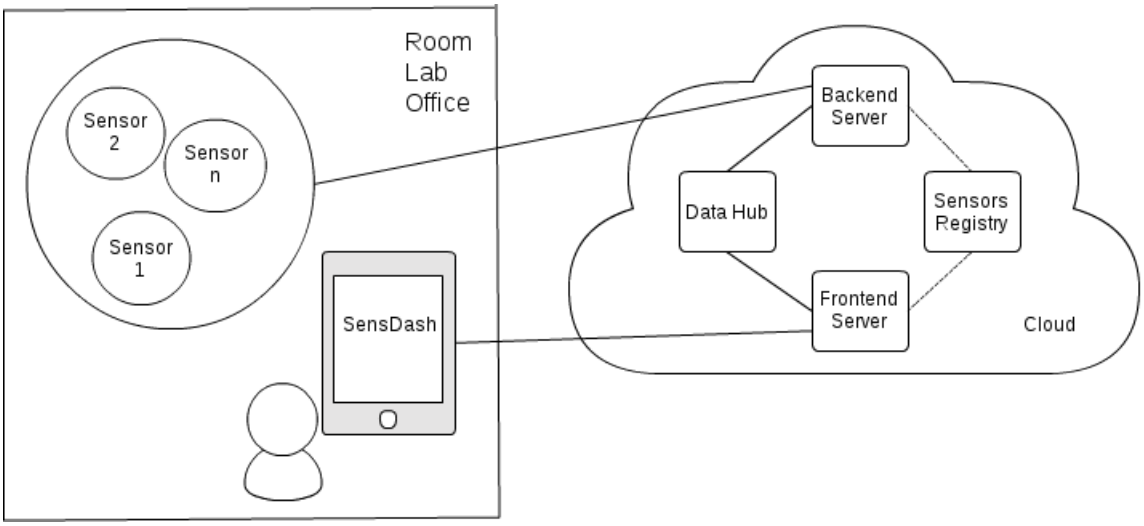

Figure 4.3: Use Case

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, Data Hub and Web-based Frontend. All these modules connects to each other as shown on the Figure 4.4.

System Architecture splitted to a backend and frontend implementation, that gives an overview how these two part interconnected. Also on the Figure 4.4 presents a end-user, which doesn't know anything about implementation behind the scenes. Registry and Data Hub defined and standardized by frontend in order to provide as much as possible common interface for collaboration between backend and frontend.

4.3.1 Sensors Registry

First of all the Sensors Registry is a module responsible for storing an info about available sensors registered in the network in order to provide information for a user. Frontend gets

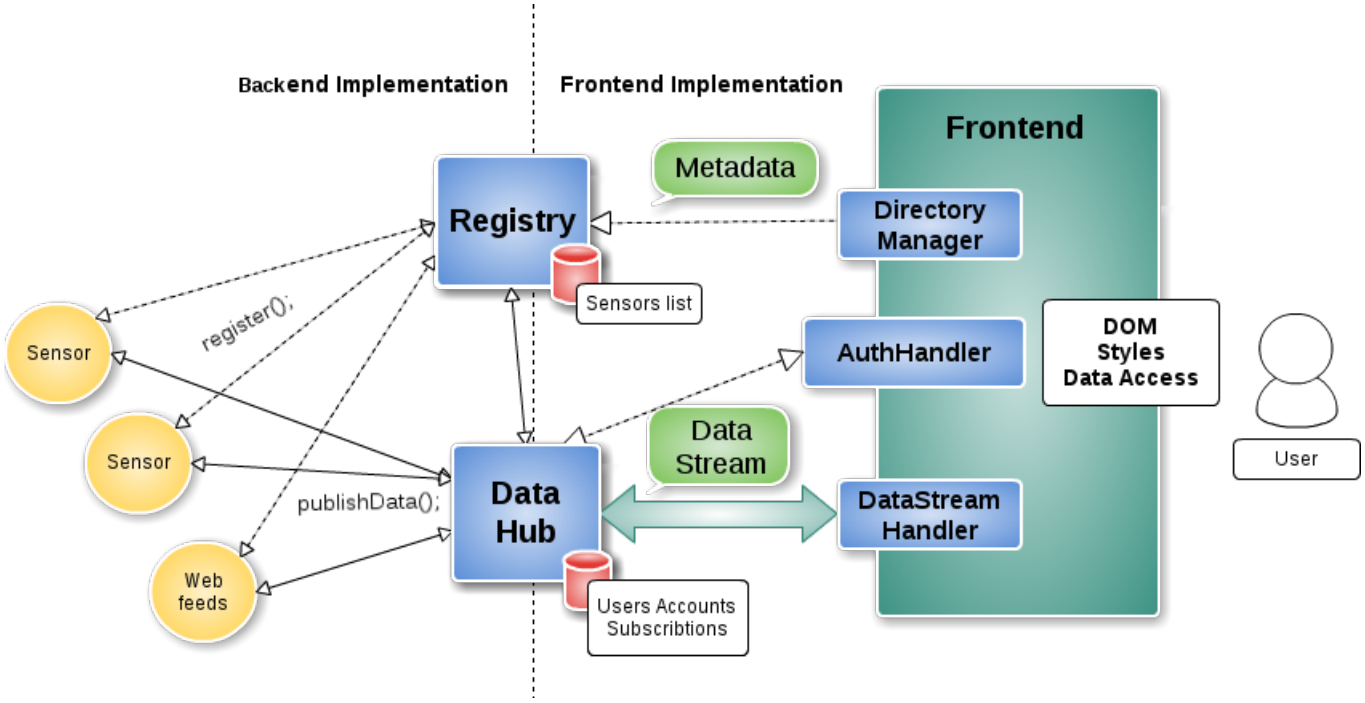

Figure 4.4: System Architecture

info from Registry about registered sensors and their availability. By using simple Web API, so that any Registry that implements suggested API and returns valid JSON, which consists such info as: id of sensor, description, availability(true or false), access(private or public), provider of sensed data, SLA, necessary bandwidth for retrieving data, title and type. By using Web 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 Registry, a data publisher must first register the sensor by providing its static description. This meta data describes sensor name, sensing type, availability, data type, access type, provider name and further details, and also if possible some real examples of data or preview possibility, as well as a free text description of the sensors, and is used in searching sensors for a given user query.
Such structure helps system to structure sensors according to the type, as a result to automatically generate the graphical container for it. Based on standard description, rendering of the sensor data in the client tier become essential part, because it simplifies the way of sensors integration.

And secondary, the Registry, provides simple interface standard in JSON format to encode these properties that can be simply transferred by using Web API approach. Thus, it is much simpler and light-weighted than XML. And separate metadata of a sensor from a real data.

4.3.2 Data Hub

Since Sensors Registry responsible for collecting metadata of sensors, Data Hub is responsible for mapping interface of particular sensor data stream into a format supported by frontend

and delivered through the common universal protocol. It means that Data hub has to satisfy next requirements:

- be aware of metadata provided by Registry to frontend
- bind metadata from registry and user choice
- get and parse sensor streaming data and reconvert it to the type supported by universal protocol
- implement universal protocol to provide exchange message with a server in order to make possible retrieve streaming data from sensor
- store history of a data sources

Authentication Handler

Authentication Handler responsible for registering user in system, providing User ID and define user role. After a user gets necessary ID and confirm his/her personality by using password and name, system automatically applies visibility rules. After a user verifies and confirms credentials, it becomes possible to bind user id with personal preferences. Such preferences can be: user subscriptions, favorites, social sharing and even cookies. Since all these data have to be loosely coupled from backend, and web-based application has no access to internal storage of portable device, this problem solved enhancing functionality of XMPP protocol and using namespace approach of XMPP server storage.

Backend entry point Data Hub and Sensor Registry are two modules that have been standardized by frontend and have to be fully implemented on a backend side. Since both of this parts support common standards such as Web API, HTTP GET, AJAX, JSON file format, it makes easily possible to implement it inside any backend system.

4.3.3 Web-based Frontend

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 is responsible for robust and efficient serving of static files. The goal is to exclude dependencies on concrete backend platforms or frameworks and to provide generic frontend as easily plugable component. Such a common and simplified design decision makes possible to extend and scale every part of distributed system independently. Specific operation logic like authentication of user, registration of sensors and users are delegated to external components such as Registry, Data Hub and Authentication Nadler, these external components can be interchanged without dependency to the system itself.

Interfaces

On the Figure 4.4 from a frontend prospect of view exist 3 communication channels:

- Registry to Directory Manager (one-way connection)
- Data Hub to/from DataStream Handler (asynchronous duplex connection)
- Data Hub to/from AuthHandler (synchronous duplex connection)

Registry to Directory Manager Interface
According to Web API and simple structure of a registry data, communication between these two modules have to be done by using HTTP GET request and JSON format, retrieved from Registry. It is one-way connection, because frontend needs only get metadata from Sensors Registry and all changes that occurs on a web page end through interface between Data Hub and DataStream Handler. Such structure provides possibility independently integrate new Sensors Registry.

Data Hub to/from Directory Manager Interface/AuthHandler
Communication between Data Hub and another 2 modules: Directory Manager Interface and AuthHandler, have to be supported through the one common universal interface. This interface have to satisfy next requirements:

- have a public license
- handle Web API requests
- support service-user n:n connections
- support different types of data sended through the one channel, message differentiation
- keep connection alive(stateful, through the TCP connection)
- simplicity of enhancement and customization

Three inetfaces have to work with data sources in the way that shown on Figure 4.4

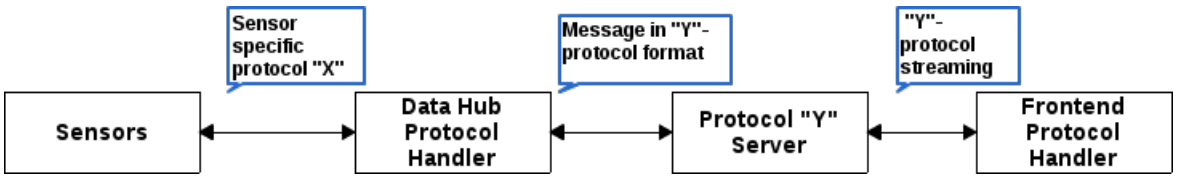

Figure 4.5: Protocol flow

Such a structure decouple sensors specific interface and interface between frontend and backend. In such a way, any sensor can be added and retrieved by frontend.

The web browser is probably the most deployed and most used application platform that has ever existed. Web browsers exist on every kind of computer and even many mobile

phones, and more importantly, the users of these devices tend to be very familiar with the browser and web applications. As more and more sophistication has been demanded of web applications, new technologies and abstractions have been created to evolve the platform. XMPP brings yet another new set of technologies and abstractions, but with it comes enormous potential for real-time, interactive, and collaborative applications. The rise of the social Web has given rise to social applications, and if developers want to take more steps toward connecting the human race, technologies like XMPP will help them do it. For XMPP developers, targeting the web browser as a platform makes enormous sense.

In order to satisfy all aforementioned requirements, nowadays exist two main protocols: *XMPP[38]*:a protocol best for connecting devices to people, a special case of the device-to-server pattern, since people are connected to the servers and $MQTT$[1]: a protocol for collecting device data and communicating it to servers. Based on their functionality and specification can be managed questions of authentification, registration and user preferences storage.

MQTT

the Message Queue Telemetry Transport, targets device data collection (Fig. 4.6). As its name states, its main purpose is telemetry, or remote monitoring. Its goal is to collect data from many devices and transport that data to the IT infrastructure. It targets large networks of small devices that need to be monitored or controlled from the cloud. MQTT makes little attempt to enable device-to-device transfer, nor to "fan out" the data to many recipients. Since it has a clear, compelling single application, MQTT is simple, offering few control options. It also doesn't need to be particularly fast. In this context, "real time" is typically measured in seconds.

A hub-and-spoke architecture is natural for MQTT. All the devices connect to a data concentrator server. So the protocol works on top of TCP, which provides a simple, reliable stream. Since the IT infrastructure uses the data, the entire system is designed to easily transport data into enterprise technologies like ActiveMQ and enterprise service buses (ESBs).

MQTT enables applications like monitoring a huge oil pipeline for leaks or vandalism. Those thousands of sensors must be concentrated into a single location for analysis. When the system finds a problem, it can take action to correct that problem. Other applications for MQTT include power usage monitoring, lighting control, and even intelligent gardening. They share a need for collecting data from many sources and making it available to the IT infrastructure.

XMPP

XMPP was originally called "Jabber". It was developed for instant messaging (IM) to connect people to other people via text messages (Fig. 4.7). XMPP stands for Extensible Messaging and Presence Protocol. Again, the name belies the targeted use: presence, meaning people are intimately involved. XMPP uses the XML text format as its native type, making person-to-person communications natural. Like MQTT, it runs over TCP, or perhaps over HTTP on top of TCP. Its key strength is a name@domain.com addressing scheme that helps connect the needles in the huge Internet haystack.

[1]MQ Telemetry Transport,`http://mqtt.org/`

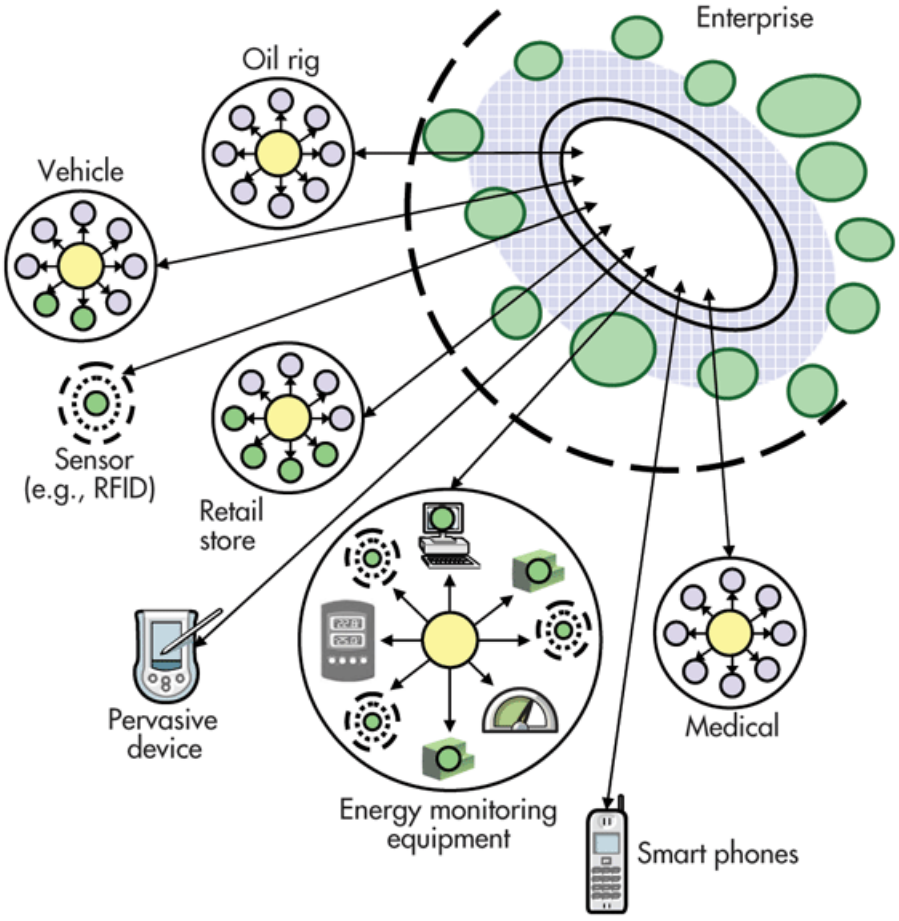

Figure 4.6: Message Queue Telemetry Transport

In the IoT context, XMPP offers an easy way to address a device. This is especially handy if that data is going between distant, mostly unrelated points, just like the person-to-person case. It's not designed to be fast. In fact, most implementations use polling, or checking for updates only on demand. A protocol called BOSH (directional streams over Synchronous HTTP) lets severs push messages. But "real time" to XMPP is on human scales, measured in seconds. Its strengths in addressing, security, and scalability make it ideal for consumer-oriented IoT applications.

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.

After a sort introduction become clear that XMPP fully satisfy all requirements in order to be used in concept of generic frontend. So the interface *Data Hub to/from Directory Manager Interface/AuthHandler* provided by using XMPP.
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

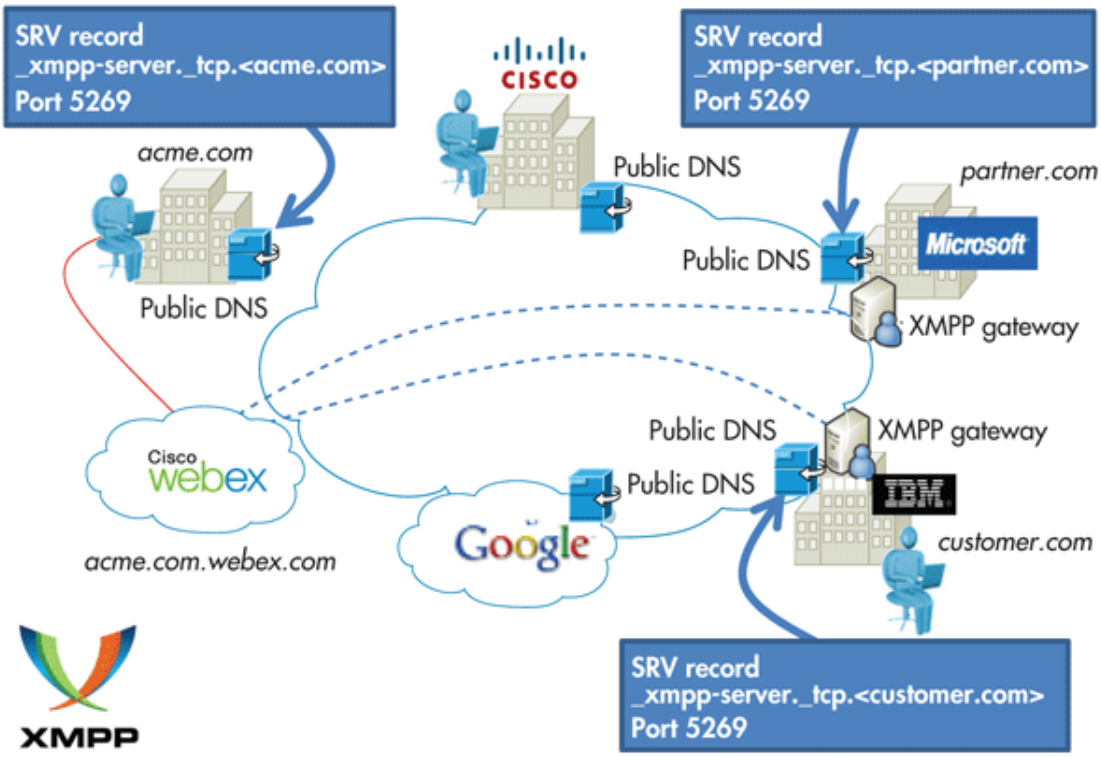

Figure 4.7: The Extensible Messaging and Presence Protocol (XMPP)[2]

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[39].

Pushing Data

HTTP clients can only request data from a server. Unless the server is responding to a client request, it cannot send data to the client. XMPP connections, on the other hand, are bidirectional. Either party can send data to the other at any time, as long as the connection is open. This ability to push data greatly expands the possibilities for web applications and protocol design. Instead of inefficient polling for updates, applications can instead receive notifications when new information is available. Not only does this result in many fewer requests, it also makes the latency between the time new information is available and the time the client is aware of this information nearly zero.

Pleasing Firewalls

Some web applications support the use of HTTP callbacks, where the web server makes requests to another HTTP server in order to send data. This would be a handy feature to push data if it weren't for firewalls, network address translation (NAT), and other realities of the Internet. In practice it is very hard to enable arbitrary connections to clients from the outside world. XMPP connections are firewall and NAT friendly because the client initiates the connection on which server-to-client communication takes place. Once a connection is established, the server can push all the data it needs to the client, just as it can in the response to an HTTP request.

Improving Security

XMPP is built on top of TLS and SASL technologies, which provide robust encryption and security for XMPP connections. Though HTTP uses SSL, the HTTP authentication mechanisms did not see much implementation or use by developers. Instead, the Web is full of sites that have implemented their own authentication schemes, often badly.

Statefulness

HTTP is a stateless protocol; XMPP is stateful. Stateless protocols are easier to scale because each server does not need to know the entire state in order to serve a request. This drawback of XMPP is less onerous in practice because most non-trivial web applications make extensive use of cookies, backend databases, and many other forms of stored state. Many of the same tools used to scale HTTP-based applications can also be used to scale XMPP-based ones, although the number and diversity of such tools is more limited, due to XMPP's younger age and lesser popularity.

- *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 group chat, network management, content syndication, collaboration tools, file sharing, gaming, remote systems control and monitoring, geolocation, middleware and cloud computing, VoIP and Identity services.

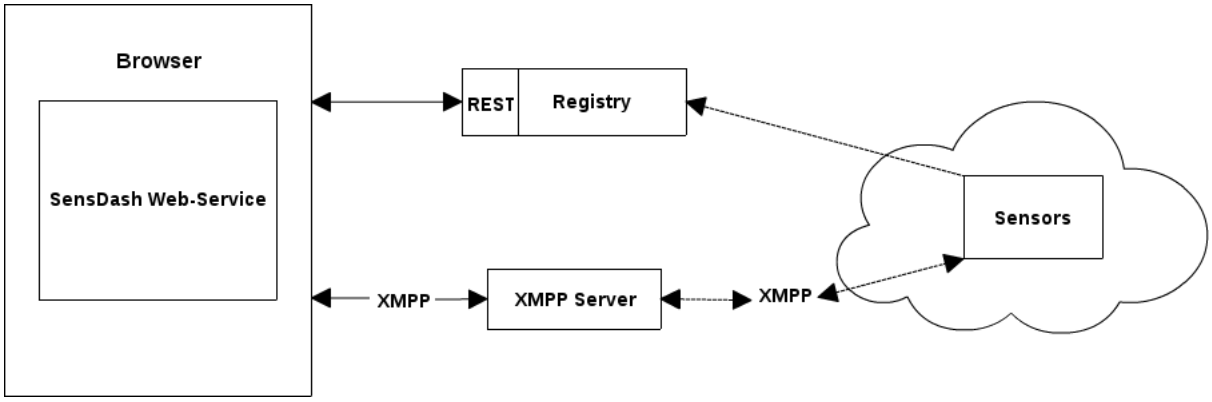

Figure 4.8: Interface

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) for the server where that user resides, separated by an at sign (@), such as username@example.com.

JavaScript MVC

JavaScript has become one of the most popular programming languages on the web, when the usage of Ajax came to light and professional programmers gave importance to the

responsiveness of the page. But now the language has become more popular than ever as the User Experience has become the key part of web development. Accessing web is not limited to browsers alone – there are lot many devices with varying screen sizes accessing the same content. With the rise of HTML5 and CSS3 the web will become more adaptive and responsive than ever and JavaScript plays a major role in it. It has also gained popularity in the server side programming which is made possible by NodeJS framework.

Increase in usage of JavaScript in modern applications demand developers to write maintainable code, separate concerns and improve testability. JavaScript is a "class" less language and it was not designed to support Object Oriented Programming. There are some DOM manipulation libraries like jQuery which simplifies client side scripting of HTML, they actually do not solve the problem of effectively handling separation of concerns. The problem with this is that it doesn't take long to get lost in a nested pile of jQuery callbacks and DOM elements without any real structure in place for applications. Source code that has a complex and tangled control structure, especially one using many GOTOs, exceptions, threads, or other "unstructured" branching constructs can lead to become a bottleneck.

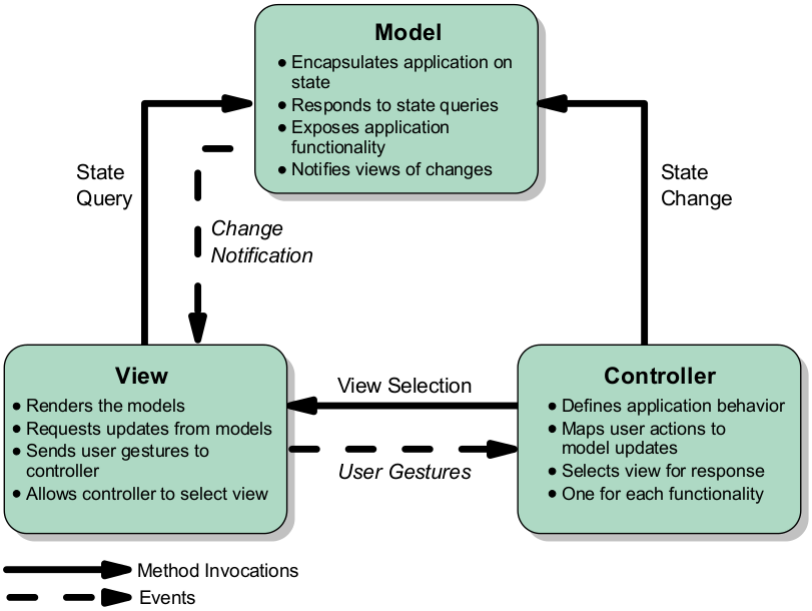

Figure 4.9: MVC Pattern

In the design shown on the Figure 4.9, takes place common Model-View-Controller pattern. Where a 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 request. 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.

Though all the frameworks out there somewhat tries to be MVC but they do not necessarily follow the pattern strictly. The idea of all the patterns is to separate Model, View and Logic that hooks the two behind which is the controller for the best separate of concerns. On the other side we do have other libraries which implement Model-View-Presenter(MVP) and Model-View-ViewModel(MVVM) pattern. For this reason we will refer these frameworks as MV* implementation. So it's much more depends on framework and researched in the Chapter 5 in details.

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. Also this tier is a entry point for backend system, thus all requirements to it already described in the section 4.3. Besides aforementioned requirements, based on section 2.2, which provides widely encompassing overview of data sources, according to it made typization of data. A User, neither a developer have to be clarified with different types of data sources that cover generic frontend. According to it, data sources can be retrieved by web page as plain text, maps of value(in order to draw graph) and images.

These three types of data changes with some time-frequency and automatically retrieved by client tier as a streaming data. So an essential part to is to determine possible characteristics of streaming data that can be retrieved by Client Tier in a lightweight scenario for mobile devices.
Streaming data 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 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[40].
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 streaming delivery because it allows the stream to actually adapt the streaming experience to the quality of the network and the device's CPU.
In other words, the media stream can increase or decrease the bit rate and resolution (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.

An important aspect in streaming data that some of data can be cashed on a server side, thus become possible to retrieve data after it was produced. But some streaming data consists only live data, thus there is no other options except live streaming, where the connection configuration, aliveness and quality become a key aspect. Consider absence of any concrete backend system, cashing on it's side and on frontend side can be covered in scope of this Master Thesis. As a good example of live streaming will be simulated map of values, images and text.

Sensor Functional Characteristics
An essential part of concept to garantee reliable and secure data transportation. Thus, based on a system architecture every data source can acquire additional properties:

- reliability
- perfomance
- security

All these three characteristics rely on quantity of available DataHubs which includes XMPP modules and provides data streaming. Since DataHub have to provide data from sensor through XMPP connection it has to support XMPP data channel configurations and plays a role of end-point for Frontend. If sensor has 2 end-points it should be mentioned in Sensor Registries attribute, the same as type of data transfer protocol(PubSub or MUC).

A simple sensor, with a low level of reliability, has one and only one end-point. A reliable sensor has two end-points, where is one is a primary, the other one is a backup/failover. A highly reliable sensor has three endpoints, which is guarantee data receiving in case if another end-points will failover.

A high-performance sensor has more than 2 end-points, out of which one is always chosen based on minimal latency message income. In case of big data transportation all data can be splitted and sended in parallel through the more than 2 end-points.

A secure sensor has at least two end-points, each of which carries some part of a data. Without each part become not possible to retrieve whole data at all.

All this characteristics have to be automatically captured by Frontend from a predefined attributes in a standard specification provided by Registry. So that developers can get to know about the expected quality of the data streams even in case some endpoints become unavailable or add new end-points. In the meanwhile, Frontend has predefined logic, which calculate numbers of related end-point for every sensor and by using graphical elements such as icons and labels, presents this information to the user.

4.5 Summary

In this chapter, according to a 3-tier architecture, the first web-based concept for sensor streaming services is to be created. Such type of structure provides clear separation of concern between different module of concept. Client tier consists GUI description and content, Application Tier provides application logic in order to interconnect backend and Client Tier, and finally Data Tier describes typization of data in order to easily interconnect it with Application Tier and visually retrieve it by using Client Tier. To every data souce was assigned such characteristics as reliability, perfomance and level of secure information. Every characteristics rely on a number of dependent for sensor XMPP end-point. As a result was identified next most important modules of Application Tier: Sensors Registry, Data Hub, AuthHandler.

Data Hub responsibilities:

- be aware of metadata provided by Registry to frontend
- bind metadata from registry with user choice
- get and parse sensor streaming data and reconvert it to the type supported by XMPP protocol
- implement XMPP protocol to provide exchange message with XMPP server in order to make possible retrieve streaming data from sensor through XMPP server

Registry responsibilities:

- stores an info about available sensors registered in the network

- provides simple interface standard in JSON format

Web-server responsibilities:

- handles delivery to end user static structure of the web-page

Frontend responsibilities:

- Interconnect all modules by using appropriate interfaces: Web API for Sensors Registry and XMPP Protocol for Data Hub and AuthHandler
- Build a responsive and adaptable to changes GUI
- Scalable system structure

Backend responsibilities:

- Implement Data Hub interface including XMPP with required extensions
- Implement Sensors Registry

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 cookies for authorization
- **Application Tier** consists Web-server, XMPP server(with extensions) and guarantee appropriate interface of collaboration between tiers via JSON format and defined structure
- **Data Tier** consists data streaming of a map of values, images and text

To make evaluations real, system will use data from temperature sensor which is provided as a testing environment in scope of ACDSee project, together with TU Dresden, BTU Cottbus - Senftenberg and RWTH Aachen University. It locates in the room INF3084, Faculty of Information Technology, Chair of Computer Science.

5.1 Development Environment

As a programming languages for implementation of this work jQuery[1] and HTML5[2] together with CSS[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 project code to be portable over different platforms and provides opportunity for robust and

[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`

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

XMPP Client on JavaScript

Web applications are cross-platform, easily deployable, and come with a large user base already familiar with them. More than that, web technologies make heavy use of HTML, and it is often the case that tools for manipulating HTML work very well on XML, and therefore, on XMPP. One such tool, familiar to many web developers, is the jQuery library. jQuery makes many mun- dane manipulations of HTML and CSS easy and fun. This power is also almost equally applicable to XML data, because it shares a very similar structure. This book's applications use jQuery to process and manipulate both the user interface, a combination of HTML and CSS, and the incom- ing XMPP data. Web technologies have their warts, but from a practical standpoint, both web developers and XMPP developers could scarcely ask for a better platform on which to create new and wonderful things. The necessity of jQuery usage caused by existent XMPP client, which is based on jQuery exceptionally-Strophe.js.

Strophe is a collection of libraries for speaking the XMPP protocol, targeting browser-based clients. While most XMPP libraries and implementations are focused on chat-based applications, Strophe takes a grander view. It has been used to implement real-time games, notification systems, search engines, as well as traditional instant messaging. The implementations are production ready, well documented, easy to use, and easy to extend. It uses BOSH, a binding of XMPP to HTTP using long polling and WebSockets, a full-duplex single socket connection to a server. Strophe.js makes creating real-time web applications easy. Strophe.js is available under the MIT license.

5.2 Web-based Framework Analysis

- **Twitter Bootstrap**
  Twitter 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. Has some 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.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. Twitter Bootstrap in addition to vanilla CSS includes support for the two most popular preprocessors, Less and Sass.

- **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. And, JavaScript plugins including dropdowns, orbit (a responsive image slider with touch support), reveal (for creating modal dialogues or pop-up windows) 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 flexible and fluid grid system that makes creating any layout possible. 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 UI components like tabs, responsive data tables, buttons, forms, responsive navigation controls, tooltips, modals and many more. It also offers a nice set of vector social icons and a full suite of pictographic icons included in FontAwesome. 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. An awesome set of responsive, resolution independent Entypo

icons, is completely integrated into the Gumby Framework. Gumby has also a very good customization.

- **Kube**
  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]

Among all of the described above Web frameworks the most reasonable to chooose Twitter Bootstrap. It has all possible nowadays visualization modules, have been used for more than 1500+ sites, well-known among developers, reach API and examples, supported by all browsers, has a build-in fluid grid systems, that is simplify a lot building an cross-browser application.
Examples of used modules of a Twitter Bootstrap are shown on a screenshots in the section 5.7. It consists: buttons, navigation tabs bar, log in form, search field, 4-columns grid layout, modals, tooltip and carousel for images preview. All GUI was made by using only CSS and HTML structure of a Twitter Bootstrap. All animations, appearence, dynamic adaptivity was done by using special tags, anchors and classes.

5.3 JavaScript MV* Frameworks

As was mentioned in the section 4.3.3, very important to realize loosely coupled system. Needs to be separated visualization of graphical modules on a web page from a code that is responsible for retrieving of a content. Thus was used AngularJS[13]. It is an open-source JavaScript framework, maintained by Google, that assists with running single-page applications. Its goal is to augment web-based applications with model–view–controller (MVC) capability, in an effort to make both development and testing easier.The library reads

[13] AngularJS, http://angularjs.org/

in HTML that contains additional custom tag attributes; it then obeys the directives in those custom attributes, and binds input or output parts of the page to a model represented by standard JavaScript variables. The values of those JavaScript variables can be manually set, or retrieved from static or dynamic JSON resources[41]. AngularJS is a toolset for building the framework most suited to application development. It is fully extensible and works well with other libraries such as jQuery, on top of which was build XMPP. Every feature can be modified or replaced to suit unique development workflow and feature needs.

AngularJS is built around the belief that declarative programming should be used for building user interfaces and wiring software components, while imperative programming is excellent for expressing business logic[14]. The framework adapts and extends traditional HTML to better serve dynamic content through two-way data-binding(Figure 5.2) that allows for the automatic synchronization of models and views. As a result, AngularJS deemphasizes DOM manipulation and improves testability.

Design goals:
Decouple DOM manipulation from application logic. This improves the testability of the code. Decouple the client side of an application from the server side. This allows development work to progress in parallel, and allows for reuse of both sides. Angular follows the MVC pattern of software engineering and encourages loose coupling between presentation, data, and logic components. Using dependency injection, Angular brings traditional server-side services, such as view-dependent controllers, to client-side web applications. Consequently, much of the burden on the backend is reduced, leading to much lighter web applications.

Two-way data binding AngularJS two-way data binding is its most notable feature

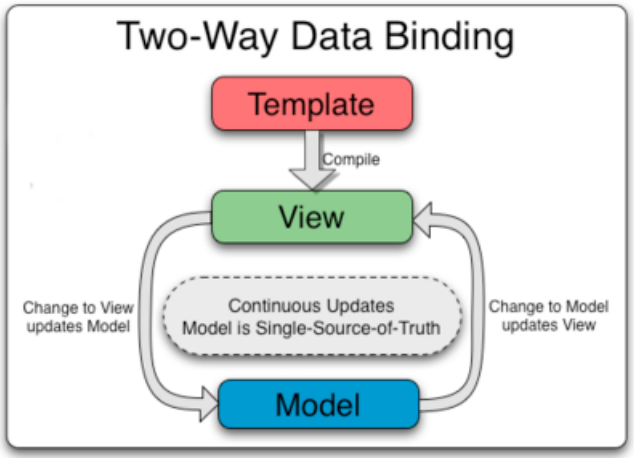

Figure 5.2: Two-way data binding[15]

and reduces the amount of code written by relieving the server backend from templating responsibilities. Instead, templates are rendered in plain HTML according to data contained in a scope defined in the model. The $scope service in Angular detects changes to the model section and modifies HTML expressions in the view via a controller. Likewise, any alterations to the view are reflected in the model. This circumvents the need to actively manipulate the DOM and encourages bootstrapping and rapid prototyping of web applications.

[14]What Is Angular?,`http://docs.angularjs.org/guide/introduction`

The way Angular templates works is different, as illustrated on the Figure 5.3. They are different because first the template (which is the uncompiled HTML along with any additional markup or directives) is compiled on the browser, and second, the compilation step produces a live view. Any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view. This makes the model always the single-source-of-truth for the application state, greatly simplifying the programming model for the developer. View in such a way simply an instant projection of a model.

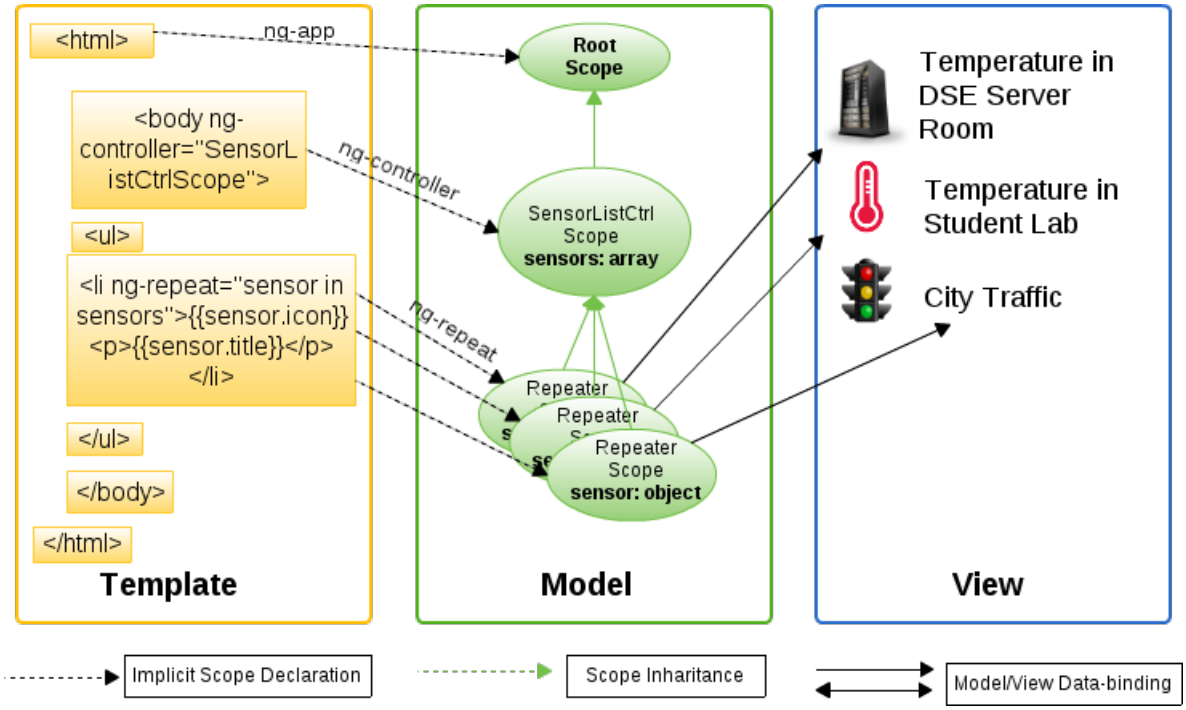

Figure 5.3: Template Model View

Because the view is just a projection of the model, the controller is completely separated from the view and unaware of it. As shown on the figure above, the resulting view can be applied to every data available ona backend. Model handeler automaticaly generates a view for every new sensor. No need to change code or add new id and dependent handlers, variables, channels. The code realization of such a flow are shown on a listing 5.1

```
1  <div id="sensor_list">
2      <div class="grid-sizer"></div>
3      <div class="masonry-brick sensor-wrapper" id="{{sensor.id}}" ng-repeat="
           sensor in sensors | filter:{title: query}">
4          <div class="sensor" ng-controller="SensorModalCtrl" ng-click="open()">
5              <div class="icon">
6                  <img class="img-responsive" ng-src="{{sensor.icon}}">
7                  <h4>{{sensor.title}}</h4>
8                  <span class="label label-success" ng-show="user.
                       check_subscribe(sensor.id)">Subscribed</span>
9              </div>
10             <div ng-show="sensor.picture">
11                 <img class="img-responsive" ng-src="{{sensor.picture}}">
12             </div>
```

```
13              <span class="description">{{sensor.description}}</span>
14          </div>
15      </div>
16  </div>
```

Listing 5.1: Template registry.html

Model explicitly integrated to the HTML. As shown on the listing 5.1: ng-repeat, ng-src, ng-click and also variables sensor.*. Listing 5.2 shows controller implementation.

```
1  var sensdash_controllers = angular.module('sensdash.controllers', []);
2
3  sensdash_controllers.controller('RegistryCtrl', ['$scope', 'Sensor', 'User',
4      function ($scope, Sensor, User) {
5          $scope.sensors = Sensor.query();
6          $scope.user = User;
7  }]);
```

Listing 5.2: Controller controller.js

On the listing 5.3 factory service, module of the AngularJS, parses REST API and creates sensors array based on factory pattern. The main goal of this pattern to create an object, independently from it's type. This pattern applied to every data source. Such type of loose coupling helps to avoid code dublication and overhead.

```
1  var sensdash_services = angular.module('sensdash.services', ['ngResource']);
2
3  sensdash_services.factory('Sensor', ['$resource',
4      function ($resource) {
5          return $resource('api/sensors/:sensorId', {}, {
6              query: {method: 'GET', params: {sensorId: 'all'}, isArray: true}
7          });
8      }]);
```

Listing 5.3: Controller controller.js

5.4 Interface Implementation

According to system architecture system has to implement two main types of intefaces:

- Registry-Directory Manager
- Data Hub-DataStream Handler and AuthHandler

XMPP connections live for arbitrarily long periods of time, but HTTP requests are quite short lived. A connection manager maintains an XMPP connection for a third party and provides access to the connection via the HTTP long polling technique.

The browser and the connection manager communicate over HTTP using a simple protocol called BOSH. Essentially, BOSH helps an HTTP client establish a new XMPP session, then transports stanzas back and forth over HTTP wrapped in a special <body> element.

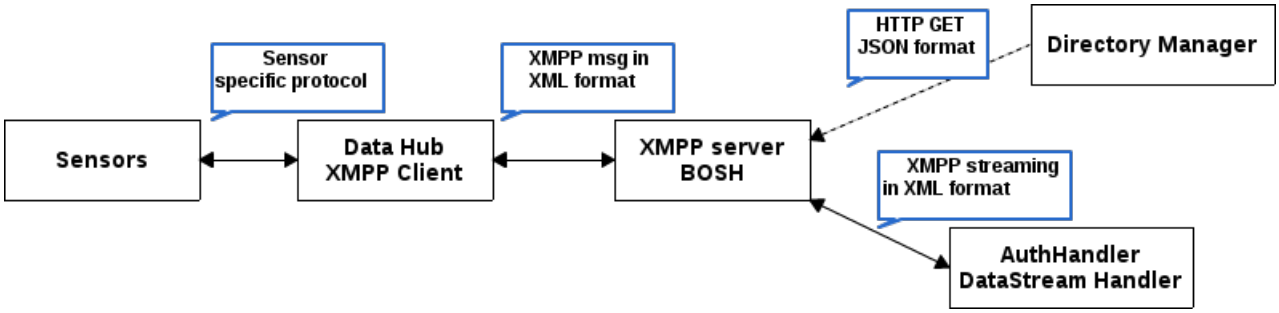

Figure 5.4: XMPP interface flow

It also provides some security features to make sure that XMPP sessions can't be easily hijacked. The connection manager communicates with an XMPP server as if it were a normal client. In this way, an HTTP application can control a real XMPP session. Because of the efficiency and low latency afforded by the long polling technique, the end result performs quite well, rivaling native connections.

XMPP connections are managed through the Strophe.Connection object. BOSH connection managers are exposed to HTTP clients as URLs, and the Strophe. Connection object you create needs to know about one of these URLs. Many XMPP servers come with support for BOSH built in, and they typically expose the service at http://example.com:5280/http-bind or http://example.com:5280/xmpp-httpbind. Some BOSH connection managers can handle communications for arbitrary XMPP servers, but generally the built-in connection managers can talk only to the server they run on.

It may seem like a lot of effort to get XMPP into a browser, but not only does this work well in practice, it turns out this technique even has some advantages over direct XMPP connections:

- Interactions with the connection manager are request by request, which allows the client to move from network to network. The managed connection stays available even if the end user's IP address changes several times.
- Because one failing request doesn't terminate the managed connection, these managed sessions are extremely robust and tolerant of temporary network failure.
- Because connection managers cache and resend data for a request, you don't have to worry about losing data when your connection is interrupted.
- HTTP is extremely firewall friendly, and because most connection managers run on standard HTTP ports, managed connections still work even in limited network environments that don't allow anything but HTTP.
- These advantages make managed connections a perfect fit for some scenarios, even when direct XMPP communication is possible.

You can create a new Strophe.Connection object just as you would any other JavaScript object, by using the new keyword 5.5. Once you have a connection object, you can call connect() and disconnect() to start and end communication with the server:

```
var conn = new Strophe.Connection("http://bosh.likepro.co:5280/xmpp-
    httpbind");
// starting a connection to example.com
conn.connect("user@example.com", "mypassword", my_callback);
// disconnecting
conn.disconnect();
```

Listing 5.4: Stanzas Format

The first two parameters to connect() are the JID and password to use to authenticate the session, and the last parameter is the callback function discussed earlier. The callback function will be called with a single parameter that is set to one of the statuses described in the previous section. A simple callback function that disconnects once the connection reaches the CONNECTED phase is shown here:

```
function my_callback(status) {
if (status === Strophe.Status.CONNECTED) {
 conn.disconnect();
 }
  }
```

Listing 5.5: Stanzas Format

Every time the connection changes its status, this callback function is executed. The callback function simply ignores any status but the CONNECTED status, and disconnects once the connection has reached that status.

Mechanics of Sessions

XMPP is a TCP-based protocol, just like HTTP, and communication happens over an established, mostly reliable socket between two endpoints. The BOSH extension to XMPP provides a bridge between this bidirectional, stateful protocol and HTTP, which is unidirectional and stateless. Because a web browser cannot directly connect to an XMPP server, a BOSH connection manager responds to requests from a browser using HTTP and uses them to manage an XMPP connection on behalf of the user(Figure). XMPP's basic model of communication is Client -> Server -> Server -> Client, and in support of this it defines a Client to Server protocol and a Server to Server protocol. Both of these protocols use XML encoded protocol directly over TCP.

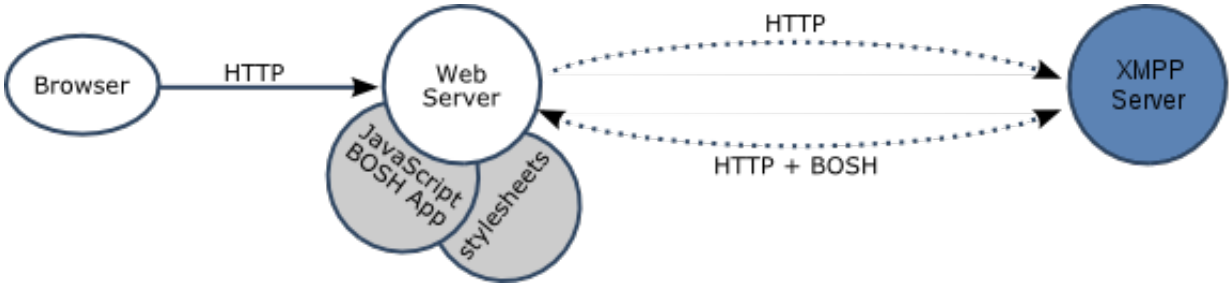

Figure 5.5: XMPP BOSH

Aside from the socket needed for XMPP communication, each managed connection has two other pieces of data associated with it: the SID and the RID. SID stands for

Session Identifier. This uniquely identifies the managed XMPP connection, and it is often a long, opaque alphanumeric string. Even though it is enough to identify the session, it is not very useful on its own. The Request Identifier (RID) identifies a particular HTTP request associated with a BOSH-managed connection. Before a connection is established, the client sends a random RID to the connection manager along with its first request. Each subsequent request increments the RID by one. The SID and the RID together provide enough information to interact with the underlying XMPP connection. Because the RID is generated randomly from a very large range of numbers, it is virtually impossible to guess the RID if you do not know one of the previous ones. Also, the connection manager will reject RIDs that fall outside of a narrow window around the current request. In this way, the BOSH-managed connection is tolerant of small errors like out of order delivery but robust to attacks like hijacking the connection. Because these two identifiers are enough to both address and make use of a managed XMPP session, if an application knows the SID and the RID, it can take over or attach to the underlying session. All it must do is send a request to the BOSH connection manager using the unchanged SID and the RID incremented by one.

The attach() function is demonstrated in the following code (listing 5.6):

```
1  var connection = new Strophe.Connection(BOSH_URL);
2      connection.attach(jid, sid, rid, callback);
```

Listing 5.6: BOSH Callback

BOSH sessions can be encrypted, and often the underlying XMPP sessions are encrypted as well. Because XMPP makes use of SASL, the authentication mechanisms tend to be quite strong.

5.4.1 XEP-0045: Multi-User Chat

Traditionally, instant messaging is thought to consist of one-to-one chat rather than many-to-many chat, which is called variously "groupchat" or "text conferencing". Groupchat functionality is familiar from systems such as Internet Relay Chat (IRC) and the chatroom functionality offered by popular consumer IM services. The Jabber/XMPP community developed and implemented a basic groupchat protocol as long ago as 1999. That "groupchat 1.0" (GC) protocol provided a minimal feature set for chat rooms but was rather limited in scope. This specification (Multi-User Chat or MUC) builds on the older groupchat 1.0 protocol in a backwards-compatible manner but provides advanced features such as invitations, room moderation and administration, and specialized room types[16].

Requirements

This document addresses the minimal functionality provided by Jabber-based multi-user chat services that existed in 2002 when development of MUC began. For the sake of backwards-compatibility, this document uses the original groupchat 1.0 protocol for this baseline functionality, with the result that:

[16]XEP0045, http://xmpp.org/extensions/xep-0045.html

Each room is identified as a "room JID" <room@service> (e.g., <jdev@conference.jabber.org>), where "room" is the name of the room and "service" is the hostname at which the multi-user chat service is running. Each occupant in a room is identified as an "occupant JID" <room@service/nick>, where "nick" is the room nickname of the occupant as specified on entering the room or subsequently changed during the occupant's visit. A user enters a room (i.e., becomes an occupant) by sending directed presence to <room@service/nick>. An occupant can change his or her room nickname and availability status within the room by sending presence information to <room@service/newnick>. Messages sent within multi-user chat rooms are of a special type "groupchat" and are addressed to the room itself (room@service), then reflected to all occupants. An occupant exits a room by sending presence of type "unavailable" to its current <room@service/nick>. The coommon system architecture has the next structure(Figure):

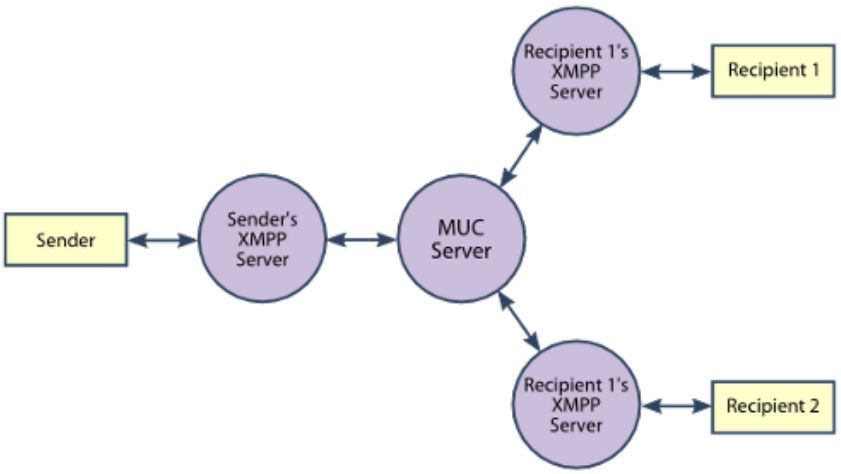

Figure 5.6: MUC System Architecture

Public Speaking

Group chat allows multiple people to gather in the same place to discuss a topic. These virtual meeting places are called rooms, and unlike in the real world, users can be in more than one group chat room at the same time. Rooms can have access controls, moderators, administrators, and even automatic logging and archival of the group's communications.

Group Chat Services

Group chat is provided as a service, usually alongside a regular XMPP server. The group chat service has its own domain; for example, the jabber.org server runs a group chat service at conference.jabber.org. Each room on the group chat service gets its own address, which looks just like a user's JID. The XMPP developer's chat room on the conference.jabber.org service is at jdev@conference.jabber.org, and general XMPP-related chat takes place in jabber@conference.jabber.org.

Entering and Leaving a Room Before you can do much with a group chat room, you must first enter the room. This is also often referred to as joining the room. When you are done participating, you leave the room. Because this mirrors the concept of a user coming and going on and offline, the multi-user chat designers decided to model this part of the protocol with <presence> stanzas. Users can join a group chat room simply by sending

available presence to the room, along with a note that they understand the multi-user chat protocol. Sending presence directly to a JID instead of to the user's server is called directed presence. Similarly, to leave, unavailable presence is sent to the room. Sending presence stanzas directly to a JID instead of to the user's server is called sending directed presence. Directed presence is quite useful in XMPP protocols and extensions because it has some special properties.

If Jane wants to join the group chat room for the Meryton ball, they will both need to send directed presence to their desired identity in the room ball@chat.meryton.lit . Their stanzas are shown in listing 5.7:

```
<presence to="ball@chat.meryton.lit/jane"
          from="jane@longbourn.lit/meryton">
    <x xmlns="http://jabber.org/protocol/muc"/>
</presence>
```

Listing 5.7: Stanzas Format for MUC

Once they have joined the room, the group chat service will broadcast all the other participants' presence statuses to them. After all the other participants' presence stanzas are sent, the server con- cludes the presence broadcast by sending the arriving participant's presence to everyone, including the new arrival. Thus, when a new participant sees their own presence broadcast back to them, they know they have fully joined the room.

The room sends the affiliations and roles of each participant along with their presence. Jane's own presence broadcast also includes a status code of 110, which signals that this presence refers to the user herself. Just as with presence updates from Jane's roster, Jane will also receive presence updates from the room as people leave and new people join on listing 5.8.

```
<presence to="jane@longbourn.lit/meryton"
          from="ball@chat.meryton.lit/jane">
  <x xmlns="http://jabber.org/protocol/muc">
      <item affiliation="member" role="participant"/>
      <status code="110"/>
  </x>
</presence>
```

Listing 5.8: Server Presence Notification

Creating Rooms Thousands of rooms in the federated XMPP network are already available for you to participate in, but sometimes you will find that the room you are looking for does not yet exist. Creating rooms is easy, and it is accomplished in much the same manner as joining a room. Actually, rooms can be created just by joining a non-existent room. Assuming the service allows the user to create new rooms, sending directed presence to the desired room JID of the new room will cause the room to be created and the user to be set as the room's owner. On the listing 5.9, Bingley creates a new room for the Netherfield party:

```
<presence to="chatter@chat.netherfield.lit/bingley"
          from="bingley@netherfield.lit/drawing_room">
    <x xmlns="http://jabber.org/protocol/muc"/>
```

```
</presence>
```

Listing 5.9: MUC Room Creation

The chat.netherfield.lit service responds with the presence broadcast for the room's new and only occupant(listing 5.10):

```
<presence to="bingley@netherfield.lit/drawing_room"
          from="chatter@chat.netherfield.lit/bingley">
    <x xmlns="http://jabber.org/protocol/muc">
        <item affiliation="owner" role="moderator"/>
        <status code="110"/>
        <status code="20"/>
    </x>
</presence>
```

Listing 5.10: Server Respond to Room Creation

Bingley has the owner affiliation and the moderator role. These attributes give Bingley special powers within the room, and you see more about these later. The 110 status code is sent, just as it was before, and a new status code of 201 is sent. This new status code signals that a new room has been created. More comprehensive information about roles, affiliations, erros and etc. can be found in the book[38].

5.4.2 XEP-0060: Publish-Subscribe

Publish-subscribe systems are everywhere-newspapers, blogs, television, and even e-mail lists. There is a channel of communication, subscribers who are interested in data sent on that channel, and publishers who can send data across the channel. The first thing an application must do for a would-be presenter is to create a channel for them to publish sketches. In XMPP pubsub these channels are called *nodes*. The protocol enables XMPP entities to create nodes (topics) at a pubsub service and publish information at those nodes; an event notification (with or without payload) is then broadcasted to all entities that have subscribed to the node. Pubsub therefore adheres to the classic Observer design pattern and can serve as the foundation for a wide variety of applications, including news feeds, content syndication, rich presence, geolocation, workflow systems, network management systems, and any other application that requires event notifications[17].

Creating Node

A pubsub node is created by sending an IQ-set stanza to the pubsub service(listing 5.12):

```
<iq to="pubsub.pemberley.lit"
    from="darcy@pemberley.lit/library"
    type="set"
    id="create1">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
      <create node="latest_books"/>
  </pubsub>
```

[17] XEP-0049: Publish-Subscribe, http://xmpp.org/extensions/xep-0060.html

```
8 </iq>
```

Listing 5.11: PubSub Node Creation

Most actions on pubsub nodes will look very similar to this one. You've seen several stanzas like this one in other chapters already; the difference here is the <pubsub> element. Pubsub nodes and their configuration are necessary and useful, but they don't do much by themselves. The real value of pubsub nodes is in the events that are published to them and broadcast to subscribers. Anything can be included in a pubsub event. The pubsub service doesn't know or care what is inside the event; it simply broadcasts this data to a node's subscribers.

Retrieving Item Elizabeth just subscribed to Mr. Darcy's latest_books node, and she has missed his event broadcasts from earlier in the week. She must investigate his past reading material if she wishes to learn more about him. Remember that Mr. Darcy configured his node to persist items. Anyone can query his node for the most recently published items. Here, Elizabeth requests the last five items by sending an IQ-get stanza to the node with the <items> action(linsting **??**):

```
1 <iq from="elizabeth@longbourn.lit/outside"
2     to="pubsub.pemberley.lit"
3     type="get"
4     id="items1">
5   <pubsub xmlns="http://jabber.org/protocol/pubsub">
6     <items node="latest_books" max_items="3"/>
7   </pubsub>
8 </iq>
```

Listing 5.12: PubSub Node Creation

The <items> element contains a node attribute just like the other actions you've seen previously. Elizabeth has also set the max_items attribute to 3 because she is only interested in the recent history. If she had omitted max_items , the server would interpret it as a request to send all the historical data it has been configured to keep. If she had set max_items to 500, which is much larger than the configured maximum for the node, the server would have sent as many as were available.

Overview

The XMPP publish-subscribe extension provides a framework for a wide variety of applications, including news feeds, content syndication, extended presence, geolocation, avatar management, shared bookmarks, auction and trading systems, workflow systems, network management systems, NNTP gateways, profile management, and any other application that requires event notifications.

This technology uses the classic "publish-subscribe" or "observer" design pattern: a person or application publishes information, and an event notification (with or without payload) is broadcasted to all authorized subscribers. In general, the relationship between the publisher and subscriber is mediated by a service that receives publication requests, broadcasts event notifications to subscribers, and enables privileged entities to manage lists

of people or applications that are authorized to publish or subscribe. The focal point for publication and subscription is a "node" to which publishers send data and from which subscribers receive event notifications. Nodes can also maintain a history of events and provide other services that supplement the pure pubsub model. Figure 5.1

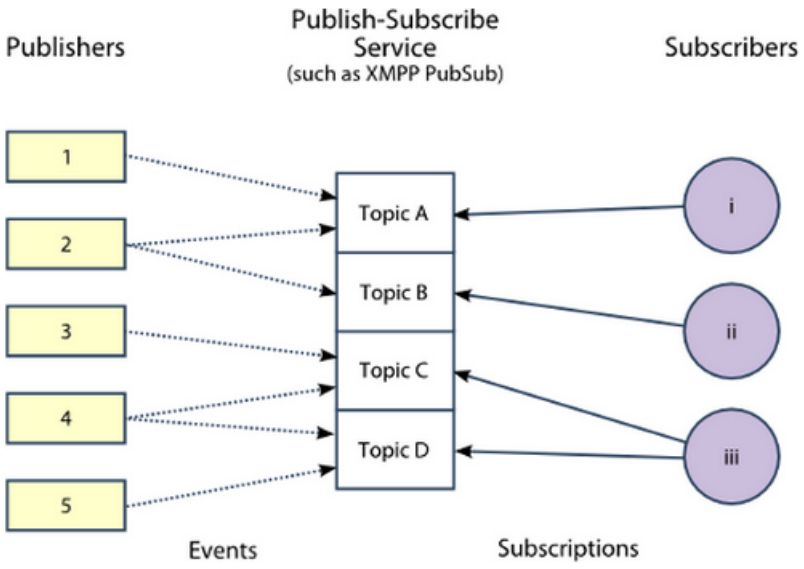

Figure 5.7: General Event Subscription

The XMPP pubsub extension is similarly generic and is usable for a wide variety of purposes. It assumes nothing about the subscribers; they may be human, or they may be machines. Pubsub nodes, unlike multi-user chat rooms, are arranged in a tree-based hierarchy. One of the benefits of the tree shape is that entities can subscribe to non-leaf nodes of the tree, and events published below that node can also be received. Events can be published as notifications or as full payloads, and the subscriber can choose which is most appropriate. Retrieval of the publishing history is built in and fairly fine grained. The subscriber has more fine grained control over the delivery destination. The basic feature set of pubsub is quite easy to implement, and the core mechanics are quite simple to understand.

XMPP Pub-Sub flow

Although the specification of XMPP PubSub is large, here can be described only short main points.

An entity publishes information to a node at a publish-subscribe service. The pubsub service pushes an event notification to all entities that are authorized to learn about the published information. Perhaps the most popular application of pubsub-like functionality is content syndication, which has become familiar from the RSS and Atom (RFC 4287[18]) feeds associated with weblogs, news sites, and other frequently-updated information available on the Internet.

Some example of code screenshot

[18]The Atom Syndication Format, `http://tools.ietf.org/html/rfc4287`

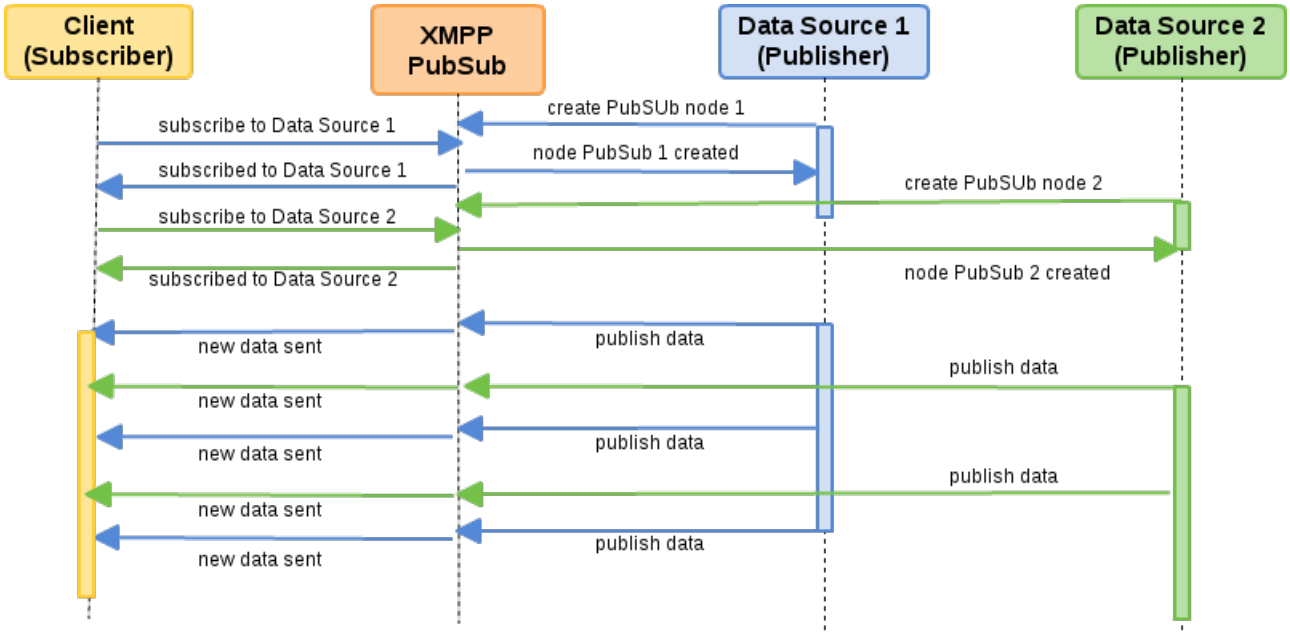

Figure 5.8: Service PubSub

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 on listinf 5.13

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

  <message to="darcy@pemberley.lit"
       from="elizabaeth@longbourn.lit/chatroom"
           type="chat">
      <body>Hello chat room</body>
  </message>

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

Listing 5.13: Stanzas Format

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.

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.

5.4.3 PubSub vs MUC

Pubsub Advantages
Publish-subscribe is a very generic system, used by many different kinds of applications. The XMPP pubsub extension is similarly generic and is usable for a wide variety of purposes. It assumes nothing about the subscribers; they may be human, or they may be machines. Pubsub nodes, unlike multi-user chat rooms, are arranged in a tree-based hierarchy. This shape is often a more close match to a given problem domain. One of the benefits of the tree shape is that entities can subscribe to non-leaf nodes of the tree, and events published below that node can also be received. Events can be published as notifications or as full payloads, and the subscriber can choose which is most appropriate. Retrieval of the publishing history is built in and fairly fine grained. The subscriber has more fine grained control over the delivery destination. The basic feature set of pubsub is quite easy to implement, and the core mechanics are quite simple to understand.

Pubsub Disadvantages
Pubsub, by being so generic, is not optimized for specialized cases. The pubsub extension is not nearly as old or as widely implemented as MUC, and the support for features in both clients and servers varies in quality and depth. Unlike MUC, it is not yet clear what the most used features are, so one must shop around a bit when an advanced feature is needed. There is no special handling of presence built in. There are a few proposed extensions to pubsub that may change this. For example, it would sometimes be useful to limit delivery to available resources only. Tooling for pubsub node creation and configuration is lacking. Tools like Switchboard and Poetry do exist, but aren't fully baked yet. MUC room creation and configuration is built in to most XMPP clients already. Pubsub has not built-in mechanism for subscribers to interact or find each other.

Multi-User Chat Advantages
Presence handling is built in to MUC at a low level. Presence is used to signal joining and leaving of room, and presence changes can also be shared with occupants of the room. MUC is optimized for chat-related use cases and builds on the decades of experience of previous chat systems, especially IRC. All the common moderation and administration features necessary in a collaborative environment are supported - kicking, banning, and various privilege levels. MUC already has many implementations, both of clients and of servers. It is one of the oldest

XMPP extensions, and as such, is quite mature and robust. Occupants in MUC rooms can interact with each other, and MUC allows for multiple levels of anonymity to be used as well as private communication.

Multi-User Chat Disadvantages

Groups of people chatting is the bread and butter of MUC, and MUC is highly optimized for this use case. For example, most MUC servers will automatically send conversation history to every new occupant and generate human-readable messages for most administrative actions. It's possible, and common, to have bots as room occupants, but the experience is designed for human consumption. There is no way to organize chat rooms except as a flat hierarchy, and there is no way to share configurations or participation across collections of rooms. The one exception to this is that most servers have a default configuration that is applied to all rooms on the server. All of these extra human-focused features and administration capabilities make implementation more difficult. Unlike pubsub, MUC implementations have a lot of edge cases to account for in order to be user friendly and robust.

5.4.4 XEP-0049: Private XML Storage

A Jabber client can store any arbitrary XML on the server side by sending an <iq/> stanza of type "set" to the server with a <query/> child scoped by the 'jabber:iq:private' namespace. The <query/> element MAY contain any arbitrary XML fragment as long as the root element of that fragment is scoped by its own namespace. The data can then be retrieved by sending an <iq/> stanza of type "get" with a <query/> child scoped by the 'jabber:iq:private' namespace, which in turn contains a child element scoped by the namespace used for storage of that fragment. Using this method, Jabber entities can store private data on the server and retrieve it whenever necessary. The data stored might be anything, as long as it is valid XML[19]. One typical usage for this namespace is the server-side storage of client-specific preferences;

Methods

get	Sent with a blank query to retrieve the private data from the server.
set	Sent with the private XML data contained inside of a query.
result	Returns the private data from the server.
error	There was an error processing the request. The exact error can be found in the child error element.

Figure 5.9: Description of Acceptable Methods

Elements

The root element of this namespace is query. At least one child element with a proper namespace must be included; otherwise the server must respond with a "Not Acceptable" error.

[19]XEP0049 specification,http://xmpp.org/extensions/xep-0049.html

A client must not query for more than one namespace in a single IQ get request. However, an IQ set or result may contain multiple elements qualified by the same namespace 5.14.

```
CLIENT:
<iq type="set" id="1001">
  <query xmlns="jabber:iq:private">
    <exodus xmlns="exodus:prefs">
      <defaultnick>Alice</defaultnick>
    </exodus>
  </query>
</iq>

SERVER:
<iq type="result"
    from="alice@likepro.co/"
    to="alice@likepro.co/"
    id="1001"/>
```

Listing 5.14: Client Stores Private Data

```
CLIENT:
<iq type="get" id="1001">
  <query xmlns="jabber:iq:private">
    <exodus xmlns="exodus:prefs"/>
  </query>
</iq>

SERVER:
<iq type="result"
    from="alice@likepro.co/"
    to="alice@likepro.co/"
    id="1001">
  <query xmlns="jabber:iq:private">
    <exodus xmlns="exodus:prefs">
      <defaultnick>Alice</defaultnick>
    </exodus>
  </query>
</iq>
```

Listing 5.15: Client Retrieves Private Data

The message format described above can be made by using two main functions: save(for saving data on the XMPP server) and load(to retrieve saved data from the XMPP server), as shown on the listing 5.16

```
save: function (property) {
    xmpp.connection.private.set(property, property + ":ns", user[
        property], function (data) {
            console.log(property + " saved: ", data);
        },
        console.log);
},
load: function (property) {
    xmpp.connection.private.get(property, property + ":ns", function
        (data) {
```

```
9                   user[property] = data != undefined ? data : [];
10                  if (property == 'subscriptions') {
11                      for (var i = 0; i < user.subscriptions.length; i++)
                            {
12                          xmpp.subscribe(user.subscriptions[i]);
13                      }
14                  }
15                  $rootScope.$apply();
16              },
17          }
```

Listing 5.16: Snippet of Save/Load preferences to a private namespace

5.5 Evaluation

Evaluation are done as a proof of concept by demonstrating a scenario of distributed XMPP-driven web site accessing data from a temperature sensor. To the prototype was given a name: SensDash(Sensor Dashboard). Temperature sensor was provided as a testing environment in scope of ACDSee project, together with TU Dresden, BTU Cottbus - Senftenberg and RWTH Aachen University. It locates in the room INF3084, Faculty of Information Technology, Chair of Computer Science. The sensor part represents a class of low-cost, high-performance sensors. It is implemented using a commercially available Raspberry Pi single-board computer with an affiliated USB thermometer and automatic WLAN and XMPP connections to a sensor MUC room established at boot time. Everything concerning sensor are installed on Mobilis server. According to the system architecture on the Figure 5.9, everything concerning sensor data are assumed is a DataHub. Which has pre-installed XMPP server which has extensions Multi-User Chat (XEP0045) MUC rooms each carry a description field which qualifies them as sensor MUC rooms. The description field is also already standardized in XMPP. The temperature sensor itself is a Data Source 1 on the Figure 5.9.

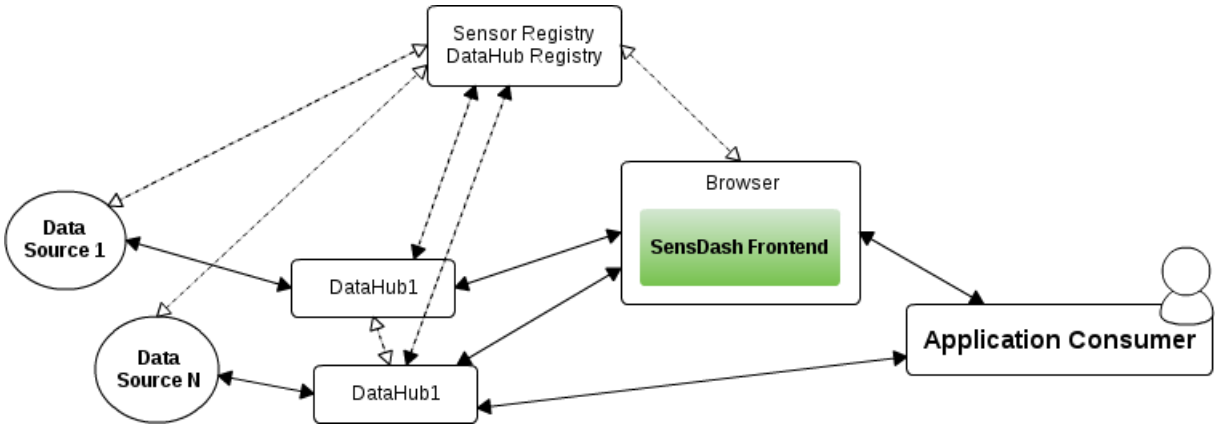

Figure 5.10: System Architecture Scheme

The temperature sensor as a MUC room,is a Data Source 1 on the Figure 5.9. And has the next JSON description(listing 5.17):

```
{
    "sensormuc": {
        "type": "AMBIENT_TEMPERATURE",
        "format": "short",
        "location": {
            "countryCode": "DE",
            "cityName": "Cottbus",
            "latitude": 51.076834,
            "longitude": 13.772586
        }
    }
}
```

Listing 5.17: JSON Description Format

Firstly it has to be registered in the Sensor Registry with a unique id and fulfill the attributes defined in JSON Registry standard accordingly. Since in scope of Master Thesis neither in scope of ACDSee project not included to implement automative Registry, where sensor have to be registered, metadata describing temperature sensor was added manually. As shown on the listing 5.18

```
  {   "id": "2",
        "title": "Ambient Temperature INF3084",
        "availability": true,
        "last_update": "2013-09-18T18:31:38+01:00",
        "description": "This sensor monitors and ensures perfect working
           conditions for students: 24-26 Celcius Degree. Whatever the
           weather is outside, student lab is always nice an cozy.",
        "sla": "Measurements taken every 10 seconds. Uptime 95% from 6:00 till
            22:00.",
        "icon": "img/icon/temper_outside.png",
        "picture": "img/sensor_images/temper.jpg",
        "access": "private",
        "provider_name": "Provider TU Dresden",
        "location": "Dresden",
        "provider_www": "http://www.inf.tu-dresden.de/",
        "preview": "",
        "reliability": "",
        "trust_level": "",
        "type": "chart",
        "dev_details": "true",
        "responsible_team": "RN",
        "administrator": "Philipp Grubitzsch",
        "EPConfig": "config.pdf",
        "end_points": [
            {
                "type": "muc",
                "name": "chat1@conference.likepro.co",
                "pwd": null
            },
            {
                "type": "pubsub",
                "name": "chat2@conference.likepro.co",
```

```
                "pwd": null
            },
            {
                "type": "muc",
                "name": "chat3@conference.likepro.co",
                "pwd": null
            }
        ],
        "template": {
            "subtitle": {
                "text": "Real-timeupdated"
            },
            "yAxis": {
                "title": {
                    "text": "Temperature"
                }
            },
            "series": [
                {
                    "data": [],
                    "name": "Ambient Temperature INF3084"
                }
            ],
            "title": {
                "text": "Ambient Temperature INF3084"
            },
            "chart": {
                "type": "spline"
            },
            "xAxis": {
                "labels": {
                    "rotation": -45
                },
                "type": "datetime"
            },
            "credits": {
                "enabled": false
            }
        }
    }
```

Listing 5.18: JSON Description Format

Demonstartion of activity scenario was made by using 3d party application consumer requirements. His name Max and has to create specific mobile application and define which sensor he needs to be there and retrieve an info from it.

Step 1: In order to find necessary sensor, explore description and data provided by it, Max has to log in into the SensDash by using peronal JID and password, received from an dministrator of a SensDash(Figure 5.10).

Step 2: Once Max, finds necessary sensor, by using search field, to get more detailed information about sensor he clicks on the field of sensor and appear popup, but to be precise, it is a new tool, called modal(Figure 5.11). In the appeared modal, user can get a detailed

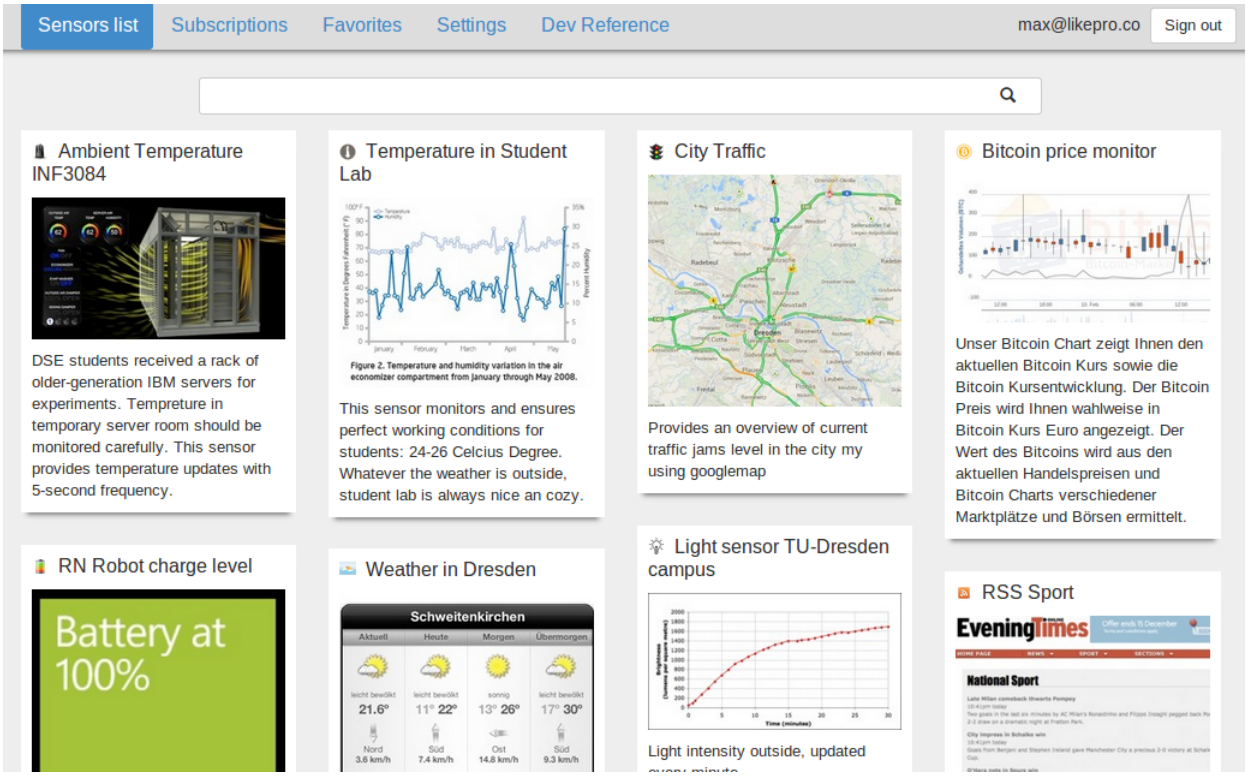

Figure 5.11: Log in to the SensDash

description about sensor: SLA, location, provider, preview, development details such as end-points quantity, end-point configuration, API references. This information gives an overview of what type of data provided by data source. In case of ambient temperature INF3084, Max can explore predefined preview, which consists graph of provided temperature, data values and it's frequency, security level and relialability.

Step 3: If Max accepts SLA(if it is exist for such a sensor) and wants to get real streaming info from sensor he needs to subscribe to it. When it is done, subscribed sensor appear in a list of Subscriptions Tab. And as soon as new data become available SensDash retrieve it in this Tab(Figure 5.12). Also, by using Subscriptions Tab and icon "star" in a right corner, become possible to add subscribed sensor to the Favorites. After clicking on the icon of favorites, sensor information will appear in a list of favorites in Favorites Tab.

Step 4: To get info about personal account Max have to use Settings Tab. Where exists personal profile settings.

Step 5: As a developer, Max might be interested in technical details of sensor data retieval process, e.g. API references, End-point configuration, sensor data format, interface and system architecture in order to interconnect with system itself. Firstly Max has to go to the Admin References Tab(Figure 5.14), and follow all necessary steps described in this Tab.

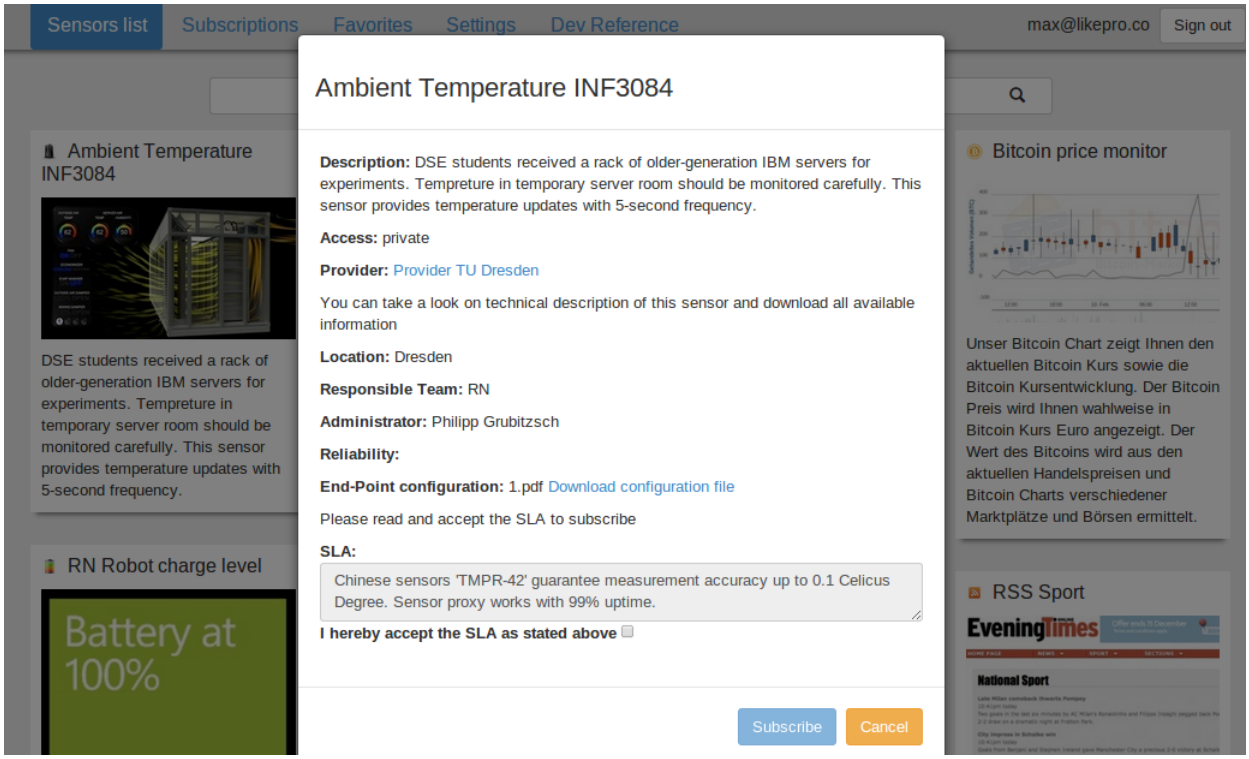

Figure 5.12: Personal Modal of a Sensor

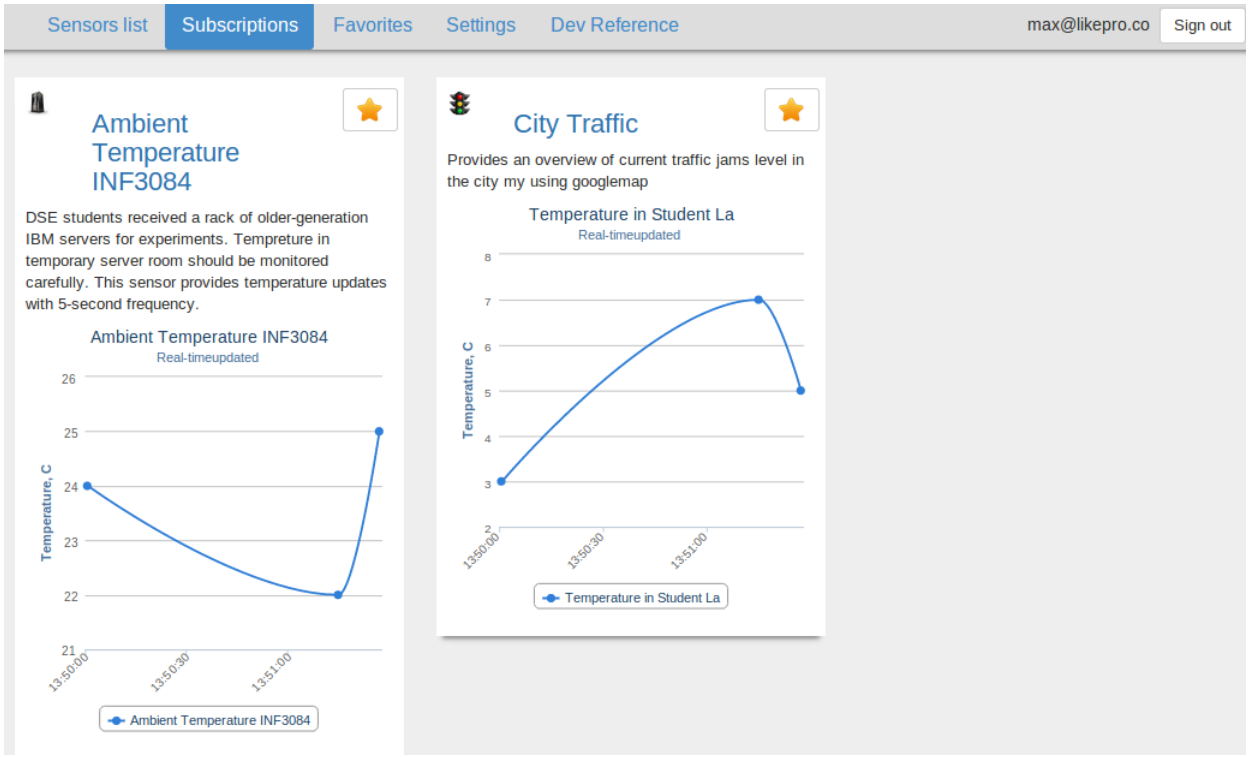

Figure 5.13: Subscriptions list

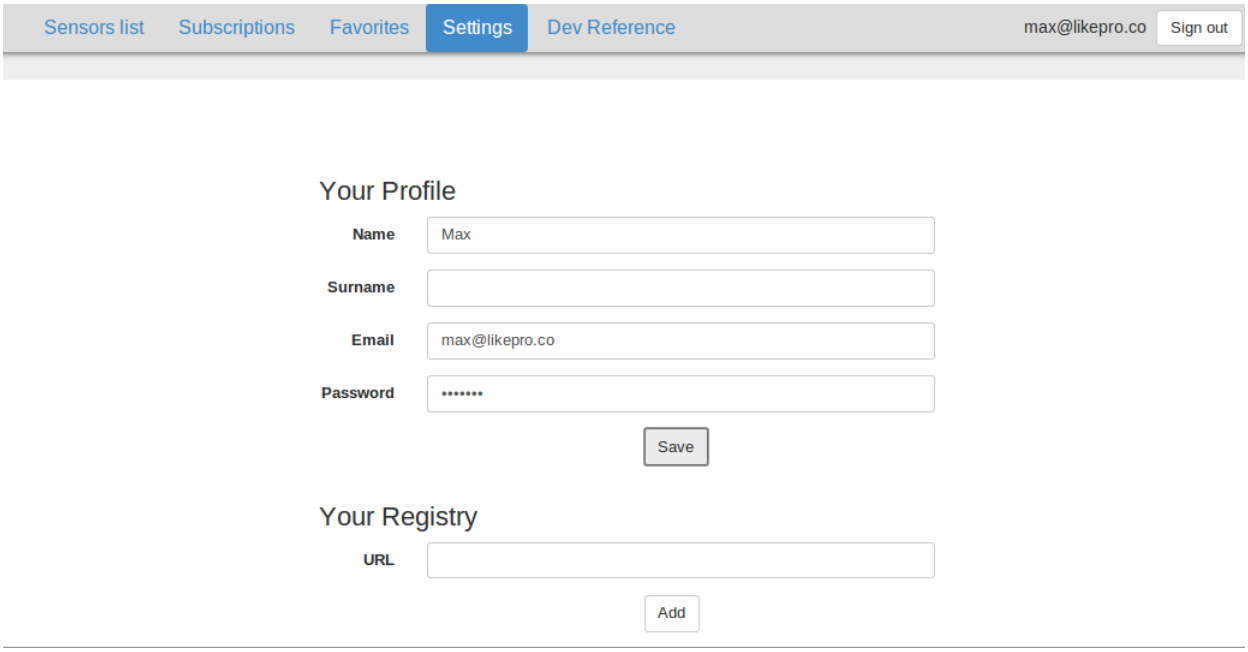

Figure 5.14: Settings Tab

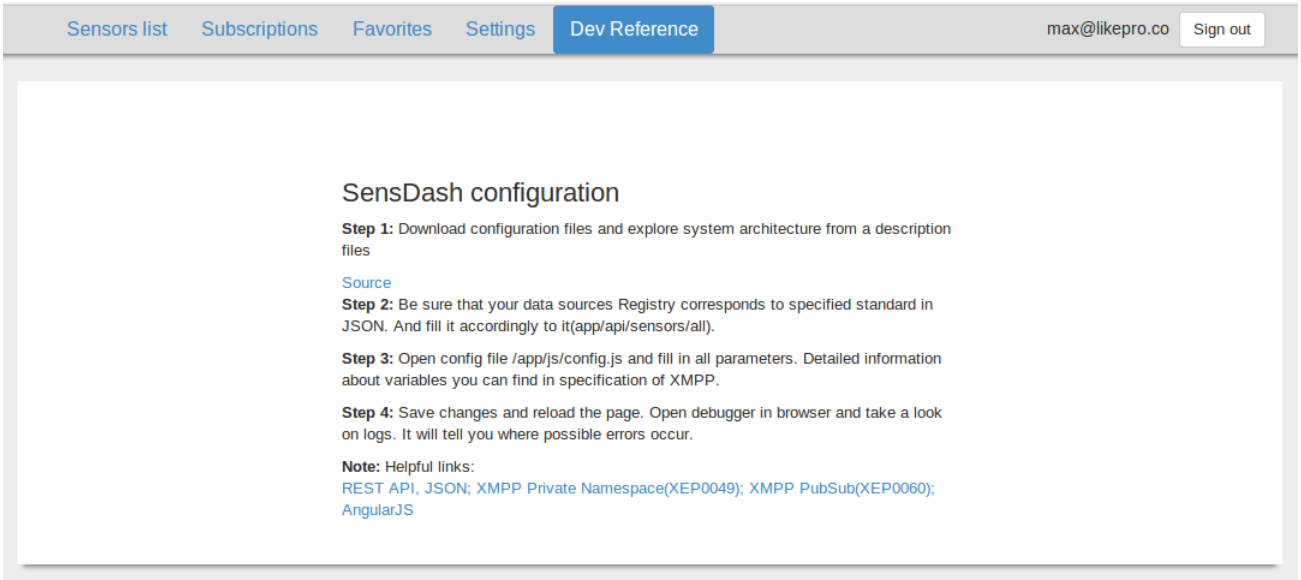

Figure 5.15: Administrator Tab

5.5.1 SensDash Implementation

After real example of SensDash usage was described in the previous section, needs to be clarified the concept overflow together with used and implemented XMPP extensions.

Log Into the system can be possible only with valid JID and password, which have to be registered on a XMPP server. Firstly, user credentials validated by input type on Frontend side and only than sended to the Backend. Once XMPP server accepts credentials, using cookies JID and password are saved in browser. So the next time when user will open the browser URL of a SensDash, cookies will automatically log in the user to the system.

Preferences Saving are done by using XEP0049. When a user logged in first time to the system, application logic creates fully empty account with no subscriptions and favorites. Once a user subscribes to any resource, this resource automatically appear on Subscriptions Tab. The same for favorites and Favorite Tab. It means that system has saved this preferences on XMPP server by using one of the extension called XEP0049, based on private spaces for every JID without interconnection with Backend. Next time when user log in to the system, all saved subscriptions and favorites will be loaded in the meanwhile and appear in corresponding Tabs.

Search Bar is made by using only Frontend, as one of the feature of the AngularJS. It makes sorting of an graphical object through the all existed titles of a data source. It is fast and very straightforward in usage. The code, responsible for realization this functionality are shown on the Figure.

Data Streaming is the point where DataHub as part of the Backend come into a picture. All data streaming works through XMPP extension called XEP0049(MUC) or XEP0060(PubSub). In presented example with ambient temprature in INF3084 was used MUC. Since generic frontend supposed to support all possible approaches, it also provides realization for MUC. By using Strophe.js for client-side was made refactoring according to skeleton defined by AngularJS.
The SensDash logic builded on top of AngularJS skeleton. By using two-way data binding, aggregation and presentation of a sensor metadatadata was done by using factory pattern. Together with creation of a sensor entity as a graphical object, all handlers, functionality and interconnection protocol also created for every object accrodnigly, based on abstract methods. Such a general manipulation of objects makes possible to loosely couple code and object. All sensors metadata will be automatically parsed from Sensor Registry and presented on a SensDash always in the same way. Everything done in runtime, such that if a new sensor registers itself in Sensor Registry it will appear as soon as user reload the page.
Functional characteristic of a sensor
In the section 4.4 was clarified 3 main characteristics which acquire sensor based on a system architecture. Since realization of a DataHub fully rely on a backend, but in proposed evaluation was used XMPP server, become possible to differentiate security, reliability and performance level based only on server configuration. Where every end-point is a MUC chat room(Figure).

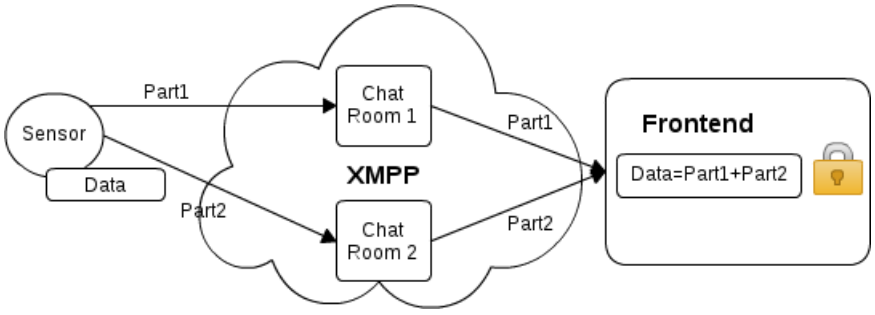

Figure 5.16: Data secure transfer using Chat Rooms

5.5.2 Summary

This chapter presents details of implementation of the generic frontend concept. At first, choosen tools and development environment were presented: jQuery, HTML5, CSS as a programming language, AngularJS together with Bootstrap to interconnect application logic with XMPP interface standard and Strophe.js was picked to implement the XMPP mechanism and it's extensions. Web API was defined and afterwards used to retrieve from the Registry all available sensors by sending direct HTTP GET request. Authentification and data streaming from a web browser and XMPP server was made by using XMPP BOSH standard and XEP0049, XEP0045 extensions, based on Strophe.js.

All used technologies, protocols, libraries and methodologies was gathered together in order to realize working prototype. A summary overview of all described above compunents and tools are shown on the Figure 5.19 Was presented the convinsing scenario based on ambi-

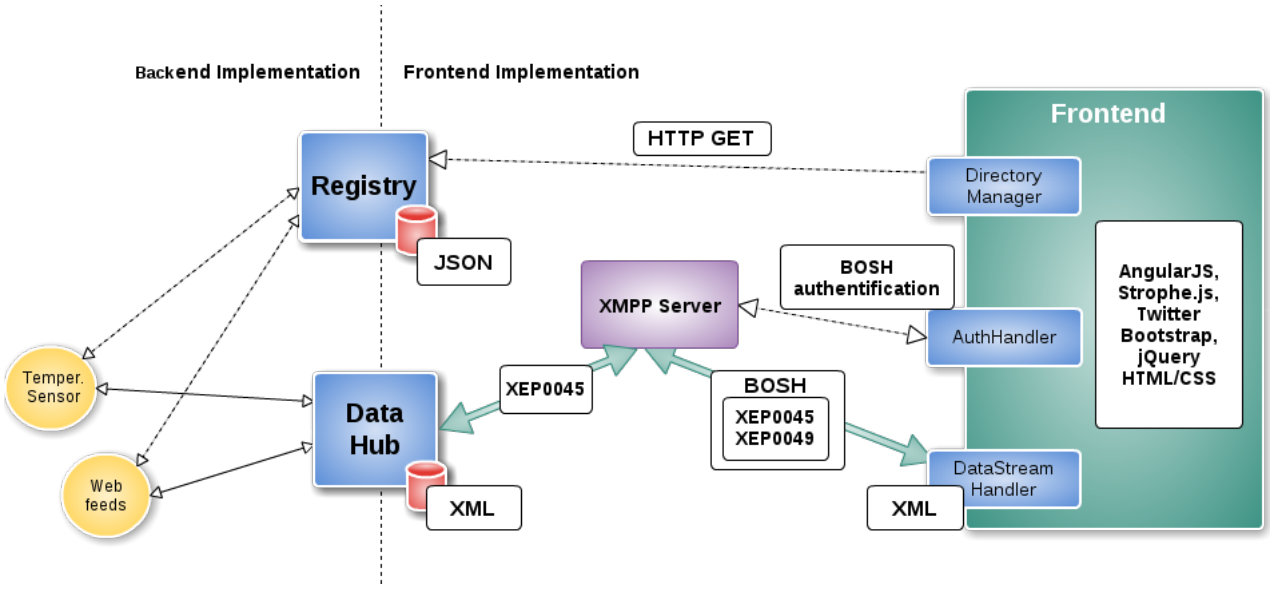

Figure 5.17: Implementation Architecture

ent temperature sensor from room INF3084 and consumer application developer's requirements.

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] Schuster Daniel, Ronny Klauck, Michael Kirsche, Renzel Dominik, and Istv an Koren. Federated access to smart objects using xmpp. 2013.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

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

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

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

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

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

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

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

[37] Wikipedia. Multitier architecture — wikipedia, the free encyclopedia, 2013. [Online; accessed 1-December-2013].

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

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

[40] 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.

[41] Wikipedia. Angularjs — wikipedia, the free encyclopedia, 2014. [Online; accessed 19-February-2014].