

FINAL REPORT

A Sensor Web Approach to Geo Sensing

Degree Program:
Master Information Technology & Systems Management

Submitted by:
Tanja Malitz



Head of Faculty: FH-Prof. DI Dr. Gerhard Jöchtl
Supervisor: DI (FH) Thomas Lampoltshammer, MSc

Salzburg, August 2013

Affidavit

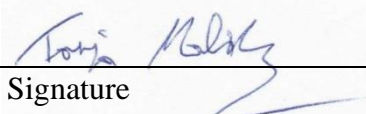
Herewith I, Tanja Malitz, declare that I have written the present thesis fully on my own and that I have not used any other sources apart from those given.

Passages that I have adopted, whether in sense or literally from other published or non-published works have been marked as such.

The present thesis has, in the same or in similar form, never been handed in to another board of examination.

16th September 2013

Date



Signature

Acknowledgements

I want to thank the international office of the Salzburg University of Applied Sciences. Without their help and getting the Marshall-Plan scholarship it would not have been possible to write this thesis in Honolulu, Hawaii.

Secondly, a big “Thank You” to the Hawaii Pacific University for their help with the infrastructure. I got a great office to work and always help from the teachers around. Special thanks to Mr. Curt Powley, Ph.D., chair of the department of Mathematics and Computer Science at HPU.

Thank you to Samuel Joseph, my supervisor from HPU. He is living in the UK, but nevertheless, he was always available for questions. And thanks for the great idea with writing a blog of my progress. I did not like the idea at first, but in the end, it was really useful.

Another “Thanks” to Thomas Lampoltshammer, my supervisor from Salzburg. It is challenging to supervise if you are located over 7,000 miles away from each other with a time difference of 12 hours.

Moreover, I want to thank the many new friends I made during that semester. It was not always easy to say “I have to study” instead of going to the beach.

But most of all, thank you to my family and friends at home. It is not easy to leave your home country, fly 25 hours and live for nearly 6 months abroad. Without your support this would not have been possible.

Thank you!



Details

First Name, Surname:	Tanja Malitz, BSc
University:	Salzburg University of Applied Sciences
Degree Program:	Master Information Technology & Systems Management
Title of Thesis:	A Sensor Web Approach to Geo Sensing
Academic Supervisor:	DI (FH) Thomas Lampoltshammer, MSc

Abstract

Geo-sensing is a topic that has become more and more important in the past few years. The wireless technology is by now reaching remote areas and the technology for sensor devices, software and network protocols got improved. Therefore, new efficient methods for querying, processing, mining and analyzing environmental real time data are required. There still exists a lack in that area.

If geo-sensor data can be analyzed and processed more effectively, new insights in the particular area of research can be gained. Thus, this thesis sets a special focus on the combination of existing data streams to new aggregations with the help of version control systems. It tries to find out if there are appropriate ways for handling near real time geo-sensor data with existing version control systems instead of using databases. Version control systems offer simple ways to store branches of data, to compare data with each other and to track changes over time.

For achieving this goal, the thesis analyses the state of the art regarding sensor web platforms. It moves on by evaluating existing version control systems. Out of that a prototypical implementation is developed. The prototype is able to combine data streams to new constructs, store constructs for later use and it offers a graphical representation of current data.

The outcome of the thesis is a comparison between using version control systems or databases for handling real time geo-sensor data. Moreover, various advantages and disadvantages between the existing version control systems are described. In addition, different visualization methods are applied for representing data stored in version control systems.

Table of Contents

1. Introduction and motivation	12
1.1. Methodology	12
1.2. Research Question	13
1.3. Ambition and Structure.....	13
2. Geo-Sensing in a general context.....	15
2.1. History of geo-sensing	15
2.2. Internet of Things	15
2.3. Geo-Sensor Networks	16
2.4. Fields of application	17
3. Availability of Sensor Data	19
3.1. Computing platforms	19
3.2. Software	21
3.3. Network protocols	21
3.4. Sensors	22
3.5. Data standardization initiatives.....	22
3.5.1. INSPIRE	22
3.5.2. Global Earth Observation System of Systems (GEOSS).....	23
3.5.3. Copernicus	23
3.5.4. Shared Environmental Information System (SEIS).....	23
3.5.5. Global Spatial Data Infrastructure Association (GSDI)	24
3.6. Data Collection	24
4. Standardized interfaces and exchange protocols.....	26
4.1. Initiatives for Standardization.....	26
4.2. Benefits of standards.....	29
5. Version Control Systems.....	30

5.1. Centralized version control systems	30
5.1.1. Advantages.....	31
5.1.2. Disadvantages	31
5.2. Distributed version control systems.....	32
5.2.1. Advantages.....	32
5.2.2. Disadvantages	32
5.3. Evaluation and comparison of existing version control systems.....	33
5.3.1. Apache Subversion (SVN).....	34
5.3.2. Mercurial.....	35
5.3.3. Git	35
5.4. Selection of two suitable systems for implementation	36
6. Visualization of Sensor Data.....	37
6.1. Terminology	37
6.2. Visualization Taxonomy.....	37
6.3. GIS (Geographic Information Systems)	39
6.3.1. Autodesk	40
6.3.2. ESRI (Environmental Systems Research Institute)	40
6.3.3. GRASS GIS	41
6.3.4. Google Earth	41
6.4. Sensor Networks and GIS.....	41
6.4.1. Sense Web project.....	41
6.4.2. PermaSensorGIS	42
6.4.3. City Sense	42
7. Perspectives and Challenges	43
7.1. Data.....	44
7.2. Hardware.....	45
7.3. Software.....	45
7.4. Sensor Networks	45

7.5. Issues and Problems	46
7.5.1. Privacy	46
7.5.2. Data Ownership and Pricing	47
7.5.3. New Focus	47
8. Requirements for a Prototype.....	48
8.1. General Description	48
8.1.1. Product Perspective.....	48
8.1.2. General Functions	48
8.1.3. User Characteristics	50
8.1.4. Restrictions	50
8.1.5. Assumptions and Dependencies	50
8.2. Specific Requirements	50
8.2.1. Non-functional requirements	50
8.2.2. Actors	51
8.3. Data Model	51
8.3.1. Use Cases	51
8.3.2. Sequence-Diagrams	56
8.4. Test Data.....	58
8.5. Online repositories.....	59
8.6. Programming language.....	59
8.7. Simulation of a Geo-Sensor.....	60
9. Actual Implementation with Subversion and Git.....	63
9.1. Start Screen.....	63
9.2. Adding a geo-sensor	64
9.3. Editing a sensor.....	64
9.4. Linking sensors or constructs	65
9.5. Visualization of data.....	67
9.6. Settings	71

9.6.1. Subversion.....	72
9.6.2. Git	73
10. Results	75
10.1. Comparison of Subversion and Git.....	75
10.2. Comparison between VCS and databases.....	76
11. Conclusion.....	78

List of Figures

Figure 1 - Examples of Geo-Sensors [1]	15
Figure 2 - Overlaps of the Internet of Things with other Research Areas [4].....	16
Figure 3 - Processing of Sensor Data [5]	17
Figure 4 - TMote-Sky Components	20
Figure 5 - Real-time Deadlines	25
Figure 6 - Standards [11].....	26
Figure 7 - A Standardized Sensor Network [14].....	28
Figure 8 - Centralized Version Control System [17]	31
Figure 9 - Distributed version control systems [17].....	32
Figure 10 - Overview: Version Control Systems	34
Figure 11- Taxonomy of Buja [19]	38
Figure 12 - Future Directions [26]	44
Figure 13 - Primary Use Cases.....	52
Figure 14 - Sequence Diagram: Add Sensor.....	56
Figure 15 - Sequence Diagram: Delete Sensor	57
Figure 16 - Sequence Diagram: Add Meta Construct	57
Figure 17 - Sequence Diagram: View Data	58
Figure 18 - Test Data File Example	59
Figure 19 - Simulation.....	61
Figure 20 - Tab control "SVN"	61
Figure 21 - Tab Control "Git"	62
Figure 22 - Start Screen.....	63
Figure 23 - GUI of adding a sensor.....	64
Figure 24 - GUI of editing a sensor.....	64
Figure 25 - GUI for linking data	65
Figure 26 - New constructs	66
Figure 27 - Proxy.....	66
Figure 28 - GUI for data visualization	67
Figure 29 - GUI 2 for data visualization	67
Figure 30 - Curve	68
Figure 31 - Visualization: Pie Chart.....	69
Figure 32 - Visualization: Bars	69

Figure 33 - GUI for saving visualizations	70
Figure 34 - GUI for loading visualizations	71
Figure 35 - Alert box for unique key violations	71
Figure 36 - Settings	72
Figure 37 - Start Screen VCS	72
Figure 38 - Service Subversion	73
Figure 39 - Workflow of linking sensors	73
Figure 40 - Service Git	74
Figure 41 - Show Log	77

List of Tables

Table 1 - Use Case 1: Add sensor	53
Table 2 - Use Case 2: Delete sensor.....	54
Table 3 - Use Case 3: Generate Meta Construct	55
Table 4 - Use Case 4: View Meta construct.....	56
Table 5 - Test Data [29]	58
Table 6 - Comparison Subversion and Git	75

1. Introduction and motivation

In recent times there have been several trends in the field of geo-sciences, which make the development of suitable sensor web platforms more and more important. For example, the wired information structure is now reaching remote areas, even those that may not have access to power supply. There has also been progress regarding power consumption, sensor materials and miniaturization of the devices [1].

These innovations bring a lot of new possibilities for environmental monitoring. Examples are bio-chemical sensors for air-pollution monitoring, vibration and sound sensors for volcano monitoring and even sensors for watching the growth of orchards. Not only are terrestrial ecology sensor systems possible; the new generation of smartphones can also be used for gathering geo-data. Users can act like sensors for monitoring environmental changes such as actual weather conditions or the occupancy of a car park. A huge advantage in this direction is the spread of smartphones all over the world [2].

Sensor web platforms are necessary to handle the huge amounts of new data being generated. Analysts should be able to filter out only the information of interest to them. Therefore, sensor data should be organized so that it is available for specific application development. There are already existing standardized exchange protocols and interfaces, however there is a gap relating to data management. The querying, processing, mining and analyzing of real-time data streams should be made more convenient for sensor web platform development. The aim of this thesis is therefore the evaluation of possible methods and a prototypical software implementation using version control systems for reaching that goal.

1.1. Methodology

The methodology of this thesis is to survey the state of the art on the topic of geo-sensing. In addition the thesis tries to optimize the analyzing of real-time data streams. To this end, a prototype which uses version control systems is developed.

The goal of the thesis is to compare which version control system is best suited for this task. There is also made a comparison to existing GIS (Geo Information Systems) which mainly use databases for data storage.

1.2. Research Question

This paper tries to answer the following research questions:

- How can data management concerning analyzing near real-time data streams be improved?
- Are there appropriate ways for handling real time geo-sensor data with existing version control systems?
- Which versioning system is the most suitable for handling data?
- Are version control systems a good alternative for storing geo-sensor data instead of databases?
- What are suitable techniques for displaying real time data streams for analysis?

1.3. Ambition and Structure

The paper starts with a general overview of the topic geo-sensing. Terms like “Internet of Things” or “Geo-Sensor networks” are introduced. It also provides an overview of common fields of application.

Furthermore, this thesis deals with the availability of sensor data. That includes available computing platforms, software, network protocols and sensors. It also investigates collection methods for data and places a special focus on real-time data.

Standardized interfaces and exchange protocols are reviewed. The paper introduces initiatives for standardization and mentions benefits of standards.

The practical part of this thesis deals with a prototypical implementation for improving the processing of sensor data. To support this, an overview of version control systems is included and also a comparison of existing solutions.

Moreover, the prototypical part includes some options for visualization. Therefore a general overview of visualization methods is given. Geo Information Systems (GIS) are introduced.

The theoretical part of this thesis ends with an overview of current perspectives and challenges in the field of geo-sensors. Following from that, the next chapter deals with requirements for a prototypical implementation. It gives a general overview of the developed prototype, the data model, test data used and the technologies used to support development.

The next chapter of the thesis introduces the finished prototype in more detail. The GUI and the functionality are also described. The differences between the used version control systems are highlighted to support a comparison between those systems. Finally a closer look is taken at the differences between version control systems and databases.

2. Geo-Sensing in a general context

This chapter provides a general overview of geo-sensing. It tries to answer how geo-information technology developed, the connection to the catchphrase “Internet of Things” and what geo-sensor networks are. Moreover it introduces common fields of application.

2.1. History of geo-sensing

Historically geomaticians were known as land surveyors. Land surveying makes use of a great deal of mathematics and physics. The German mathematician Carl Friedrich Gauss (1777-1855) spent about 20 years of his life establishing a geodetic coordinate system. The use of photogrammetric techniques for collecting geo-data has been going on since the 19th century. Geo-data is the most important prerequisite for conducting research in the field of geoscience and for achieving a detailed understanding of Earth-related processes [3].

Today geo-information technology is becoming increasingly widespread. Data are mostly collected via airborne and orbiting sensors using photogrammetric techniques.



Figure 1 - Examples of Geo-Sensors [1]

Geo-sensor networks have created a lot of new opportunities for collecting a great variety of geo-data. Figure 1 shows some examples of current geo-sensor devices.

2.2. Internet of Things

Geo-sensor devices belong to the “Internet of Things”; a catchphrase for technologies that makes it possible to connect the physical world (things like sensors) to the Internet¹.

¹ Contiki project, <http://www.contiki-os.org> (August 2013)

Figure 2 shows an overview of which fields of research are related to the “Internet of Things”.

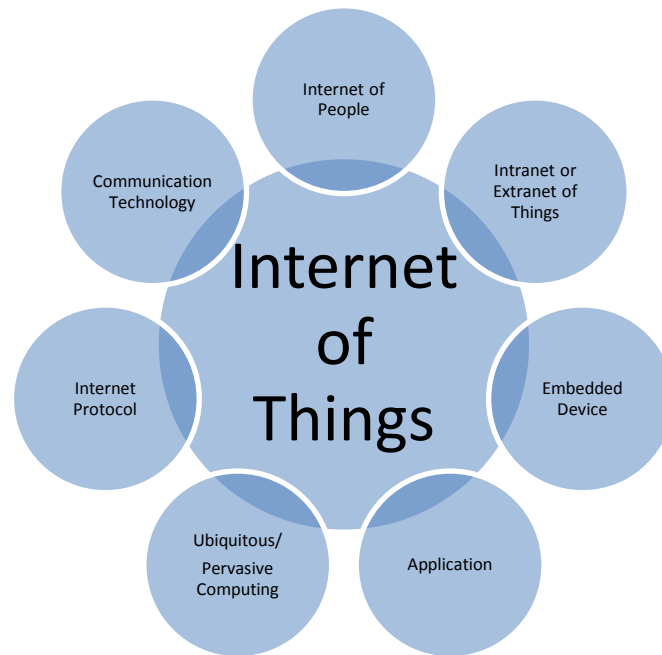


Figure 2 - Overlaps of the Internet of Things with other Research Areas [4]

“Ubiquitous spatial computing” is a related term that means that computation is moved to everyday devices through embedded technology and always-on connectivity [5].

2.3. Geo-Sensor Networks

Geo-sensor networks are a combination of small sensors and tiny computers [5]. They are sensor enabled small devices and can be distributed throughout a geographic environment [6]. Thus geo-sensor networks can be defined as networks that monitor phenomena in a geographic space [7].

If geo-sensor networks are using real-time data they can be considered as a sort of “environmental microscope” [5].

The nodes of those networks have the following tasks:

- Production of sensor data streams
- Processing of data streams locally
- Processing of aggregated data (for the minimization of the communication)

The nodes can relay information, but they can also process it locally. Therefore the nodes are reusable, re-programmable systems [5].

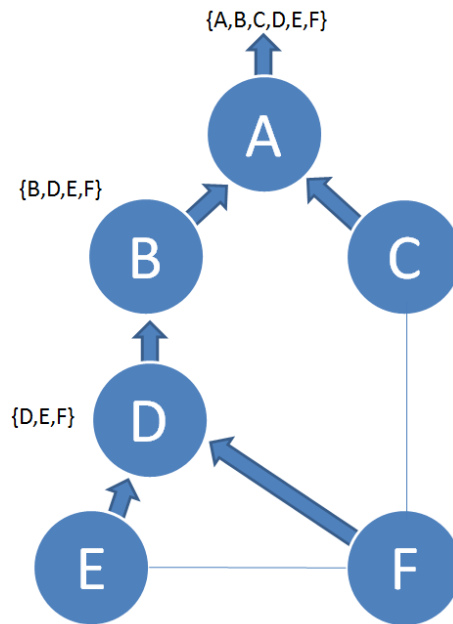


Figure 3 - Processing of Sensor Data [5]

Figure 3 shows an example how data processing in sensor networks can work.

2.4. Fields of application

Geo-sensor networks open a wide range of new possibilities such as [5]:

- Coastal monitoring and ocean exploration
 - Mapping ocean floor
 - Coastal monitoring
- Drought management
- Forest fire detection
- Precision agriculture
- Habitat monitoring

In general, three application types can be distinguished based on their observation characteristics [1]:

- terrestrial ecology observing systems
- geological observation systems
- aquatic observation systems

The following paragraphs will explain each of these systems in more detail.

Continuous monitoring is common in systems that observe terrestrial ecologies like the observation of the growth or the health of plants. In Australia from 2006 to 2008 a project monitored the growth circumstances of a nectarine orchard. The orchard has been covered with about 270 sensors and a gateway connected to the internet [1].

Geological observation systems describe real-time detection applications like a volcano sensor network deployment. For example, the volcano Mount Pinatubo on the island of Luzon in the Philippines erupted on June 15th 1991 after about 600 years of dormancy. Scientists were interested in monitoring the mud flow which is a kind of dynamic spatial field. With the help of geo-sensor networks they can find out if one of the major tributaries has split or if the mud flow is still expanding [6].

The group of aquatic observing systems includes geo-sensor systems that are mobile or attached to mobile objects. Mobile objects include things such as cars, animals and ocean buoys. Also in this group are tsunami early warning systems or coastal and ocean observation systems [1].

3. Availability of Sensor Data

This chapter introduces methods to collect sensor data. Available computing platforms, software, network protocols and sensors for that purpose are presented. There also exist data standardization initiatives that try to make it easier to share data between different institutions. The chapter ends with presenting possible types of data collection with an emphasis to real-time data collection.

In 1999 Neil Gross expressed the following vision:

“In the next century, planet earth will don an electronic skin. It will use the Internet as a scaffold to support and transmit its sensations. This skin is already being stitched together. It consists of millions of embedded electronic measuring devices: thermostats, pressure gauges, pollution detectors, cameras, microphones, glucose sensors, EKGs and electroencephalographs. These will probe and monitor cities and endangered species, the atmosphere, our ships, highways and fleets of trucks, our conversations, our bodies - even our dreams.” [9, p. 1]

Gross’ vision looks close to being realized. As matter of fact, there have been several research efforts since the 1990s towards the design of tiny computing platforms as well as on appropriate operating systems that run on these platforms. There have been huge improvements regarding miniature, low-cost microelectronic and mechanical systems. These systems have limited on board processing capabilities, limited storage and short-range wireless connections [1].

3.1. Computing platforms

The goal for miniature computing platforms is to reduce them to the size of sand grains. This will enable development of a sensor network that consists of thousands or even millions of sensors that are coated with micro-sensors as small as a 1,000th millimeter. This would create an “environmental microscopic” view of geophysical phenomena [1].

The following paragraphs introduce some of the commercially available computing platforms Mica Mote series, Dust networks and TMote Sky.

Mica motes are commercially available from a company called Crossbow. Motes are also known as smart dust and wireless sensing networks. A Mica mote can work up to one year powered by two AA batteries [9].

Dust networks offers several ways to connect smart devices. The company's portfolio consists of wireless embedded products with advanced network management, security features and ultra-low power consumption for wire-free operations².

Figure 4 shows a picture of a TMote Sky computing platform.

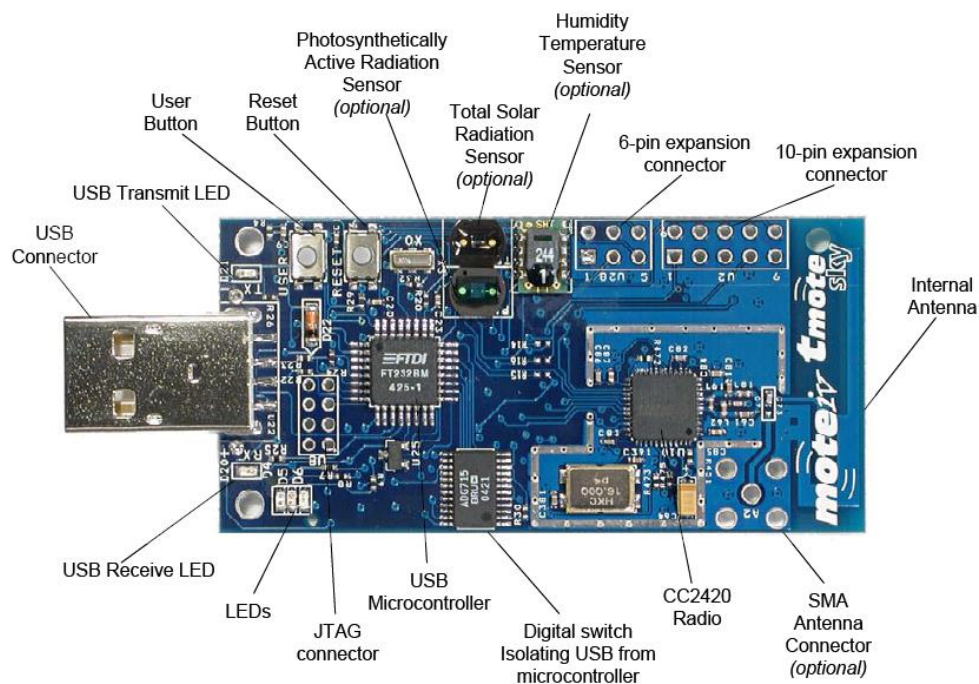


Figure 4 - TMote-Sky Components³

TMote Sky has been programmed by three students from the University of California, Berkeley who founded the Moteiv Corporation which has then been acquired by Sentilla. TMote Sky is a mote for sensor network applications with the goal of low power operation and long term deployment⁴.

² Linear Technology, <http://www.linear.com> (August 2013)

³ IPFW, www.etc.ipfw.edu (August 2013)

⁴ VB Profiles, <http://venturebeatprofiles.com> (August 2013)

3.2. Software

Because of the decreased size of the computing platforms it is also necessary to rethink and evolve the supporting software. TinyOS and Contiki are examples for operating systems that have been developed especially for wireless sensor networks.

TinyOS is an event-based operating environment. It needs only a few kilobytes of code to store the entire operating system and only a few hundred bytes of RAM to run it. It is available as open source [1]. TinyOS is based on the programming language NesC (network embedded systems). NesC is an extension to the programming language C. It has been designed to support the concepts and the execution model of TinyOS [10].

Contiki is another open source operating system. It offers communication for tiny, battery-operated, low-power systems with the internet. It can be used for city sound monitoring, streetlights, industrial monitoring or alarm systems⁵.

3.3. Network protocols

The network connection between sensor nodes has also undergone improvements. There is a need for low-power, robust and ad-hoc communication protocols between sensor nodes. Normally, a node has a reliable communication range between 10 to 100 meters. Therefore messages have to be sent in a multi-hop fashion from sensor to sensor until they reach their destination. Network protocols are required to help route messages from a node to its destination reliably using the least amount of energy. Sensor networks are resource-constrained, so that data collection, message routing and the coordination between nodes have to be integrated on one chip. As a consequence, these networks are difficult to program, debug and test [1].

Although there has been a lot of progress and the research domain is in an active state, experience with small-scale geo-sensor networks is still limited. Many applications are still prototypes and sensor nodes are often rather match-boxed than sub-millimeter sized. In the future, ideal geo-sensor networks will be able to work without wires for power or communication. This would make it easier to deploy them in remote areas [1].

⁵ Contiki Project, <http://www.contiki-os.org> (August 2013)

3.4. Sensors

New sensors have been developed using semiconductor fabrication technologies. MEMS (micro-electro-mechanical systems) sensors consist of components between 1 to 100 micrometers in size. They are made from silicon, polymers or metals like gold, titanium and platinum [1].

Sensors for temperature, humidity, light, acoustic and vibration are also available for geosciences today. There are also micro-chemo sensors for the detection of small concentrations of certain gases in the air and bio-chemical micro sensors for the detection of spores or bacterial growth in certain locations [1].

However, all these tiny sized sensor platforms will not replace larger scaled platforms. Rather, the variety of sensor platforms will increase. Not every type of application needs small sensors. There will be appropriate platforms for every area of interest. The research field of new sensor is very active at the moment. Soon there will be new sensor types commercially available [1].

3.5. Data standardization initiatives

As an effect of the enormous improvement of the computing platforms, software, network protocols and the sensors themselves, the amount of available spatial data in digital form has been exploding. Various national and international efforts towards establishing spatial data infrastructures (SDI) exist. They promote and share geospatial information throughout governments, public and private organizations and universities [11].

Several initiatives try to simulate the sharing and reuse of expensive geographic information, as the following paragraphs show.

3.5.1. INSPIRE

INSPIRE (Infrastructure for Spatial Information in the European Community) is a directive from the European commission for developing a spatial information infrastructure. The goal of the initiative is the sharing of environmental data between official organizations and institutes between Europe. The directive has entered into force on the 15th May 2007. The full implementation is required for the year 2019. INSPIRE is based on a set of principles

including things such as requiring data to only be collected once and kept where it can be maintained most efficiently⁶.

3.5.2. Global Earth Observation System of Systems (GEOSS)

GEOSS is an initiative for developing a global and flexible network of content providers. It is a “system of systems” that should link together existing and planned observing systems all over the world. It should also support the development of new systems where gaps currently exist. A goal is to work with common technical standards so that data from different institutions can be linked. GEOSS should help to empower the international community to protect itself against natural and human-induced disasters, for understanding environmental sources of illnesses, managing energy resources and many more environmental related topics⁷.

3.5.3. Copernicus

Copernicus is a system like GEOSS, but for Europe. It consists of a set of systems for monitoring the Earth. Sources are earth observation satellites and in-situ sensors. In-situ sensors are sensors like ground stations, airborne or sea-borne measurements. Copernicus was called GMES (Global Monitoring for Environment and Security) before December 2012⁸. It got the name Copernicus in memorial for the European scientist and observer Nicolaus Copernicus who searched for a better understanding for the world in the 16th century [12].

3.5.4. Shared Environmental Information System (SEIS)

SEIS has been launched by the European Environmental Commissioner Stavros Dimas in January 2008. SEIS wants to improve the usage of ICT technologies for collaboration between organizations. The collaboration between European public sectors should help to increase growth, security, jobs, freedom and health and to create a safe environment for Europe. INSPIRE and GMES are initiatives for the realization of SEIS [12].

⁶ INSPIRE, <http://inspire.jrc.ec.europa.eu> (August 2013)

⁷ GEOSS, <http://www.earthobservations.org/geoss.shtml> (August 2013)

⁸ Copernicus, <http://copernicus.eu> (August 2013)

3.5.5. Global Spatial Data Infrastructure Association (GSDI)

GSDI is one of the first organizations that tried to encourage an international cooperation between spatial institutions. GSDI is supported by the U.S. Geological Survey (USGS). It tries to set up local, national and international SDIs (Spatial Data infrastructures) [11].

3.6. Data Collection

Collecting sensor data consists of monitoring the entire covered region of interest and reporting any data of interest.

Two types of monitored phenomena can be distinguished: continuous and discrete monitoring. Continuous monitoring means for example the monitoring of a toxic cloud within a city. Discrete monitoring can be the checking if a car did pass or not [5].

Furthermore, there can be distinguished between the following types of data collection tasks [5]:

- Raw data queries
- Aggregation queries (e.g. min, max, average)
- Data estimation queries (e.g. the estimation of an toxic cloud)
- Qualitative queries (e.g. trying to find an event)

Regarding real-time systems, one can make a distinction between hard-, firm- and soft-systems [13]:

- Soft
If response-time constraints of soft real-time systems are not met, the performance of these systems is degraded but not destroyed.
- Firm
In a firm real-time system a few missed deadlines do not lead to a total failure. But missing several deadlines may lead to complete or catastrophic system failure.
- Hard
If one response-time constraint of hard real-time systems is not met this leads to complete and catastrophic system failures.

Figure 5 shows each type of real-time system graphically.

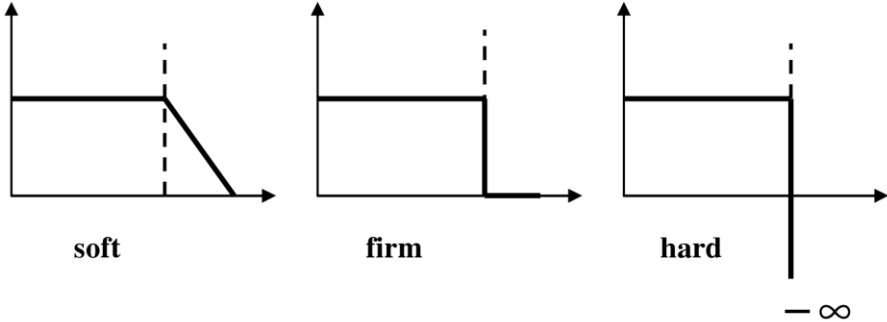


Figure 5 - Real-time Deadlines

Within the context of geo-sensor systems the word “real-time” is often not a pre-set numerical time, but more often means a qualitative expression such as “immediately” or “ad-hoc” [14]. This thesis presents an implementation of a system dealing with weather data which does not require hard deadlines. As a consequence the implementation will be restricted to soft deadlines.

4. Standardized interfaces and exchange protocols

The following chapter deals with initiatives for standardization and introduces the “Open Geospatial Consortium” (OGC). It also points out the benefits of standards.

Earth systems, like the atmosphere, the hydrosphere or the biosphere, are connected to each other. Thus it is very important to share information between several geospatial disciplines. Scientists seek to manage water, waste, energy, pollution, forests, oceans, climate and more data in a standardized way [15]. Therefore, standards are a prerequisite for the creation of interoperable and portable infrastructures. They help to achieve a maximum interoperability if they are applied throughout the whole workflow [11].

4.1. Initiatives for Standardization

There are efforts of different institutions for the standardization of geospatial data and services. The following figure shows a good overview:

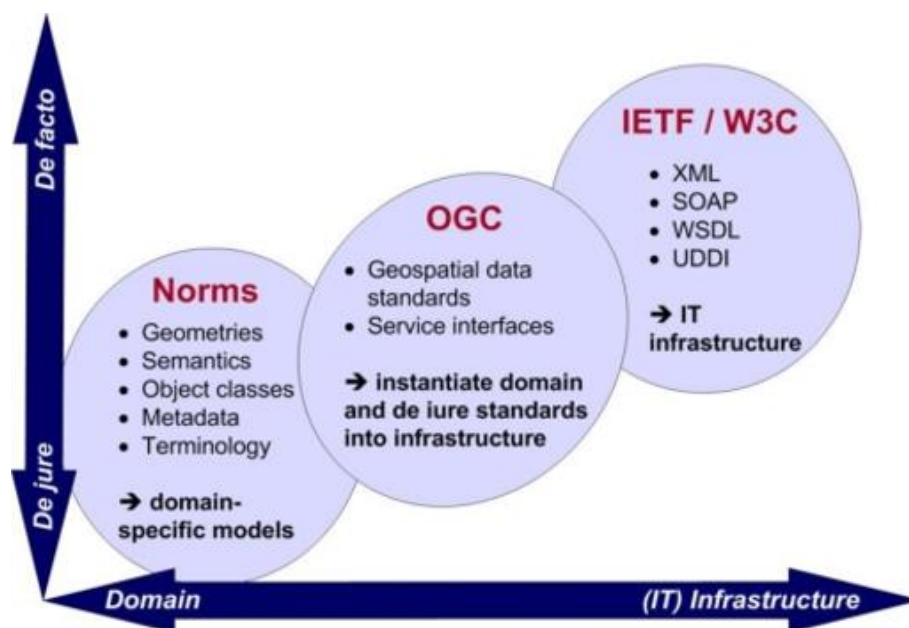


Figure 6 - Standards [11]

“De facto”-standards are standards or commonly used technological specifications. Often these are specified by the World Wide Web Consortium (W3C) and developed “on-demand”. “De jure” standards are domain-specific definitions and mostly legally binding [11]. The “Open Geospatial Consortium” (OGC) was founded in 1994 as a bridge between these two areas. It is the leading institution for the establishment of geospatial standards and moreover a

nonprofit organization. It involves universities, research organizations, NGOs, companies and government organizations working together to develop institutional information sharing standards [15].

The OGC also develops standards for other domains such as aviation, business intelligence, emergency response, mobile internet or sensor webs [15].

One group of standards, developed by the OGC concerning sensor networks, is called “Software Web Enablement” (SWE) [14]. It consists of following standards [14], [15]:

- **Sensor Model Language (Sensor ML)**
This standard specifies models and XML encoding for specifying the geometric, dynamic and observational characteristics of sensor systems. Low level definitions of atomic process models and process chains allow the specification of many different types of sensors from simple visual thermometers to earth observing satellites. The processes and components are all defined in GML (Geographic Markup Language). Sensor ML supports discovering different types of sensors and the processing and analysis of the retrieved data.
- **Observations & Measurements (O&M)**
O&M specifies a description of sensor observations in the form of general models and XML implementations. Several terms for the measurements and the relationship between them are labeled. The measurement results are expressed as quantities, categories, temporal or geometrical values, arrays or composites of these. The framework provides document models for exchanging information of observation acts and their results within different scientific and technical domains.
- **Transducer Model Language (TML)**
TML provides a method and a message format describing how raw transducer data should be interpreted.
- **Sensor Observation Service (SOS)**
This standard defines a web service interface for querying observations, sensor metadata and representations of observed features. It also provides a standard for

registering and removing sensors.

- **Sensor Planning Service (SPS)**

The SPS supplies an interface for planning an observation query. A feasibility check is performed while data from several sensors is set up.

- **Sensor Alert Service (SAS)**

SAS identifies predefined events and then generates and sends alerts in a standardized protocol format.

- **Web Notification Service (WNS)**

This service delivers alerts to end-users using e-mails or text messages. It also provides an open interface so that services can exchange asynchronous messages with each other.

- **Sensor Web Registry**

This registry stores metadata such as the location of a sensor, what they measure or whether they are static or mobile.

The following figure shows how the different standards work together in a sensor network:

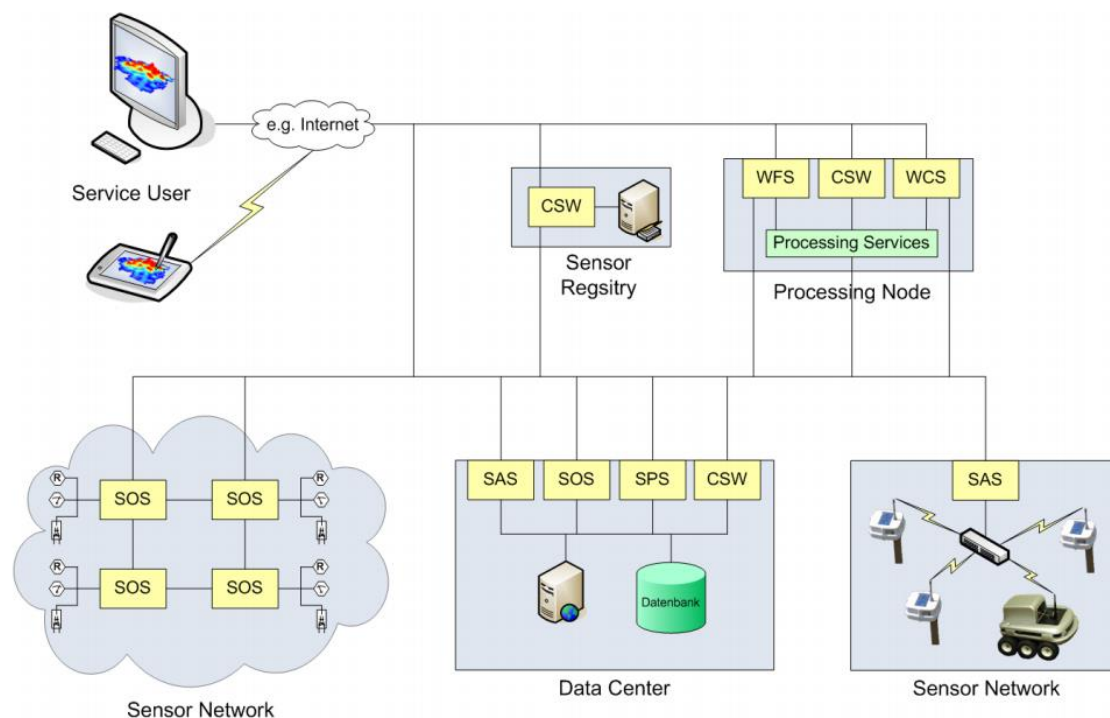


Figure 7 - A Standardized Sensor Network [14]

The OGC is currently working to specify a common namespace for these standards. It is hoped that this will help minimize redundancy and improve reusability of the various standards [11].

4.2. Benefits of standards

Applications can benefit in several ways by using standards. The following list shows some examples [11].

- Interoperability between services and different heterogeneous data sources is improved.
- The extensibility of the system increases.
New data sources can be integrated more easily if the system is built in a modular structure.
- Automated machine-to-machine (M2M) communication becomes easier.
Computers can use well-known interfaces for combining services.
- With uniform interfaces request consistency is guaranteed.
This is important for the implementation of client applications.

Although using standards has a lot of advantages, developer should consider that using them increases the effort for establishing the system when first deploying it. Implementing a standard can be quite time consuming. Nevertheless, over the long-term, the operation, administration and transaction costs should be minimized [11].

5. Version Control Systems

For the practical part of the paper an introduction to version control systems is necessary. Therefore, this chapter introduces different types of version control systems: centralized and distributed systems. It points out advantages and disadvantages of each system and makes a comparison between existing software solutions. The chapter ends with a selection of two suitable systems for the practical implementation.

Version Control Systems (VCS) are tools that help developers to manage changes in their software. They are used for documentation, sharing and merging of code. This is an essential part of software development and effective use of VCS contributes to the success of projects. Software is usually developed in teams where members work parallel on the same code. So it is very important to have a tool for sharing and merging changes [16].

Because this task is so important, there are already a lot of tools available. In this thesis the author examines using these systems for managing real-time geo-data. The process of merging geo-data streams with each other is not completely dissimilar to merging code of several developers to each other. Also documentation is needed for handling sensor data.

In general there can be made a classification of two types of version control systems: distributed and centralized systems.

5.1. Centralized version control systems

Centralized version control systems consist of one single server that contains all the versioned files and a number of clients that are allowed to check out and import files to that server. This architecture has been a standard for many years [17].

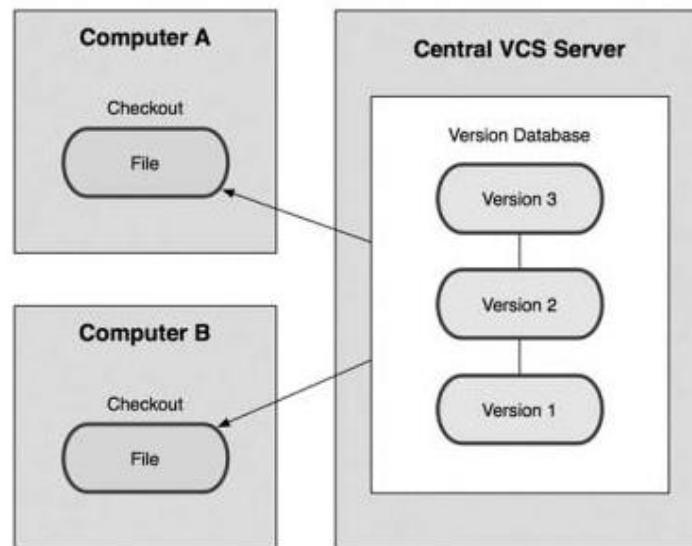


Figure 8 - Centralized Version Control System [17]

Figure 8 shows graphically the structure of a centralized version control system.

5.1.1. Advantages

A big advantage of centralized version control systems is the easier way of learning how to use them. They don't offer so many features so that the learning curve of new users is really quick. Another advantage is that everybody in a project knows what the others are doing at the moment. It is not possible for one user to hide his work. For administrators a central version control system is easier to deal with than a decentralized one. It can also be seen as advantage that centralized version control systems are very common in companies and therefore the existence of a huge know how basis.

5.1.2. Disadvantages

The centralized server is a single point of failure. When the server goes down for one hour, nobody can work with the version control system any more. Another disadvantage is, that if the hard disk, where the central server is stored on, becomes corrupted, all the data are lost. When using central version control systems making proper backups of the server is essential.

Examples for centralized version control systems are Subversion, CVS or Perforce.

5.2. Distributed version control systems

In a distributed version control system, each client fully mirrors the whole repository like figure 9 shows [17].

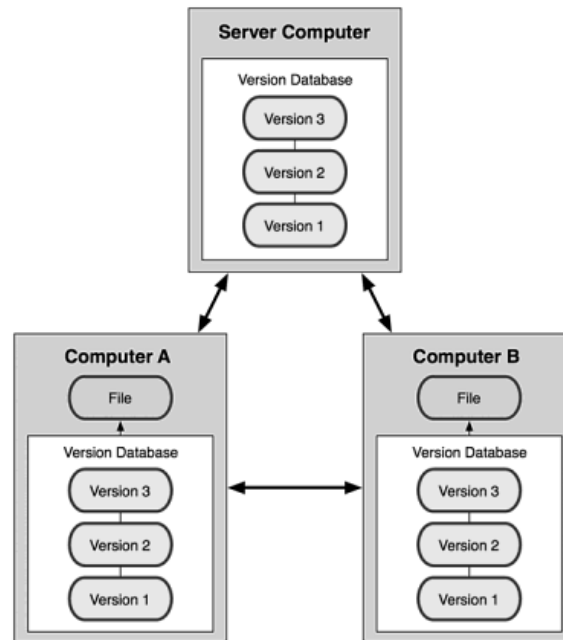


Figure 9 - Distributed version control systems [17]

This composition brings different advantages and disadvantages compared to centralized version control systems.

5.2.1. Advantages

Every checkout of a client when using a decentralized system is a full backup of the repository. If the repository goes down, any of the client repositories can be copied back on the server and fully restores the system. Another advantage is that a hierarchical model for workflows is possible. So it is possible to work with different people in different groups within the same project. Decentralized systems also show an increase on speed. Storing the data to the local repository can happen really fast.

5.2.2. Disadvantages

Decentralized systems are sometimes considered as more complex to learn [18]. For storing the data to the remote connection a SSH key has to be used. The setup for administrators is

more difficult in the beginning. However, decentralized version control systems also offer more features than centralized ones.

Examples for decentralized version control systems are Git (this software can also be used as centralized version control system), Mercurial, Bazaar or Darcs.

5.3. Evaluation and comparison of existing version control systems

This chapter introduces existing version control systems and evaluates them. Out of that evaluation two systems will be chosen for subsequent implementation of a geo-data processing system.

The following table shows a general overview of existing centralized and decentralized systems:

	Initial Release	Developer	Platform	License	Repository model
Subversion ⁹	2000	Apache Software Foundation	Unix, Windows, Mac	Open source	centralized
Perforce ¹⁰	1995	Perforce Software Inc.	Unix, Windows, Mac	Proprietary	centralized
Team Foundation Server ¹¹	2005	Microsoft	Windows	Proprietary	centralized
GIT [17]	2005	Junio Hamano	Posix, Windows, Mac	Open source	decentralized
Mercurial ¹²	2005	Matt Mackall	Unix, Windows, Mac	Open source	decentralized

⁹ Subversion, <http://subversion.apache.org> (August 2013)

¹⁰ Perforce, <http://www.perforce.com> (August 2013)

¹¹ Microsoft, <http://www.microsoft.com> (August 2013)

¹² Mercurial, <http://mercurial.selenic.com> (August 2013)

Bazaar ¹³	2005	Canonical Ltd.	Unix, Windows, Mac	Open source	decentralized
----------------------	------	----------------	--------------------------	-------------	---------------

Figure 10 - Overview: Version Control Systems

For further evaluation the author chose three of the versioning systems in the table. The version control systems Subversion, Git and Mercurial have been picked, because they appear to be the most widely used systems and they are open source applications [16].

5.3.1. Apache Subversion (SVN)

CollabNet founded the Subversion project in October 2000. In February 2010 it became an open source Apache Software Foundation Project. SVN is a centralized version control system. Collaboration with other developers, even in remote locations, is possible since SVN uses HTTP. HTTP (Hypertext Transfer Protocol) is a standard protocol which is allowed by most firewalls [18].

SVN offers a lot of features such as a merging tool, branching support, commit messages and a whole lot more. It tries to solve conflicts if two developers have been working on the same place in a file. SVN also features true atomic commits. That means that either a whole commit completes or nothing is committed, which prevents repositories ending up in a corrupted state [18].

Because SVN is open source, easy to learn and offering a lot of features that other version control systems had not previously offered, it found a wide adaption by a large number of companies. That results in a wide support with third-party applications. Nevertheless, Subversion suffers from the disadvantages of centralized systems as discussed in chapter 5.1.2. If using a slow internet connection the speed of updating or committing data goes down rapidly. Also SVN merging abilities suffer if a file is not cleared of the additional code generated [18].

¹³ Bazar <http://bazaar.canonical.com/en> (August 2013)

5.3.2. Mercurial

After the free version of Bit Keeper was removed from the market the Mercurial project was started in April 2005 by Matt Mackall. Mercurial is open source and is a decentralized versioning system that includes all the advantages and disadvantages of those kinds of systems. For example changes are usually pushed to the local repository which gives a huge speed increase. Secure Shell (SSH) can be used to push to remote locations. SSH is very similar to the standard HTTP protocol but more secure. That can be an advantage if all HTTP ports are closed in a locked-down network [18].

Mercurial is written in Python, which ensures good cross-platform compatibility. It is mostly a command line tool but there are also graphical implementations available. Mercurial offers also more features such as allowing change to be exported to a file. Another user can import that file to a remote repository still under the original name of the first user. This can be useful if new code has to be reviewed and approved of other team members before committing [18].

Because of all these features Mercurial found a wide number of users including the companies Mozilla, Netbeans and Growl.

5.3.3. Git

Git started to be developed around the same time as Mercurial. Linus Torvalds, the inventor of the Linux kernel, programmed the first version of Git in just four days. Git was developed for managing the source code for the Linux kernel development with two core ambitions: speed and security. It is a decentralized version control system [18].

Git takes a special focus on rapid branching. In Git it is possible to make separate branches for special features that can be merged back in the repository after they have been finished. Git is also very scalable. Even managing a huge project will not slow it down [18].

The local use of Git is quite impressive. As one disadvantage it can be said, that the set up and learning curve of Git is perhaps more difficult than for other systems. The communication with a remote Git repository requires having SSH keys for the local and the remote machine. Nevertheless, there are a lot of books and online resources available for getting to know this versioning system [18].

5.4. Selection of two suitable systems for implementation

For the practical part of the paper, the implementation of version control systems for the processing of real-time geo-data, the author of the thesis decided to use Subversion and Git.

These two systems were chosen to provide a comparison between centralized and decentralized systems. They are two very popular systems which are used in many companies. Git won over Mercurial because it is faster and offers more (branching) features [18].

6. Visualization of Sensor Data

This chapter focuses on the visualization of sensor data. The visualization taxonomy of Buja et al (1996) is introduced. GIS (Geographic Information Systems) are described in more detail through presenting existing software solutions. Moreover, a connection of GIS with sensor networks is made.

6.1. Terminology

Visualization not only means the construction of graphs but also the process of interacting with the parameters of the graph. Many graphical views of a data set should be provided for understanding and gaining better insight into the data [19].

Visualization is especially important for spatial data, because graphical tools are often more intuitive for non-specialists. Thus people, who have knowledge of the subject, even if they are not statisticians, are better able to participate in the process of getting data insight [19].

Data visualization in general is concerned with graphical tools. For spatial data visualization, such as a map display, cartographic tools are needed. This is called “cartographic visualization” [19].

It is possible to make a distinction between categorical and quantitative visualization methods. Examples for categorical systems are bar charts or mosaic plots. Whereas examples for quantitative systems are boxplots, histograms, dot-plots, quartile-plots, residual-plots or time series plots [19].

Categorical visualization methods are more difficult to show by graphical approaches.

6.2. Visualization Taxonomy

Buja et al (1996) created a taxonomy of the current research on visualization of high-dimensional data [20]. They divided data visualization into two parts: rendering and manipulation [20].

Rendering means the decision of what to show in a plot and what type of plot to use for the data. This means for example techniques for displaying distributions. There is also a division made between univariate and multivariate data [19].

Manipulation of data refers to operations on each plot and how to organize multiple plots for exploring data [19]. The following tasks for data exploration can be defined: finding gestalt (identifying patterns, shapes ...), posing queries and making comparisons [20].

Linking multiple views means a mechanism that links the graphical query to the graphical response. A user should be able to pose a query graphically and the computer should present the response graphically too [20].

“Pattern perception” refers to the detection and assembly of geometric objects for discovering patterns in the encoded data. Table lookup is about the task of making queries on individual cases [19].

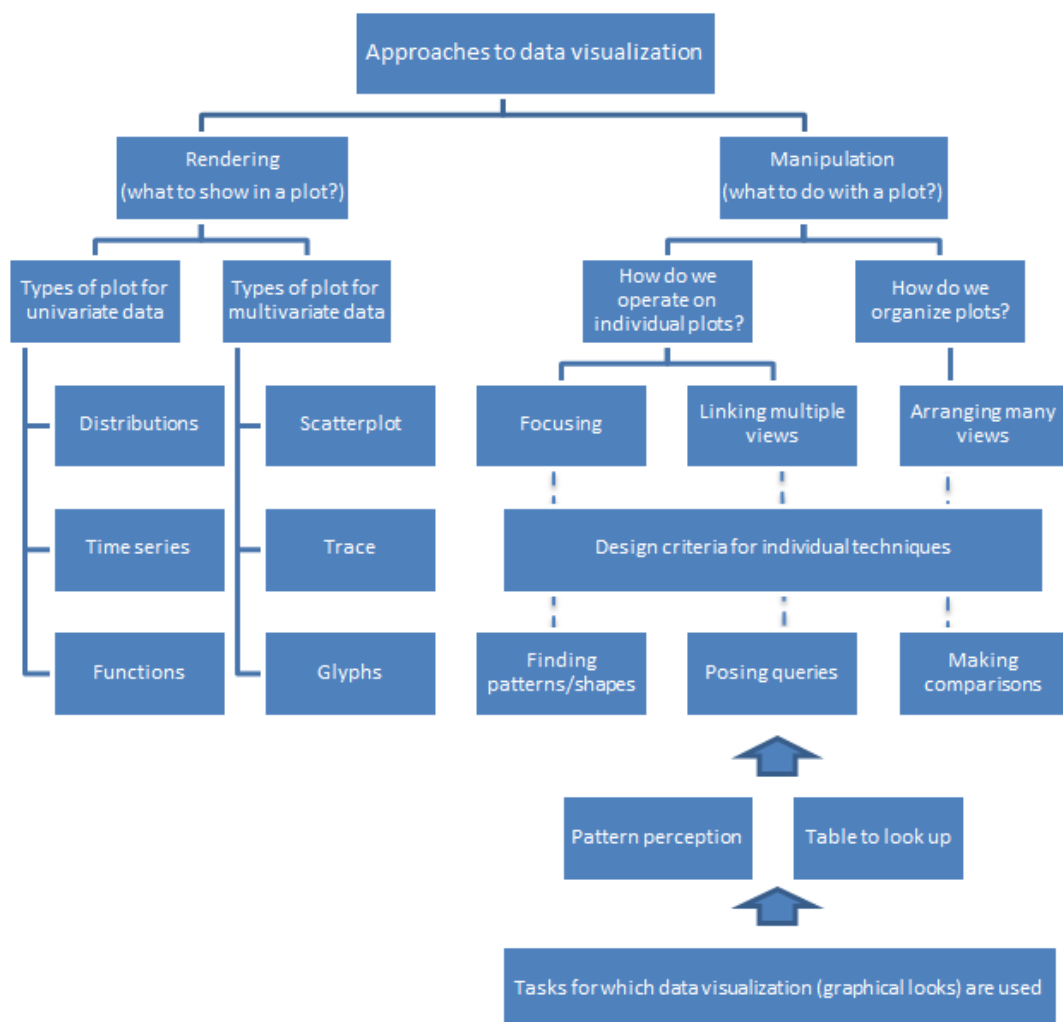


Figure 11- Taxonomy of Buja [19]

Figure 11 shows the taxonomy of Buja et al (1996): the interface between visualization tasks and the development of visualization techniques.

6.3. GIS (Geographic Information Systems)

Geographic Information Systems or so called GIS are software tools helping to manage and visualize spatial data. There exist many definitions of those systems.

Blakemore (1986):

“Computer packages, which integrate the storage, manipulation, analysis, modeling and mapping of digital spatial information [21].”

Burrough (1986):

“Powerful sets of tools for collecting, retrieving at will, transforming and displaying spatial data from the real world [22].”

Anenucci (1991):

“Computer system that store and link non-graphic map features to allow a wide range of information processing and display operations, as well as map production, analysis and modeling [23].”

In general, GIS consist of five functional components [24]:

- Data acquisition and data verification
- Data storage and database management
- Data transformation and analysis
- Data output and presentation
- User interface

There are a lot of different GIS software systems on the market. Examples for commercially available GIS are:

- Autodesk
- ESRI
- Bentley Systems
- Intergraph
- Mapinfo

Examples for open-source GIS are:

- GRASS GIS
- Quantum GIS
- Open JUMP

Examples for Online-GIS are:

- Google Maps
- Google Earth
- Open Street Map

The following sections describe some well-known GIS in more detail.

6.3.1. Autodesk

Autodesk Inc. is an American software development company for 2D and 3D-design, engineering and entertainment software. It introduced the software AutoCAD in 1982, but it has also a broad portfolio with other software for the global market. For example AutoCAD Map 3D is a model-based GIS- and mapping-software. It offers features like point cloud tools for importing, visualizing and styling large sets of 3D laser scanning or LIDAR data. It also includes conversion functionality for GIS and CAD to industry models. Planning and analysis tools make it possible to perform queries, create thematic maps and topologies and to create reports¹⁴.

6.3.2. ESRI (Environmental Systems Research Institute)

ESRI (Environmental Systems Research Institute) is an American company situated in California. It sells geo-information systems. Its best known system is called ArcGIS. This is a platform for designing and managing solutions through the application of geographic knowledge. ESRI Location Analytics offers data visualization and geographic intelligence. With the product ESRI data the company also provides a range of ready-to-use data for GIS visualization and analysis¹⁵.

¹⁴ Autodesk, <http://www.autodesk.com> (August 2013)

¹⁵ ESRI, <http://www.esri.com> (August 2013)

6.3.3. GRASS GIS

GRASS (Geographic Resource Analysis Support System) GIS is an open-source GIS for data management, image processing, graphics production, spatial modeling and visualization of different types of data. It was originally developed by the US Army Construction Engineering Research Laboratories for land management and environmental planning of the military. The software evolved to offer a wide range of applications in many areas. It is currently used in many US governmental agencies and in academic and commercial settings all over the world¹⁶.

6.3.4. Google Earth

Google Earth is a software product of the company Google. It is a virtual globe, map and geographic information system. The standard version is free, but there also exist commercially available versions called Google Earth Pro and Google Earth Enterprise. The commercial versions offer more feature such as measuring tools for distances, the ability to import huge vector-data or building of customized maps. Because the standard version is free, there are already millions of users worldwide. Google Earth is one of the best known GIS¹⁷.

6.4. Sensor Networks and GIS

There exist several approaches for joining sensor information in GIS applications [11].

6.4.1. Sense Web project

Sense Web is a research project of Microsoft. The aim of the project is to establish a Wikipedia-like sensor platform. Users are allowed to include their own sensors in the system. This is helping to get a “community effect” of building a dense network of sensors by aggregating existing and newly deployed sensors within the application. A disadvantage of that project is that it is not based on open (geospatial) standards. It is only based on standard web services [11].

¹⁶ Grass GIS, <http://grass.osgeo.org> (August 2013)

¹⁷ Google Earth, <http://www.google.com/earth/index.html> (August 2013)

6.4.2. PermaSensorGIS

PermaSensorGIS has been developed by several companies. Its goal is to combine sensor systems with GIS-based visualization technologies. Its sensing devices measure rock temperature at ten minute intervals. They have been developed for optimal resource usage like for example data aggregation, power consumption or the communication within the sensor network. In the current implementation a number of open standards and open-source services are used [11].

6.4.3. City Sense

The City Sense project uses sensors to show the overall activity level of a city or activity hotspots in real-time. It also links Yelp and Google to show venues that are operating at specific locations. It uses an urban sensor network for measuring data¹⁸.

¹⁸ Sense Networks, <https://www.sensenetworks.com> (August 2013)

7. Perspectives and Challenges

This part of the thesis deals with perspectives and challenges concerning geo-sensor applications. A forecast of the future development of sensor data, hardware, software and network protocols is made. In addition, issues and problems that can occur like privacy, data ownership or pricing, are mentioned.

GIS are a powerful mechanism for managing information. They started with humble origins; only a set of simple ideas and rather inefficient software. GIS have grown into a sophisticated, global industry in only a few decades. GIS play a dual role. They function as mainstream technology for the management of geographic information and also as a tool for the effective use of resources [25].

Figure 12 shows the development of geo-information systems. The evolution has been more cyclical than linear. In the 1970s the focus was on computer mapping, what means just the visualization of the data. This led to spatial data management. The focus was then on the management of data structures. Geo-references mean the linking of digital maps to databases for querying. Today, GIS focus is on multimedia mapping with 3D and virtual reality visualization which represents a cyclical return to the beginning. The next innovation according to the cycle will likely be focused on data structures and analysis [26]. As said in the introduction, there is a lack of support for querying, processing, mining and analyzing of real-time data streams. This thesis attempts to address this problem.

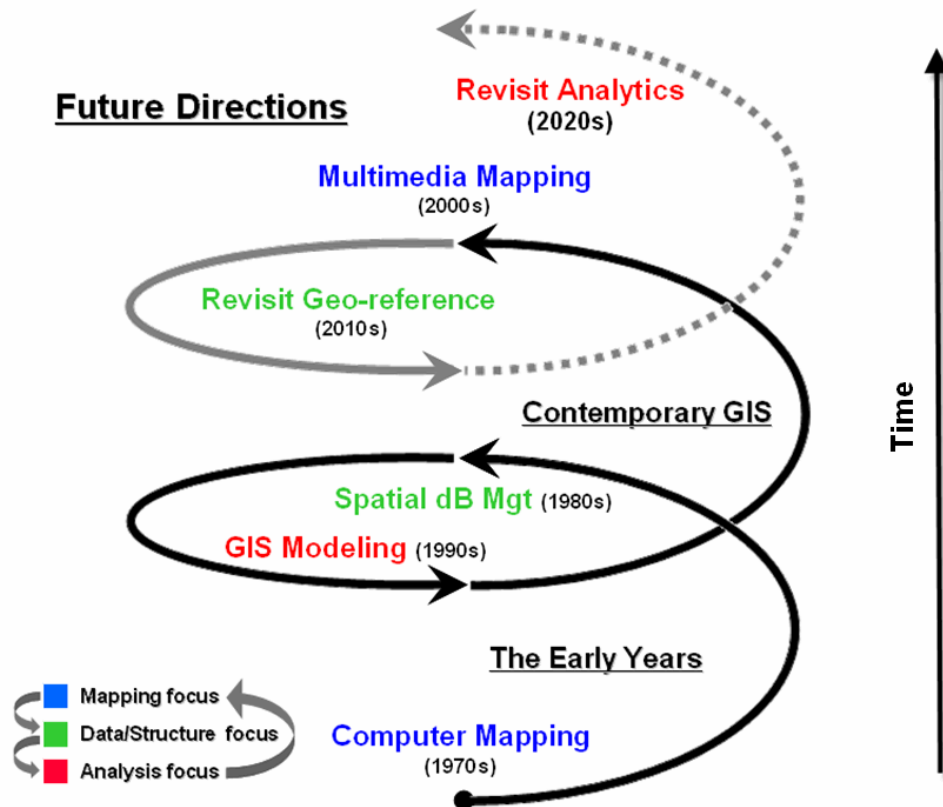


Figure 12 - Future Directions [26]

The following sections explain in more detail some perspectives and challenges of geo-sensor networks and GIS.

7.1. Data

In the future, there will most probably exist an easier access to digital data. This is one of the greatest opportunities for GIS. Data delivery has already been revolutionized by the Internet and search tools built upon its structure [25].

There will be a lot of new types of data, more complete data, higher-resolution data and more timely data. One opportunity is the increasing high-resolution data that is coming from aircraft and spacecraft such as NASA's Earth Observation in the form of remote sensing data. GPS data can also be used since their precision has increased over time and is now standard equipment in many devices such as in public and private vehicles as well as mobile phones [25].

7.2. Hardware

There have been at least four revolutions concerning hardware for GIS in the last decade: the workstation, network, microcomputer and mobility revolution. Certainly, that development will move on [25].

The power and the storage to work with massive databases are likely to continue increasing. Networks are now able to include many new types of computers, such as microcomputers. Nearly every computer is connected to the Internet or can use networks. The internet is the primary source for data exchange, information search and retrieval. Already many GIS's have developed online modules like the ESRI Internet Map Server or Intergraph GeoMedia WebMap. Microcomputers are getting more and more inexpensive and the platforms are becoming more and more widely distributed. There is also a trend towards mobility. This generates a lot of new GIS applications [25].

7.3. Software

Software is getting more intelligent and more mainstream as examples like Google Earth already show. The challenge for GIS software is to make it able to intelligently convert data between structures without intervention of the users.

7.4. Sensor Networks

Regarding the following characteristics of sensor networks, there are several challenges [5]:

- Constrained (Energy, computing power, communication and bandwidth are limited)
- Untethered
- Failure prone

General challenges:

- The design, deployment and management of robust and massively distributed systems which consist of hundreds or thousands of physically-embedded devices
- Ad-hoc communication and collaboration of sensor nodes
- Adaption and self-configuration of the network based on events
- Self-healing skill in case of hardware failure

Local challenges (for each sensor node):

- Local energy management
- Local sensing, data collection and processing
- Collaboration and coordination with neighbor nodes

Global challenges (the sensor network as a whole):

- System lifetime and energy management
- Large phenomena sensing and tracking
- Global change detection processing

In the past there has been a paradigm shift concerning the organization of the network. The decision making and collaboration became a local task. Each sensor nodes has a so called “self-organization” [5].

This paper focuses on the global challenge of “detection processing”, which means the mining and analyzing of the data after each node has finished its work. This will be evaluated with the use of version control systems. These systems are described in more detail in chapter 5.

7.5. Issues and Problems

Although geo-information system offer a lot of possibilities for the future, there are also some issues and problems which should be considered.

7.5.1. Privacy

With geo-sensor networks it is possible to link private data like personal income, information about the family or health records to geographic locations [25]. For example many cities have public property records that give property and owner information. Fears appear that such data could be used in damaging ways. Laws and legal procedures have to be reconsidered [27].

7.5.2. Data Ownership and Pricing

There are different opinions about the price and ownership of geo-sensor data. It is very expensive to produce data, but copying is nearly for free. Some people say that if federal government creates data at the public's expense, copies should be given out for free. Otherwise they would have charged a second time for the data. Others say data are a product protected by copyright and patent and should be sold for profit [25].

In general, there currently exist two different pricing strategies: cost-based pricing and value-based pricing. Cost-based pricing means that the price of a product consists of the costs of the production and a markup. This strategy leads to relatively high prices. Hence, it can stop the further development of new geo information products. Nevertheless, a large number of firms currently use this pricing technique. Value-based pricing sets the price according to the value a potential buyer attaches to the product. This technique is more suitable for environmental data [28].

7.5.3. New Focus

Geo-sensor networks are based on change. Global climate change, global warming, the ozone hole and other impacts of people on the environment will open a lot of new areas for that technology [25].

8. Requirements for a Prototype

This chapter evaluates if version control systems are suitable for improving the processing, mining and analyzing of real-time geo-data streams. For that purpose, a prototypical implementation using the version control systems Subversion and Git has been developed.

Before presenting the prototype, the requirements of the developed software have to be made clear. The following chapters show the implementation requirements.

8.1. General Description

The general description includes the product perspective, general functions, user characteristics, restrictions and assumptions and dependencies.

8.1.1. Product Perspective

At the moment, as far as the author knows, there exists no geo-sensor data approach using versioning systems. Versioning systems already contain a lot of features that are helpful to simplify the mining, processing and analyzing of real-time data streams. Therefore the creation of such a system seems to fill a market gap.

8.1.2. General Functions

General functions and requirements for the software are:

- [RQ-1] MUST
Sensors generate data that are stored via a version control system

Example for a related user story:

John is a meteorologist. He has many sensors in a field study each generating temperature readings. In order to conduct his research he must securely store all the data reliably as it comes in from the sensors.

- [RQ-2] MUST
A scientist combines different sensors to a new meta construct

Example for a related user story:

John likes to create visualizations that combine humidity data with data on wind speed that another scientist has been storing. He wants to save these combinations of data securely to show them to his colleagues in a few weeks.

- [RQ-3] MUST

Another scientist combines meta constructs to meta-meta constructs

Example for a related user story:

Alfred, a colleague of John, sees John's work. He has another idea of a good combination of weather data. He combines John's construct of humidity and wind speed with sensor data of air pressure for getting an even better weather forecast.

- [RQ-4] MUST

A scientist can view the data via a graphical representation

Example for a related user story:

Alfred wants to view his new data combination to John graphically to show him his results.

- [RQ-5] MUST

Two scientists perform different analysis methods and fuse their work

Example for a related User Story:

Anna, another meteorologist, sees the work of John and Alfred. She uses both of their combinations of weather forecasts for her own work related to tsunami forecasts.

- [RQ-6] CAN

If graphical representations are stored as well in the version control system, a piece of software, like a "time machine" can be created for quick access

Example for a related user story:

Mary, the head meteorologist of the company, wants to view the work of her

employees for a presentation at a conference. She can check out the work of each meteorologist in the past weeks and sees how their work is evolving.

8.1.3. User Characteristics

The expected users of the system are scientists in the field of geography. They should have expert knowledge about the specific geo-data and about how to combine different geo-data streams. It is expected that they know how to handle a software program with a common known user interface in windows.

8.1.4. Restrictions

The system will be developed in C#. Although it is possible to run a .NET program using different operating systems, the version of this thesis will be a prototype that can only be used under windows. The C# program will not be available as an online version. To run the software, a usual in trade internet connection has to be provided. The minimal display resolution is 1024x768.

8.1.5. Assumptions and Dependencies

For using the system one of the following version control systems has to be installed:

- Apache Subversion
- Git

It is possible to switch between the two versioning systems. The system allows working with different kind of geo-data. For example data of a weather station or data from a volcano measurement can be used. Therefore the accuracy of new meta constructs is not checked.

8.2. Specific Requirements

Specific requirements include non-functional requirements and the specific actors.

8.2.1. Non-functional requirements

As non-functional requirements, following aspects of the hardware, the software and design constraints have to be available:

- Hardware
 - Standard specification trade computer
 - Minimal 2 GB RAM
 - Minimal Display resolution 1024x768
 - Broad band internet connection

- Software
 - Operating System Microsoft Windows 7 or higher
 - Version control system Subversion or Git
 - Access to real-time geo-sensors

- Design Constraints
 - A modern, appealing design will be used. The UI will be easy to use with common techniques like drop-down-menus, the mouse or the keyboard. The GUI will be ergonomic designed and stick to common standards.

8.2.2. Actors

For using the software, just one type of actor is involved: scientists who want to handle real-time geo-data.

8.3. Data Model

For analyzing the data model of the implementation use case and sequence diagrams have been created.

8.3.1. Use Cases

Figure 13 shows a general overview of the considered use cases.

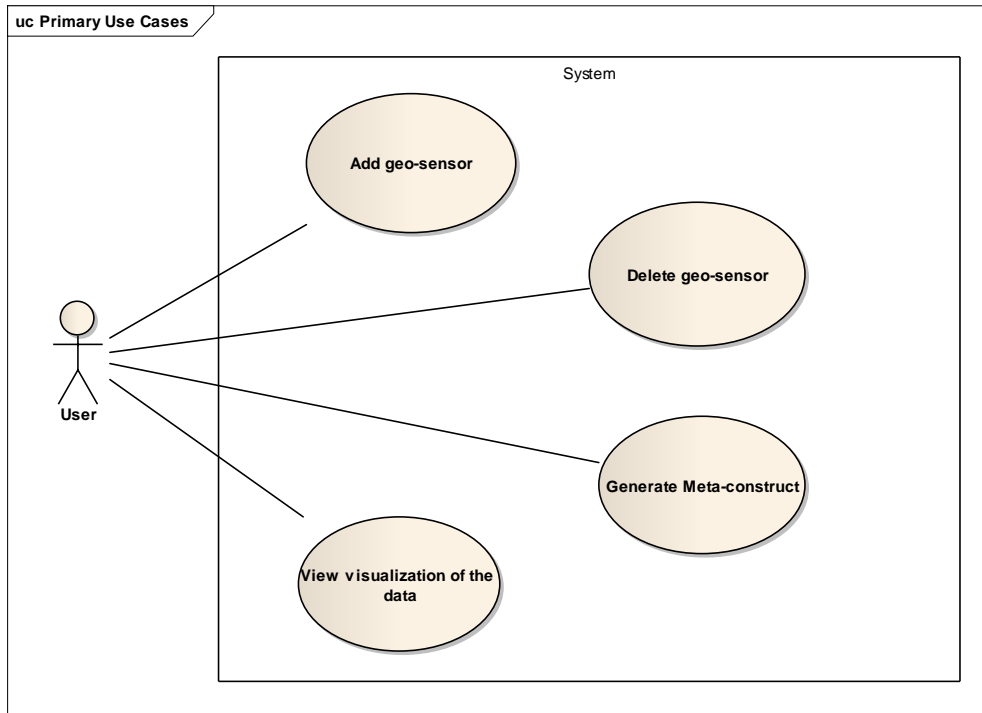


Figure 13 - Primary Use Cases

The tables 1 to 4 provide a more detailed description of each use case.

Use Case ID	1
Use Case Name	Add geo-sensor to system
Actors	User (Scientist)
Description	The user selects a button called “Add Sensor” for adding a new geo-sensor to the system. After that, he is able to choose an existing sensor from his file system. The chosen sensor file has to be already added to an existing version control systems repository. The version control systems “Apache Subversion” or “Git” are possible options to choose. After confirmation, the system displays the data in the geo-sensor file. The display view will be refreshed in a specific interval, so that new data which has been added to the VCS-file in the background will get displayed.
Pre-Conditions	User selects a versioning system: Git or SVN User must have access to the repository of the specific versioning system

Post-Conditions	Selected geo-sensor data is displayed
Normal Flow	<ol style="list-style-type: none"> 1. Select versioning system 2. Choose specific input stream 3. Give the stream a name 4. Confirm action
Exceptions	<ul style="list-style-type: none"> • Data Stream not available • Unknown data format
Priority	High
Frequency of Use	High
Business Rules	<p>Number of sensors which can be added to the systems is unlimited</p> <p>No login necessary. User is identified through versioning system account.</p>

Table 1 - Use Case 1: Add sensor

Use Case ID	2
Use Case Name	Delete geo-sensor from system
Actors	User (Scientist)
Description	<p>The user selects a “Delete sensor”-option. Existing geo-sensors or meta-constructs are displayed. The user can select one or more sensors or constructs he wants to delete. After that he presses a confirmation button. If the user selected a geo-sensor, the sensor will no longer be viewed in the system. If the user selected a meta-construct, it will also no longer be viewed, but also no longer be saved to the version control system in the background.</p>
Pre-Conditions	Geo-sensor must have been added to the system

Post-Conditions	Geo-sensor isn't displayed any more. Existing meta-construct with that sensor are still available
Normal Flow	1. Select existing sensor 2. Press "Delete button" 3. Display is refreshed, sensor is not visible any more
Exceptions	-
Priority	Medium
Frequency of Use	Medium
Business Rules	Every added sensor can be deleted. No login necessary. User is identified through versioning system account.

Table 2 - Use Case 2: Delete sensor

Use Case ID	3
Use Case Name	Generate a new meta construct
Actors	User (Scientist)
Description	The user selects a "make new construct"-option. A new window for doing that will be displayed. The user can now select one or more existing geo-sensors or meta-constructs of the system. He also has to insert combination rules for the sensors and give the new construct a meaningful name. After that, the user has to confirm his action. The new construct will now be saved to the version control system in the background. The standard view will appear again and the new construct is visible.
Pre-Conditions	Geo-sensors must have been added to the system.
Post-	New meta-constructs will be stored in system.

Conditions	
Normal Flow	<ol style="list-style-type: none"> 1. Select existing geo-sensors or meta-constructs 2. Define rule for connection 3. Give the new connection a name 4. Confirm action
Exceptions	<ul style="list-style-type: none"> • No rule defined • Defined rule is not legal
Priority	High
Frequency of Use	High
Business Rules	<p>Number of constructs which can be added to the systems is unlimited.</p> <p>New constructs can be built out of existing constructs.</p>

Table 3 - Use Case 3: Generate Meta Construct

Use Case ID	4
Use Case Name	Graphically view meta construct
Actors	User (Scientist)
Description	<p>The user selects a “View Data”-option. He sees possible options for displaying the data. The possible options will vary with the data of the sensor. E.g. for numerical data different views are possible like for alpha-numeric data. The user selects a view and one geo-sensor or meta-construct of the system. He then confirms his action. After that a new window will be opened displaying the data in the specific format.</p>
Pre-Conditions	Geo-sensors have been added to the system
Post-Conditions	New windows opens that displays the data

Normal Flow	<ol style="list-style-type: none"> 1. Select one geo-sensor or meta-construct 2. Select a specific kind of view 3. Confirm action
Exceptions	<ul style="list-style-type: none"> • Specific view is not available for this kind of data
Priority	Medium
Frequency of Use	Medium
Business Rules	Not every view suits for every kind of data.

Table 4 - Use Case 4: View Meta construct

8.3.2. Sequence-Diagrams

Possible interactions between the user, the system and the specific version control system are displayed from figure 14 to 17 with the help of sequence diagrams.

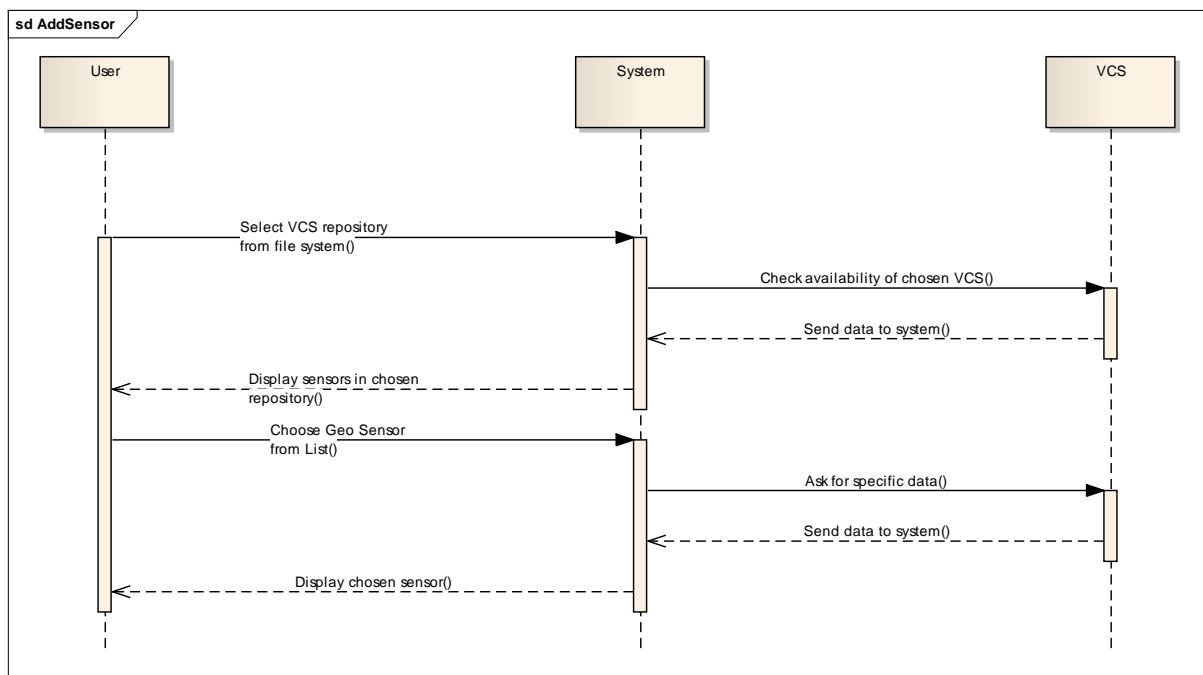


Figure 14 - Sequence Diagram: Add Sensor

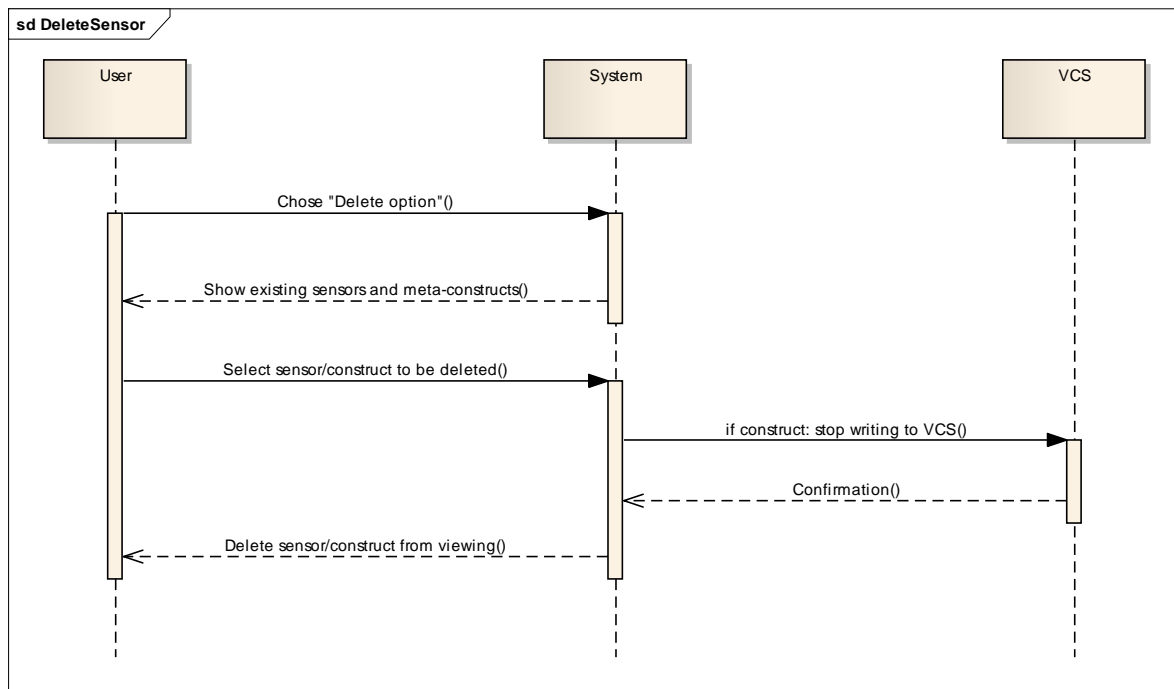


Figure 15 - Sequence Diagram: Delete Sensor

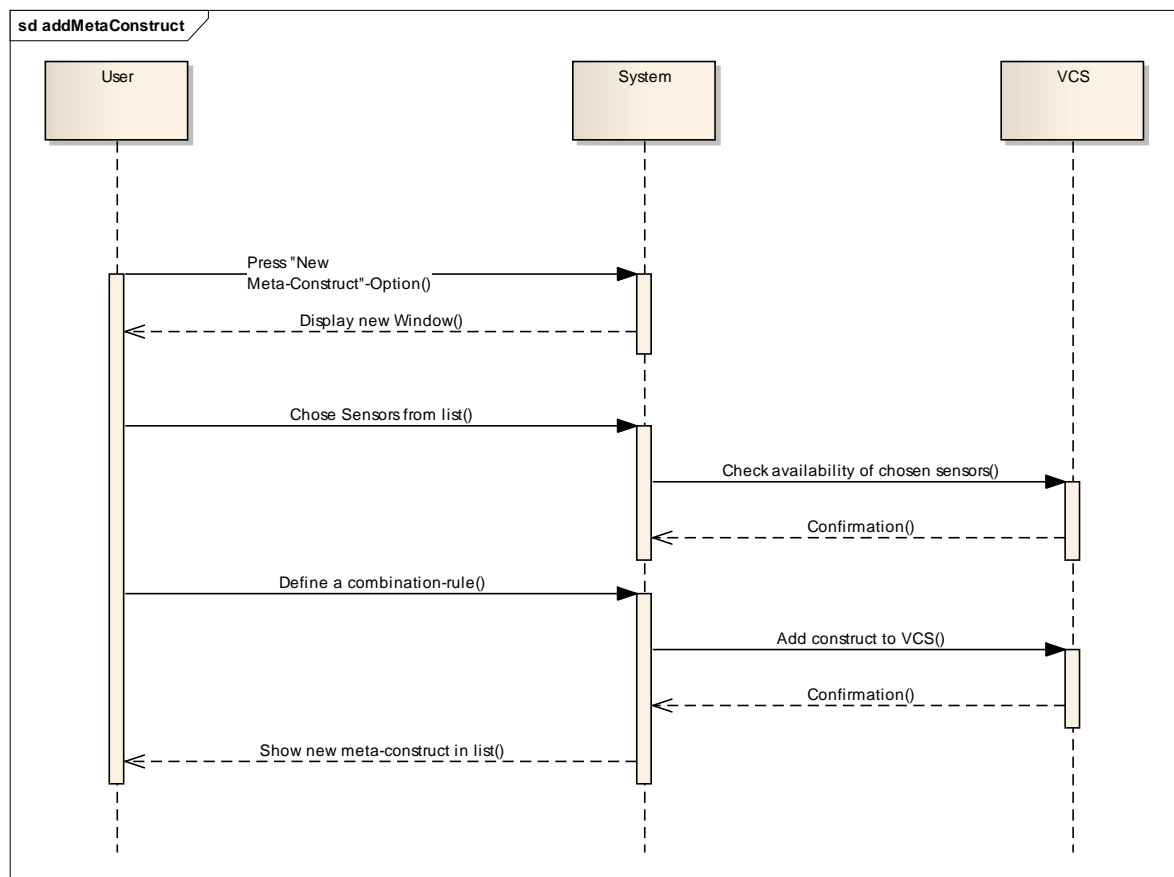


Figure 16 - Sequence Diagram: Add Meta Construct

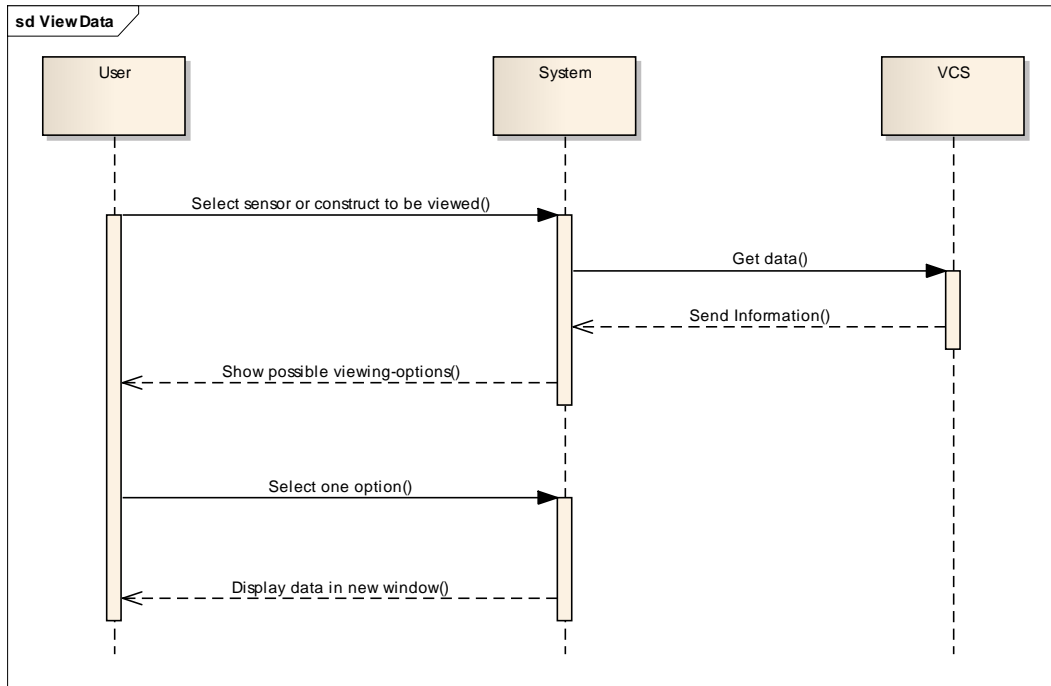


Figure 17 - Sequence Diagram: View Data

8.4. Test Data

The implementation can be used with any kind of geo-sensor text data. For the concrete examples in this thesis, data from a weather station in Germany has been used. Data are available in a period of five minutes for the years 2005 until 2012 [29].

These parameters are available in a CSV-format as table 5 shows.

Timestamp	Temperature	Humidity	Air pressure	Rain	Wind	Direction
01.10.2012 00:05	5.8	79	1019	0	0	105
01.10.2012 00:10	5.7	80	1019	0	0	105

Table 5 - Test Data [29]

In a real environment it is not usual to have all the data in one file. Normally a lot of different sensors are delivering the data. Therefore each parameter was stored in a separate txt-file.

Figure 18 shows an example of one test file:

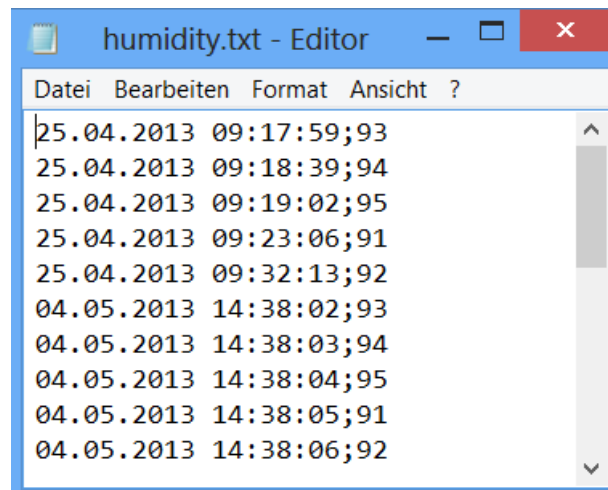


Figure 18 - Test Data File Example

The timestamp and the value are stored in the file separated by a semicolon.

Data are usually stored in a database to support further processing. This thesis evaluates if it is possible to use a version control system instead.

8.5. Online repositories

Subversion and Git repositories can be created in free online hosts. These are used to store the data for the prototype system.

Google provides free Subversion hosting through the website “Googlecode”¹⁹ which allows free project hosting for open source projects, while free Git repositories are provided by Github²⁰.

8.6. Programming language

Prior to developing the prototype the programming languages C# and Java were evaluated as alternatives.

Both languages offer libraries for the implementation of the selected version control systems:

- SVNKit

SVNKit is a pure Java toolkit for implementing all features of Subversion into Java

¹⁹ Google Code, <http://code.google.com/intl/de> (May 2013)

²⁰ Github, <https://github.com> (May 2013)

code using an API. It is free to use for open source projects²¹.

- JGit

JGit is the same like SVNKit for implementing Git in Java applications. It is hosted by Eclipse²².

- SharpSVN

SharpSVN is a Subversion client api for .Net applications. It is licensed under the Apache 2.0 license. That means it is allowed to use it in open source and commercial projects²³.

- GitSharp

GitSharp connects Git to .Net and Mono. It is fully compatible to the original Git and should function as a light weight library for C# applications²⁴.

After developing four test programs with each versioning library, it was clear that there were not a lot of differences between the implementations. Nevertheless, the programming language C# was chosen. The prototype should be able to visualize sensor data and the user interface should be easy to develop. In the opinion of the author, Visual Studio offers more support for these tasks than Eclipse.

A disadvantage of using C# and .NET is that the developed interface is not web based. This is a drawback concerning long term stability and sharing the application. Nevertheless, this is not the focus of this paper. The prototype is developed for proving the theory of using version control systems for geo-sensor networks.

8.7. Simulation of a Geo-Sensor

For writing this thesis, no real time geo-sensor has been available. As a consequence a simulation of a sensor was developed.

The user interface of the simulation is shown in figure 19.

²¹ SVNKit, <http://svnkit.com> (August 2013)

²² Eclipse JGit, <http://eclipse.org/jgit> (August 2013)

²³ CollabNet, <http://sharpsvn.open.collab.net> (August 2013)

²⁴ GitSharp, <http://www.eqgon.com/index.php/GitSharp> (August 2013)

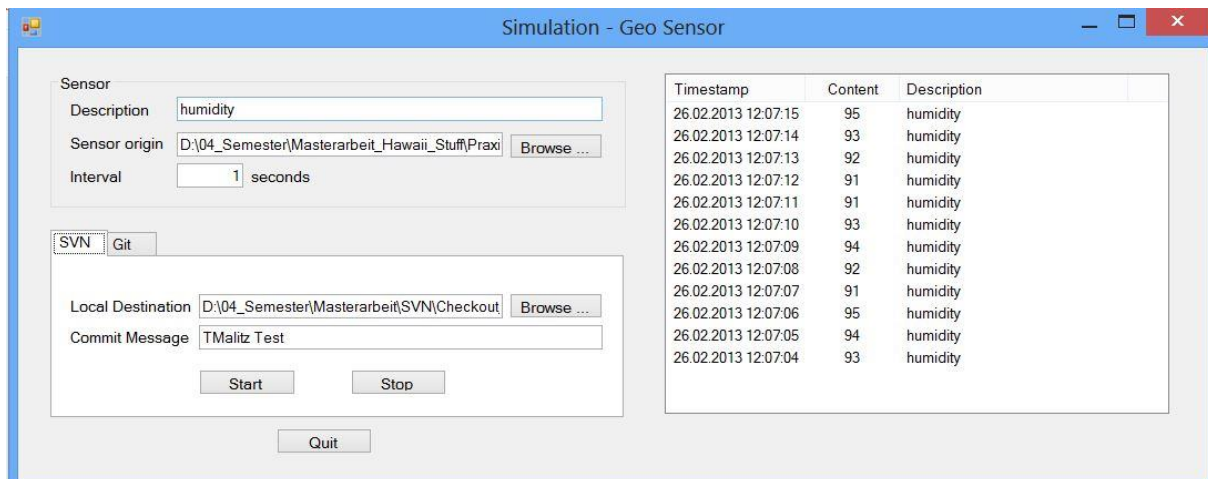


Figure 19 - Simulation

The following paragraphs describe the functionality of the simulation software.

In the group box called “Sensor” the user can store the description of the current sensor. For example, if a user is working with weather data, this can be “humidity”. In the field “sensor origin” the user must place a path to a .txt-file, where the data of the sensor is stored. This can be done with the use of the “Browse...” button. There is the possibility to specify a specific interval, in which the sensor file is read in units of seconds. If no specific interval is given, the software works with a one second interval.

In the tab control section, the user can choose if the current simulation works with Apache Subversion or Git. Figure 20 shows the tab control view for Subversion.

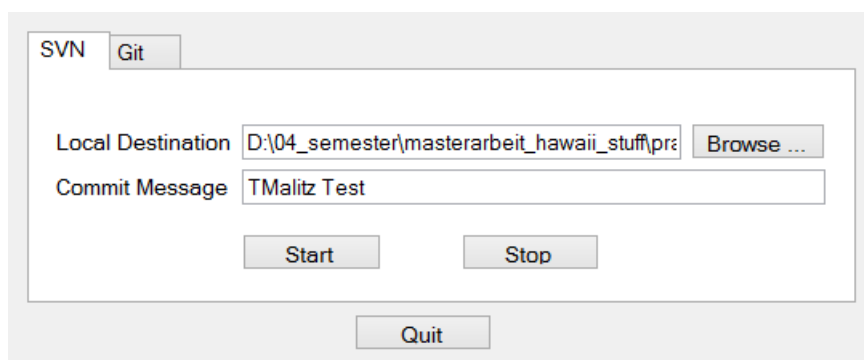


Figure 20 - Tab control "SVN"

In the field “Local Destination” the user must specify a path to a file in the local system containing an existing versioning system repository. This is the directory, where the data should be stored and committed to the versioning system. It is necessary to provide a commit message for the versioning system. This should be a meaningful, short phrase.

Upon pressing the start button, the system starts to import the data from the sensor file to the versioning system in the given interval. The imported data are displayed on the right section of the user interface. The fields “timestamp”, “value” and “description” are filled in. It is possible to stop this process by pressing the button “Stop”.

Figure 21 shows a similar tab control, but for the versioning system Git.

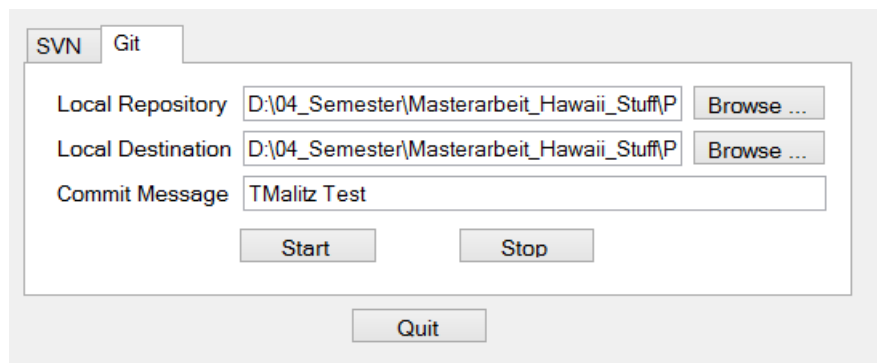


Figure 21 - Tab Control "Git"

For the version control system Git, a local repository path is needed. It has to be a path on the local file system where a Git repository is already cloned. As with Subversion, the local destination of the sensor file and a commit message must also be provided. The button “Start” starts the import of the data from the sensor and the button “Stop” stops it.

To exit the whole application, a user can press the button “Quit”.

9. Actual Implementation with Subversion and Git

The following section describes the concrete implementation of the prototype using the versioning systems Apache Subversion and Git. The available screens and settings are presented as well the visualization options of the data.

9.1. Start Screen

Figure 22 shows the start screen of the application.

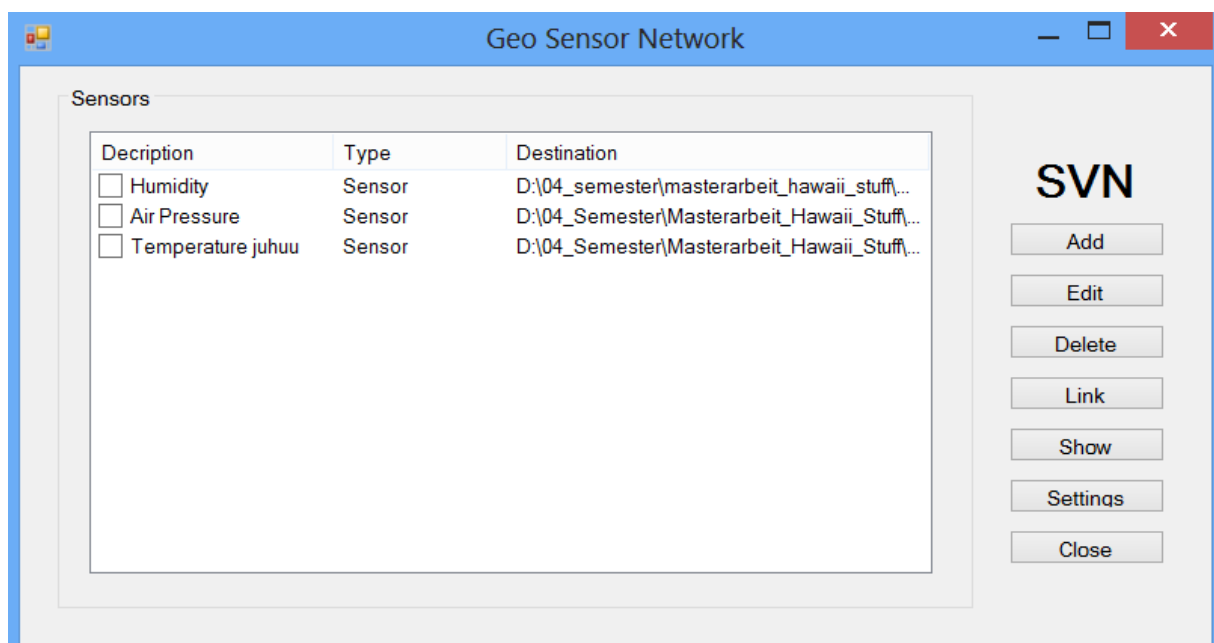


Figure 22 - Start Screen

The group section called “Sensors” contains a description, a type and a destination column. The description column contains the name of the sensor or the construct. This name can be chosen freely. The user is not allowed to store two sensors in the system with the same description.

The system distinguishes between two types of sensors: “real” sensor data and meta-constructs.

If adding a sensor with the button “Add”, the software expects a real sensor. If linking two or more sensors or constructs with the button “Link”, the software stores the new construct with the type “Construct”.

In the column “Destination” the path to a file in the local versioning system is displayed. It is a path to a Subversion or Git repository. From this location the commit to the system is executed.

9.2. Adding a geo-sensor

Figure 23 shows the user interface to add a new sensor to the system:

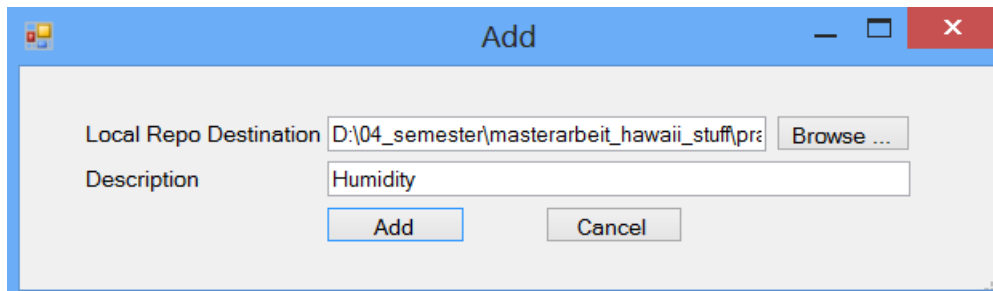


Figure 23 - GUI of adding a sensor

A local repository destination has to be provided. This has to be a local path to an existing file in a Subversion or Git directory. The description that is chosen for characterizing the sensor has to be a unique name. No sensor can occur twice in the system.

Before adding a sensor to the system, the simulation software described in chapter 8.7 has to be started. After clicking the button “Add” the new sensor is displayed on the start screen of the system.

9.3. Editing a sensor

Figure 24 shows the mask for editing a sensor.

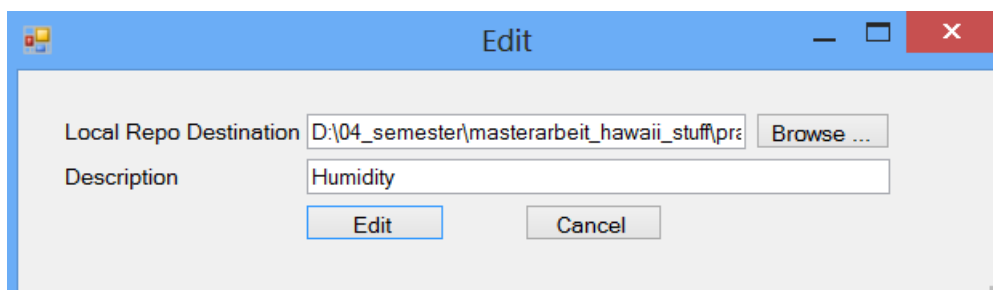


Figure 24 - GUI of editing a sensor

It is only possible to edit sensors, but not constructs. The reason is that constructs start a windows service in the background. It is not possible to edit an already working windows service. It has to be stopped and installed again for that purpose.

This mask allows editing the local repository destination and the description of a sensor. The description of a sensor has to be a unique name in the system. Therefore this mask can be very useful.

9.4. Linking sensors or constructs

In figure 25 the GUI of linking sensors is displayed.

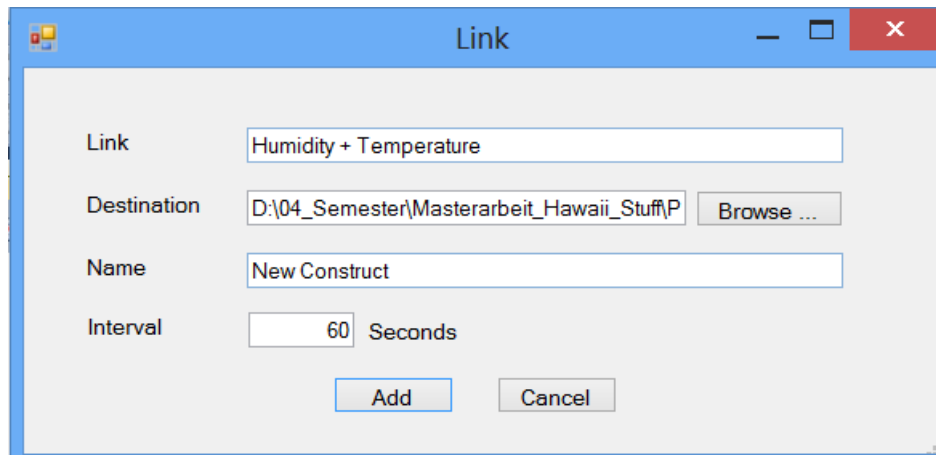


Figure 25 - GUI for linking data

By default, after clicking the button “Add” the selected sensors or constructs from the start screen will be shown in the field “Link” connected by a plus-sign (+). The user can change this default view. For example in figure 25, the two sensors get multiplied by 3. In the background, this functionality is realized with a code library called “Math Parser .NET C#” from the website “Code Project”²⁵.

In the field “Destination” the path to a local file in the repository has to be stored. Every new construct has to be given a unique name too. Like with adding sensors, it is not allowed to have two or more sensors or constructs with the same name.

After clicking the “Add” button, the new construct will be stored to the versioning system. This is realized in the background with using windows Services. Windows Services are described in more detail at the end of that chapter.

The operation can also be aborted by clicking the button “Cancel”. Then the start screen shows up again. After clicking the button “Add”, the result of the link is saved in a .txt-file.

²⁵ Code Project, <http://www.codeproject.com/Tips/381509/Math-Parser-NET-Csharp> (May 2013)

The timestamp and the calculated value are saved, separated by a semicolon, as figure 26 shows.

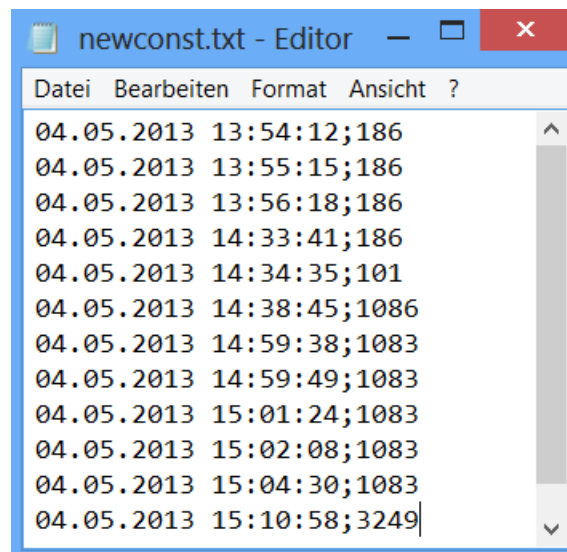


Figure 26 - New constructs

Windows services are applications that run in the background of the operating system Windows. They can be started automatically by booting the system, manually from a user using the Service control panel applet or by an application that uses service functions. They can execute even when no user is logged in the system [14].

In the prototypical application, windows services are used to store meta-constructs to a version control system in the background. The service functions as a proxy to the version control system, like figure 27 shows.

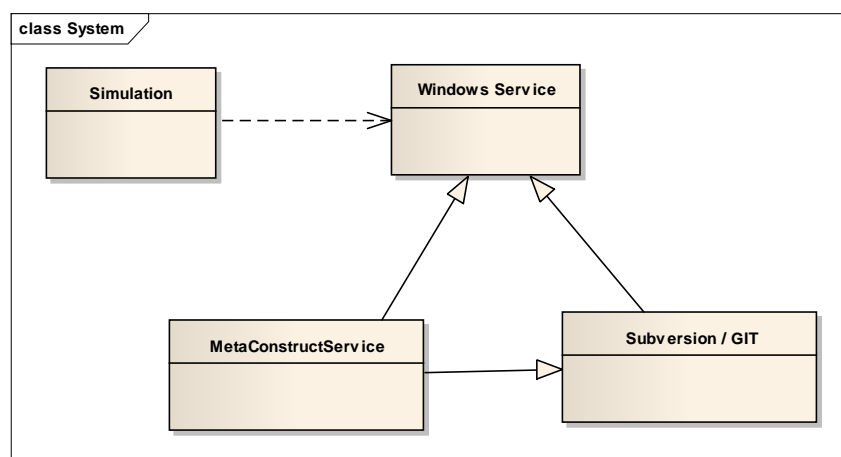


Figure 27 - Proxy

For each new construct, a new service is installed and started. Depending on the settings of the application, either a service using Apache Subversion or Git is started.

9.5. Visualization of data

Figure 28 shows how data can be visualized. One or more sensors or constructs have to be selected before clicking the button “Show” on the start screen.

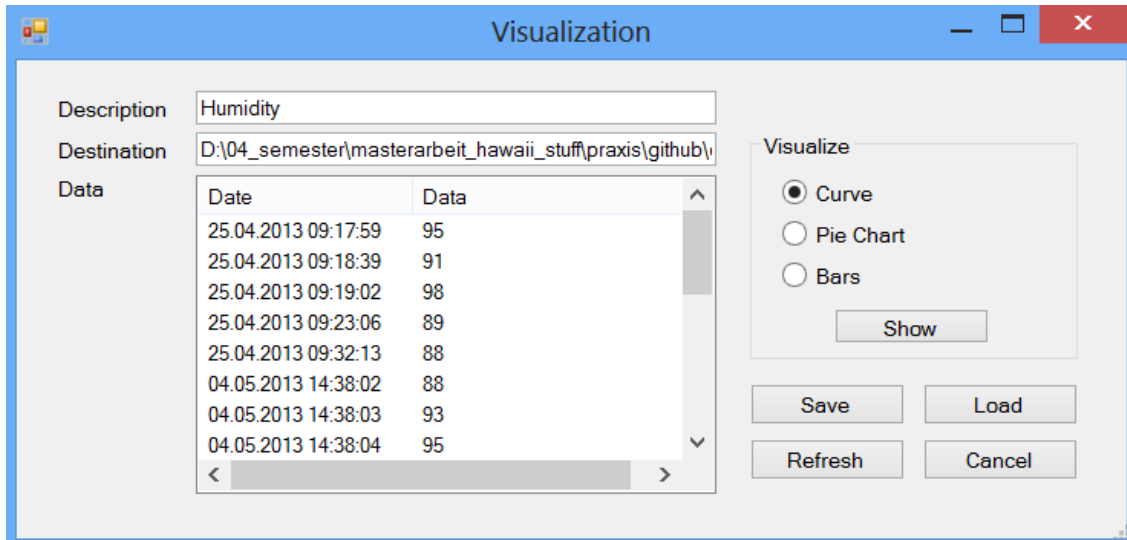


Figure 28 - GUI for data visualization

Per default, by opening the dialogue, the name of the selected sensor is displayed in the field description.

If only one sensor or construct is selected, the destination and data fields are filled with values. The destination space displays the local path where the sensor or construct file is stored. In the section data, the content of the sensor file is displayed. If more than one sensor or construct should be displayed, this section looks like figure 29. Most of the time it is necessary to compare more than one data stream with others.

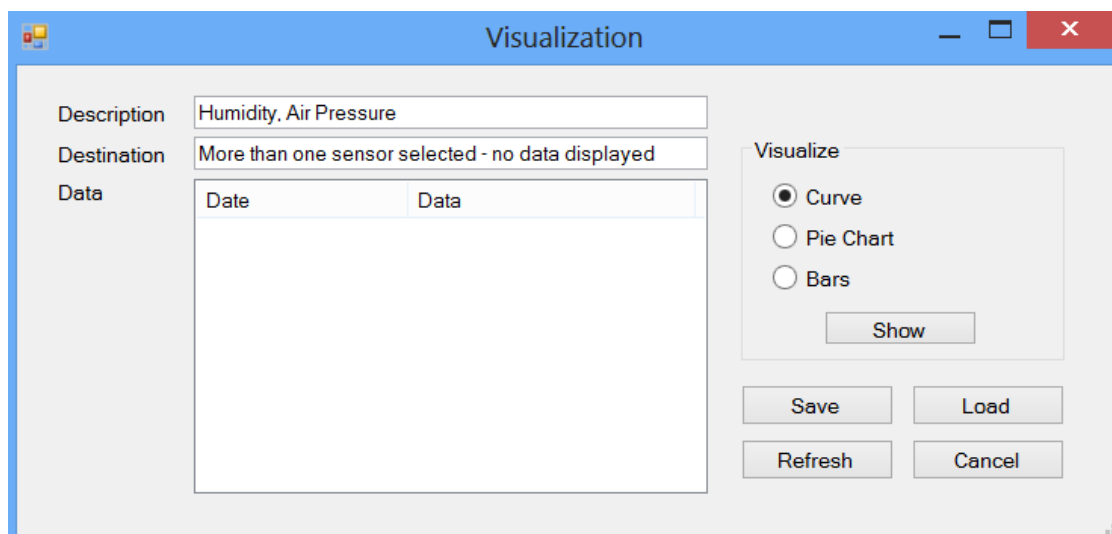


Figure 29 - GUI 2 for data visualization

After pressing the button “Refresh”, the sensor file is imported again and the data section is updated.

For visualization three options are available: a Curve, a Pie Chart or the Bar view. After selecting one option and pressing the button “Show” a new window with the visualization opens.

Figure 30 shows the graphical representation of the test data in a curve. On the x-axis the time intervals are displayed (distance 1 between each interval). On the y-axis the value is displayed.

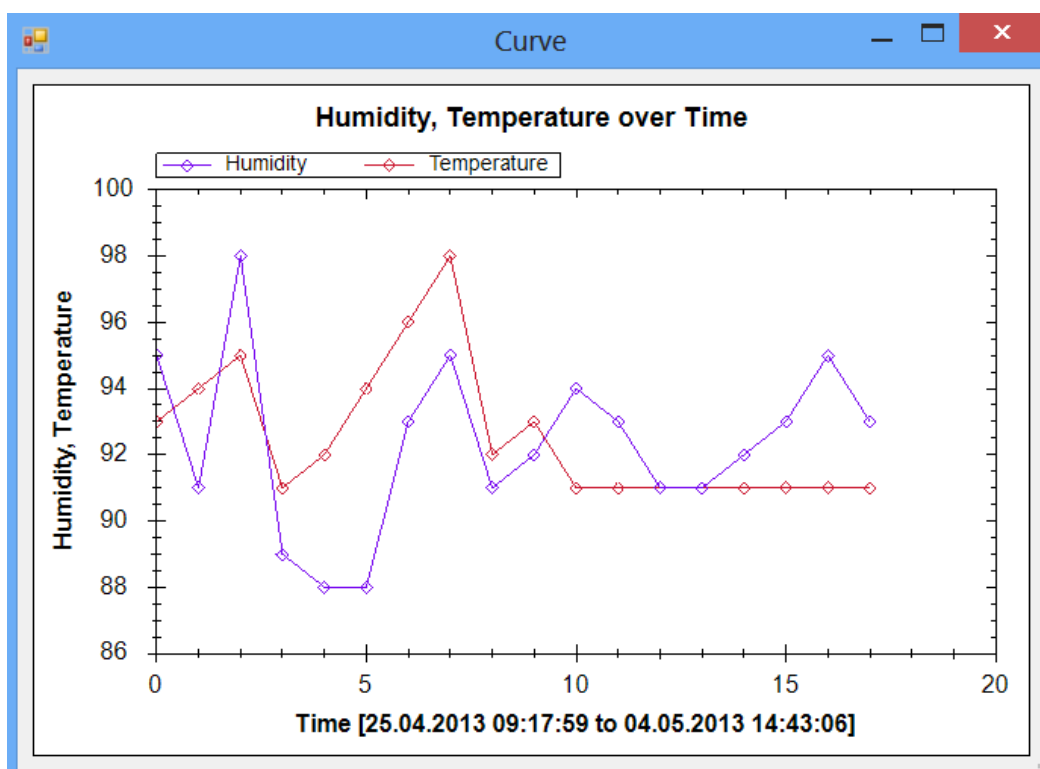


Figure 30 - Curve

The example shows a comparison between the humidity and temperature from the test data. Scientists can so easily compare different sensors and find new connections in them (e.g. for new constructs).

Figure 31 shows the visualization option “Pie Chart”. The pie shows how often each value occurs.

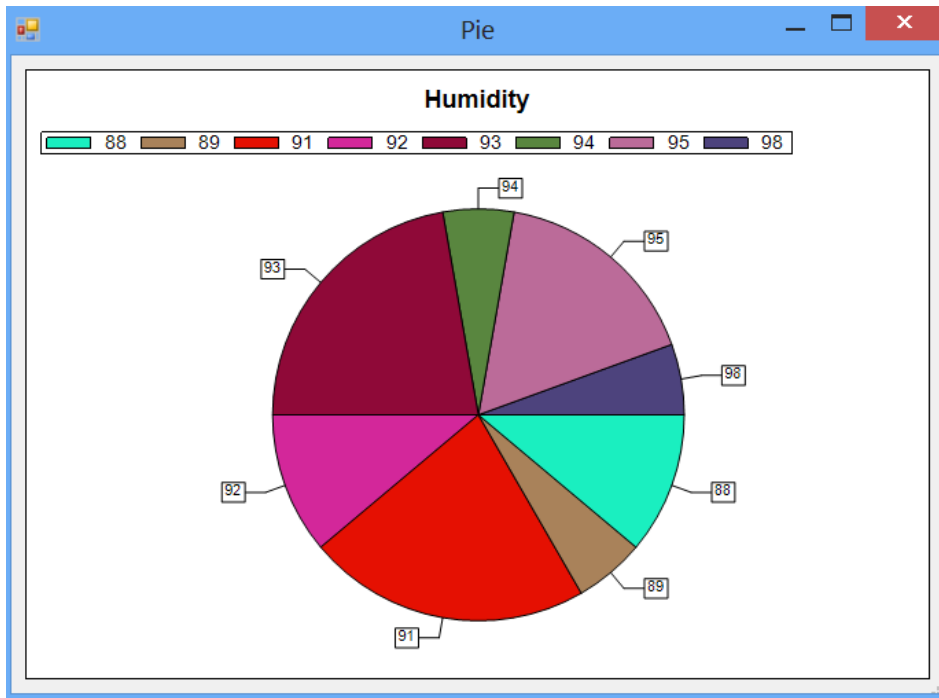


Figure 31 - Visualization: Pie Chart

In figure 32 the bar view is displayed. In this example, the values of humidity and temperature are compared.

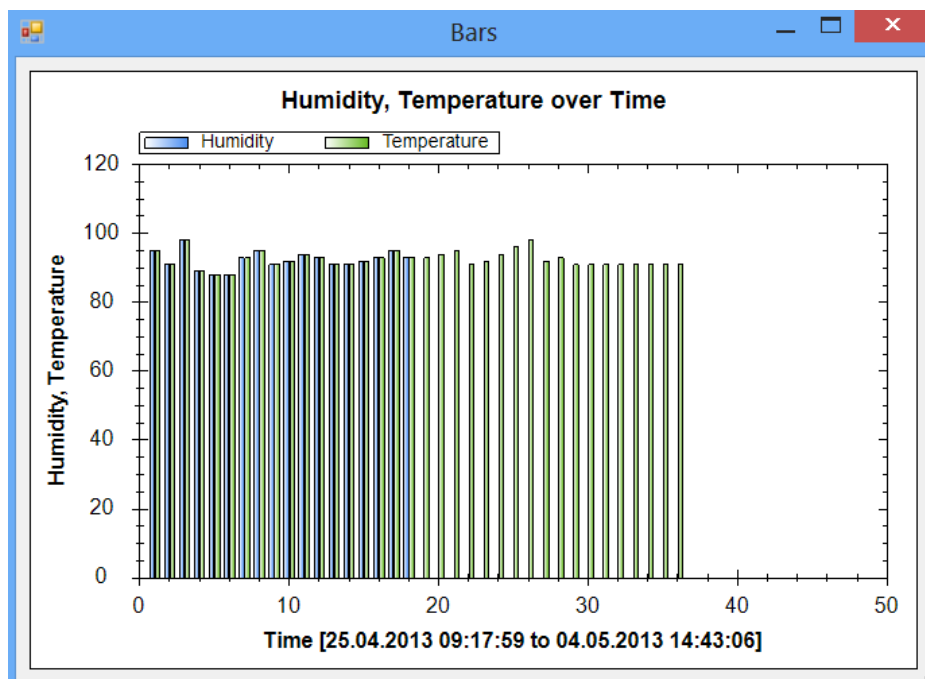


Figure 32 - Visualization: Bars

The graphical representation has been realized using a .NET library called “Charting” from the website Code Project²⁶.

It is also possible to save visualizations. This makes sense, if one scientist wants to store his results and show them to one of his colleagues (see user story for requirement 5 in chapter 8.1.2).

After clicking the button “Save” the window displayed in figure 33 opens.

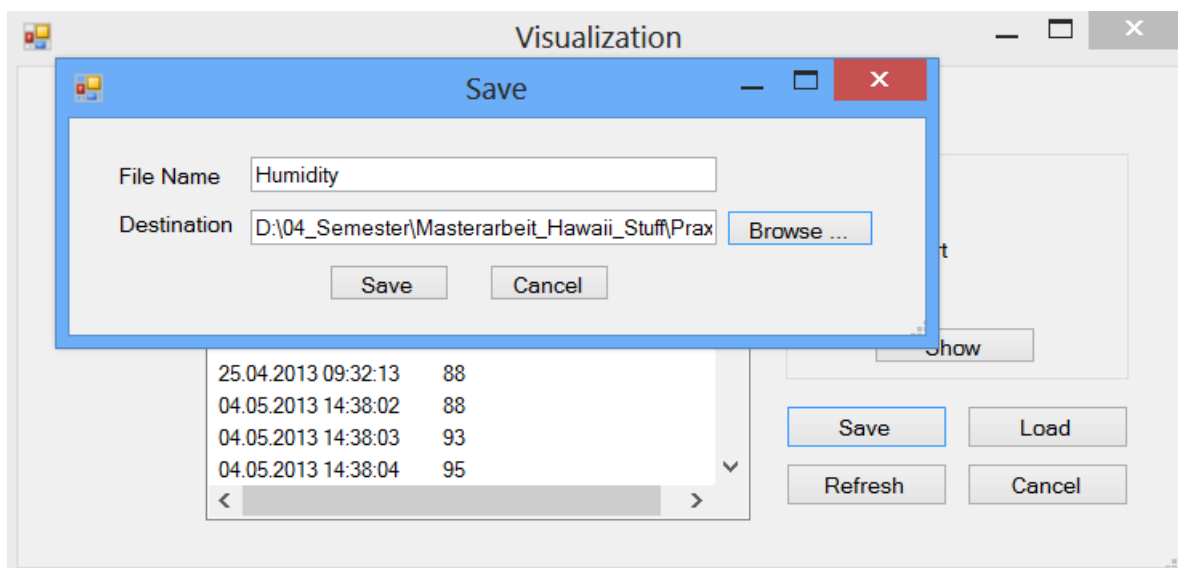


Figure 33 - GUI for saving visualizations

A file name and a destination for storing the visualization can be chosen. After clicking the button “Save” the needed data are stored in a .txt-file on the local file system.

If another scientist wants to load a stored visualization again, the button “Load” has to be clicked like represented in figure 34. It is only required to know the path to the stored file.

²⁶ Charting, <http://www.codeproject.com/Articles/5431/A-flexible-charting-library-for-NET> (May 2013)

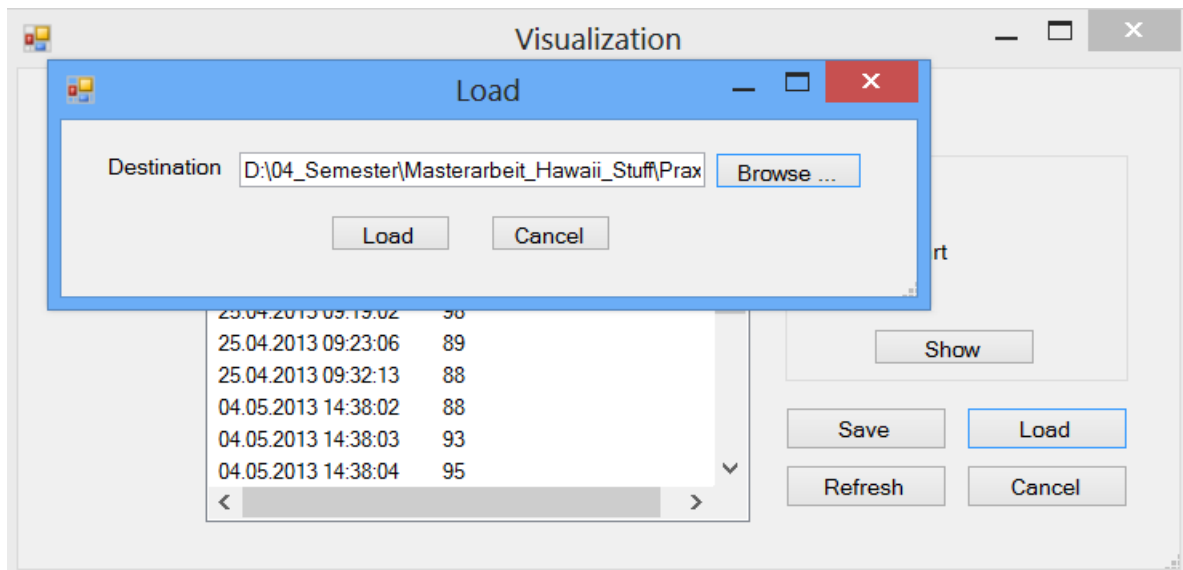


Figure 34 - GUI for loading visualizations

The loaded sensors have to have a unique name. If this is not the case, the loading fails with the error message displayed in figure 35.

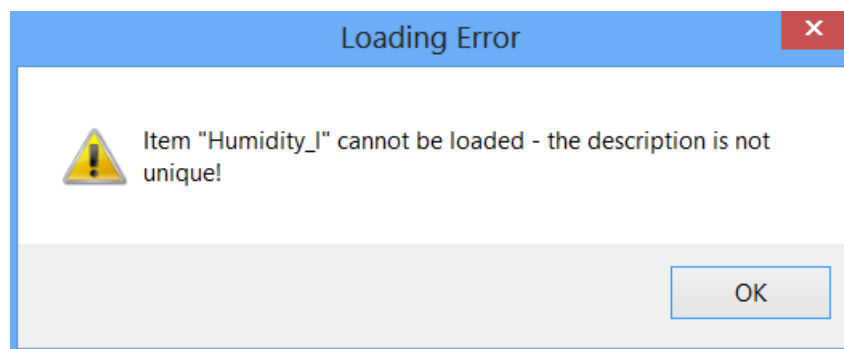


Figure 35 - Alert box for unique key violations

The existing sensor has to be renamed or a new name for the loaded sensor has to be chosen and the transaction can be started again.

9.6. Settings

Figure 36 shows the mask for saving the settings of the current simulation.

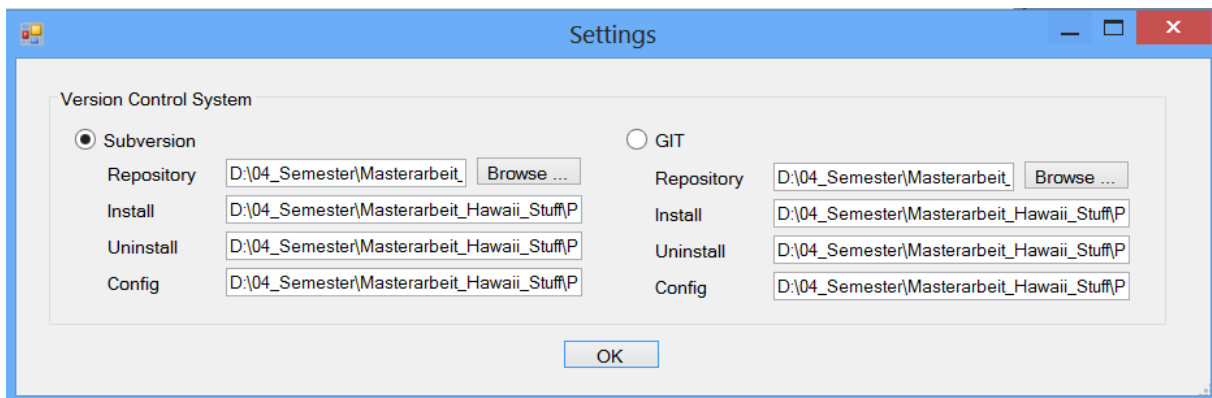


Figure 36 - Settings

Either Subversion or Git must be used in the simulation. A mix of the two versioning systems is not possible.

If the user chooses the option “Subversion”, a Subversion repository, a path to the install-, uninstall- and config-files have to be stored. All this values are stored as default settings by opening this window. The install-, uninstall- and configuration-file point to the currently using windows service.

When using the Git option, the same values are also presented as default. The only difference in that option is that the version control system Git is used and therefore a different windows service is started up.

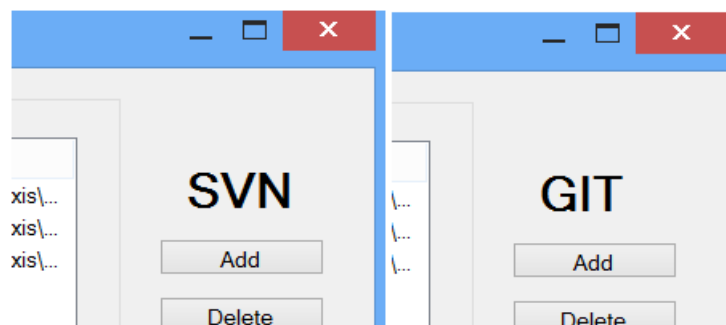


Figure 37 - Start Screen VCS

The current setting is displayed on the start screen on the top right corner like figure 37 shows.

9.6.1. Subversion

When adding a new construct to the sensor using Subversion, a new windows service especially for that version control system is installed and started. This process can be viewed in the windows service control panel like figure 38 shows.

Dienste (Lokal)					
SimulationSVN_Humidity_Temperature					
	Name	Beschreibung	Status	Starttyp	Anmelden als
	Sicherheitskonto-Manager	Durch den Sta...	Wird ausgeführt	Automa...	Lokales System
Den Dienst beenden	SimulationSVN_Humidity_Temperature		Wird ausgeführt	Manuell	.\Tantschi
Den Dienst neu starten	Skype Updater	Enables the d...		Automa...	Lokales System

Figure 38 - Service Subversion

The windows service for Subversion uses the C# library Sharpsvn. The last line of each sensor text file is read. The value is connected according to the link definition and calculated using the library “Math Parser .NET C#”. This workflow is illustrated in figure 39.

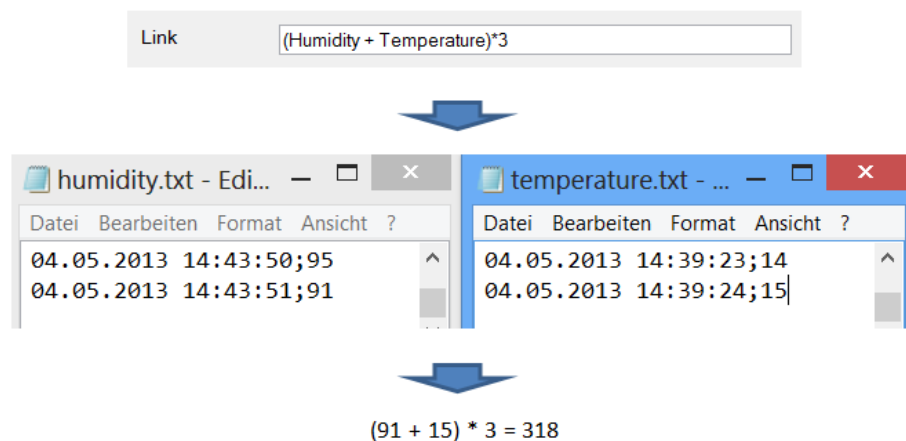


Figure 39 - Workflow of linking sensors

By evaluating the software solution using Subversion following problems and issues have been encountered.

Subversion is very slow. It takes quite a long time to commit to a remote repository. The commit time of Subversion and the timestamp of each sensor file can be very far apart.

The solution of using the last line of each .txt-file is not failure free. There are mistakes in the result, if one sensor file gets updated every 10 seconds and the second one every minute.

If a user is working on the local Subversion repository while the sensor is committing, locks can easily occur (Failure message: SVN_ERR_WC_LOCKED). This can be resolved with making a Subversion “Cleanup” of the project folder manually.

9.6.2. Git

If the user links a new construct using Git as preferred version control system, the workflow is similar to Subversion.

A Windows service using Git is started, like figure 40 shows.

The screenshot shows the Windows Services console for 'Dienste (Lokal)'. A table lists several services, with 'SimulationGIT_New Construct' highlighted in blue. The table has columns for Name, Beschreibung, Status, Starttyp, and Anmelden al:.

Name	Beschreibung	Status	Starttyp	Anmelden al:
Sicherheitskonto-Manager	Durch den Sta...	Wird ausgeführt	Automa...	Lokales Syste
SimulationGIT_New Construct		Wird ausgeführt	Manuell	.\Tantschi
Skype Updater	Enables the d...		Automa...	Lokales Syste
Smartcard	Verwaltet den...		Deaktivi...	Lokaler Dien:

Figure 40 - Service Git

Following issues have been discovered using the version control system Git:

Changes are committed to the local repository. Thus, this solution is very fast. There is no big time difference between the .txt-timestamps and the commit-file. Nevertheless, the windows Git client doesn't support to push to the remote repository. This has to be done manually. This has the advantage that the user can control the file before it is pushed to the remote repository. Making changes afterwards is very complicated with the Subversion solution.

Concerning the fitting of sensor data, the same problems occur as with the Subversion implementation. Git is more stable than Subversion. Locks of the working repository nearly never occurred during testing.

10. Results

The results of the actual implementation using Subversion and Git are described in the following paragraphs. There is made a comparison between Subversion and Git as well a comparison between version control systems in general and databases.

10.1. Comparison of Subversion and Git

Table 6 shows a quick overview of the advantages and disadvantages of the two used version control systems.

	Subversion	Git
Advantages	<ul style="list-style-type: none"> • Save to remote repository: immediate backup 	<ul style="list-style-type: none"> • Speed • Reliability • Check new constructs on local file system before pushing to remote repository
Disadvantages	<ul style="list-style-type: none"> • Working copy locks occur more often • Difficult mapping between .txt-files (which lines belong to each other?) • No checking of new constructs possible before pushing to the remote repository 	<ul style="list-style-type: none"> • Difficult mapping between .txt-files (which lines belong to each other?) • No direct pushing to remote repository possible. Push has to be made manually with using a SSH-key.

Table 6 - Comparison Subversion and Git

These results have already been described in more detail in chapter 9.6.1 and 9.6.2.

In general, there are not many differences between the two version control systems. The only huge difference is the centralized and decentralized architecture which has been introduced in the theoretical part of this paper.

Regarding the result of this thesis, Git is a better solution for using version control systems for real time geo-sensor networks. The system is faster and the user has the possibility of checking and controlling of new meta-constructs before pushing them to a remote directory.

10.2. Comparison between VCS and databases

Nevertheless, the use of no database brings several disadvantages.

- Selection of the data
General approved methods of selecting data (SQL) cannot be used when working with .txt-files. It is difficult to know, which lines of data belong to each other.
- Standards
Standards concerning geo-data are referring to the use of databases or XML-schemas. They cannot be applied to .txt-files which are stored in a versioning system.
- Big data
Databases are built for handling a huge amount of data. Version control systems are built for managing software development history. Thus, databases work better with big amounts of data.
- Connection to GIS
Nearly every available geo-information system provides a connection (interface) to a database.

But there are also advantages using a version control system for processing of geo-data.

- View log
Despite for databases, it is very easy to show the history of several versioning branches. A log overview is implemented in Subversion and Git, like figure 41 shows.

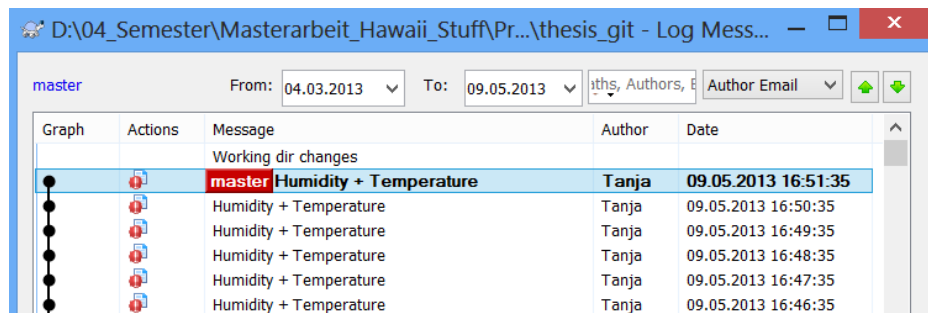


Figure 41 - Show Log

- Meta Constructs

The storing of meta constructs can be easily implemented using versioning systems, like the practical implementation described in chapter 8 and 9 shows.

- Costs

Most of the versioning systems are available as open source. Thus, they are much cheaper to implement than a database solution.

- Lightweight implementation

Version control system implementations don't need a lot of resources. Most of the systems offer a library for integrating them in common programming languages. In a lot of companies version control systems are already used for code management. So it is easy to integrate new software using version control systems.

There appear different advantages and disadvantages with version control systems and database solutions. As result of this thesis no solution is better than the other, for each application type a suitable implementation has to be chosen.

11. Conclusion

The key objective of this thesis was to find out, if data management concerning analyzing near real-time data streams can be improved by using version control systems to store data. Version control systems are a good alternative for smaller implementations. Using version control systems instead of data warehouses has the advantage of a lightweight implementation. Most of the version control systems are available as open source, thus the costs of implementation are much lower. It is also simple to integrate a new sensor in the system with version control systems. The user just has to make a commit of a sensor file, it is not necessary to adjust data to a specific database schema.

Moreover, version control systems offer good possibilities for data aggregation and data fusion. For example different data streams can easily be connected to each other and stored in the system as new branches. The systems already offer options to check the history of branches and to compare branches to each other.

Programming software which uses version control systems is not too difficult, because most of the systems are available as open source and they offer libraries to integrate them in common programming languages. Besides, a lot of companies already use version control systems for their code management. This makes it simple to integrate those systems in a company environment.

It does not make a big difference which version control system is used. The thesis made an evaluation of different existing systems. The only big difference is the distinction between centralized and distributed systems. It depends on the preferences of a user which system he or she should choose. In general, nearly every version control system on the market is suitable for a geo-sensor implementation.

Thus, for small sensors or solutions that are not used worldwide, version control systems are a suitable solution. But, if exchanging protocols with other institutes, a solution based on standards (like from the OGC) is the better alternative. They are all based on xml- or database standards and offer better integration options to other software parts. Moreover, with using a database it is easier to query data with tools like SQL. Anyway, all that options come with higher costs and a much more difficult integration in a company environment.

As new research questions, after having the results of this thesis, the following can be formulated:

How can geo-sensor data that is stored in version control systems be integrated to existing GIS?

How can geo-sensor data that is stored in version control systems meet common geo-sensor standards (e.g. standards from the OGC)?

How can querying geo-sensor data that is stored in version control systems can be made easier?

In conclusion it can be said that it depends on the kind of purpose of the used geo-sensor data to decide which possibility is the best. If a lightweight, cheap and easy-to-integrate solution that is not shared worldwide is needed, version control systems are a good alternative.

Bibliography

- [1] Nittel, S. (2009). A survey of geosensor networks: Advances in dynamic environmental monitoring. *Sensors*, 9(7), 5664-5678.
- [2] Grosky, W. I. et al. (2007). SenseWeb: An infrastructure for shared sensing. *Multimedia, IEEE*, 14(4), 8-13.
- [3] Lemmens, M. (2011). *Geo-information: Technologies, Applications and the Environment*. New York: Springer.
- [4] Uckelmann, D., Harrison, M., & Michahelles, F. (2011). *Architecting the internet of things*. New York: Springer.
- [5] Nittel, S. (2008). *Geosensor Networks: New Challenges in Environmental Monitoring using Wireless Sensor Networks*. Available Online: http://www.academia.edu/2791581/Introduction_to_advances_in_geosensor_networks [cited 17.08.2013]
- [6] Duckham, M., Nittel, S., & Worboys, M. (2005). Monitoring dynamic spatial fields using responsive geosensor networks (pp. 51-60). Presented at the *13th annual ACM international workshop on Geographic information systems*. 31 October-5 November 2005, Bremen, Germany.
- [7] Nittel, S., et al. (2004). Report from the first workshop on geo sensor networks. *ACM SIGMOD Record*, 33(1), 141-144.
- [8] Gross, N. (1999). *Businessweek Online*. Available Online: http://www.businessweek.com/1999/99_35/b3644024.htm [cited 17.08.2013]
- [9] Brain, M. (2004). *How Motes Work*. Available Online: <http://computer.howstuffworks.com/mote.htm> [cited 17.08.2013]
- [10] Dishongh, T. J., McGrath, M., & Kuris, B. (2009). *Wireless sensor networks for healthcare applications*. Norwood: Artech House.
- [11] Resch, B. (2009). *Live Geography - Standardised Geo-sensor Networks for Real-time Monitoring in Urban Environment*. Doctoral dissertation, University of Salzburg.

- [12] European Commission (2012). *Copernicus: new name for European Earth Observation Programme*. Available Online: http://europa.eu/rapid/press-release_IP-12-1345_en.pdf [cited 17.08.2013]
- [13] Laplante, P. A. (2004). *Real-time systems design and analysis*. Canada: Institute of Electrical and Electronics Engineers.
- [14] Resch, B. et al. (2009). Urban Sensing Revisited – Common Scents: Towards Standardised Geo-sensor Networks for Public Health Monitoring in the City (pp. 16-18). Presented at the *11th International Conference on Computers in Urban Planning and Urban Management - CUPUM2009*. 16-18 June 2009, Hong Kong, China.
- [15] Opegeospatial Consortium (2013). *OGC Standards and Supporting Documents*. Available Online: <http://www.opengeospatial.org/standards> [cited 17.08.2013]
- [16] Kleine, M., Hirschfeld, R., & Bracha, G. (2012). *An abstraction for version control systems*. Potsdam: Universitätsverlag Potsdam.
- [17] Chacon, S. (2009). *Pro Git*. New York: Apress.
- [18] Kemper, C., & Oxley, I. (2012). *Foundation Version Control for Web Developers*. New York: Apress.
- [19] Haining, R. P. (2003). *Spatial Data Analysis: Theory and Practice*. Cambridge: Cambridge University Press.
- [20] Buja, A., Cook, D., & Swayne, D. F. (1996). Interactive high-dimensional datavisualization. *Journal of Computational and Graphical Statistics*, 5(1), 78-99.
- [21] Blakemore, M. (1986). Geographical Information Systems. In R. J. Johnston et al. (Eds.) *Dictionary of Human Geography* (p. 18). Oxford: Blackwell.
- [22] Burrough, P. A. (1986). *Principles of Geographic Information Systems for Land Resources Assessment*. Oxford: Clarendon Press.
- [23] Antenucci, J. C. et al. (1991). *Geographic information systems: A guide to the technology*. New York: Springer.

[24] Melnick, A. L. (2002). *Introduction to geographic information systems in public health*. Maryland: Aspen Publishers.

[25] Fazal, S. (2008). *GIS Basics*. New Delhi: New Age International.

[26] Berry, J. K. (2013). *Beyond Mapping III*. Available Online: <http://www.innovativegis.com/basis/mapanalysis/> [cited 17.08.2013]

[27] Davis, B. E. (2001). *GIS: A visual approach*. Albany: OnWord Press.

[28] Krek, A. (2006). Geographic Information as an Economic Good. In M. Campagna (Ed.), *GIS for sustainable development*. Boca Raton: CRC Press.

[29] Radl, P. (2013). *Private Wetterstation 61169 Friedberg/Hessen*. Available Online: <http://wetter61169.de/download/> [cited 17.08.2013]

List of Abbreviations

NesC	network embedded systems C
MEMS	micro electro-mechanical systems
SDI	spatial data infrastructure
INSPIRE	Infrastructure for Spatial Information in the European Community
GEOSS	Global Earth Observation System of Systems
GMES	Global Monitoring for Environment and Security
SEIS	Shared Environmental Information System
GSDI	Global Spatial Data Infrastructure Association
USGS	U.S. Geological Survey
W3C	World Wide Web Consortium
OGC	Open Geospatial Consortium
SWE	Software Web Enablement
SVN	Apache Subversion
VCS	Version Control System
GIS	Geographic Information Systems
Sensor ML	Sensor Model Language
O&M	Observations & Measurements
TML	Transducer Model Language
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SAS	Sensor Alert Service
WNS	Web Notification Service
M2M	machine-to-machine
ESRI	Environmental Systems Research Institute
GRASS	Geographic Resource Analysis Support System