A Key-Value-based Persistence Model for Sensor Networks

A report presented to the faculty of
San Francisco State University
In partial fulfillment of
The requirements for
The degree

Master of Science
In
Computer Science

by

Marcello Alves de Sales Junior

San Francisco, California

December 2009

Copyright © 2009 by Marcello Alves de Sales Junior

CERTIFICATION OF APPROVAL

I certify that I have read *A Key-Value-based Persistence Model for Sensor Networks* by Marcello Alves de Sales Junior , and that in my opinion this work meets the criteria for approving a culminating experience submitted in partial fulfillment of the requirements for the degree: Master of Science in Computer Science at San Francisco State University.

Prof. Arno Puder, Ph.D.
Professor of Computer Science

Prof. Marguerite C. Murphy, Ph.D.
Professor of Computer Science

A Key-Value-based Persistence Model for Sensor Networks

Marcello Alves de Sales Junior
San Francisco, California
2009

Sensor Networks are becoming important to different scientific and industrial communities because of what they produce: the raw data of diverse domains. In order to make use of the collected data, researchers may have to dissect the characteristics of the sensor network in question, regarding different properties such as the purpose and location of the observed data, as well as how the data is described. For this reason, this work's first contribution is a set of data persistence taxonomies based on the state of the art of Data Persistence for Sensor Networks, which can be used to classify the properties of the produced raw data. In order to evaluate the proposed taxonomies, a data persistence component for NetBEAMS was designed and implemented. NetBEAMS is a component-based sensor network infrastructure developed to improve the operation of the SF-BEAMS environmental sensor network. SF-BEAMS is deployed in Tiburon, California, and managed by the Romberg Tiburon Center (RTC). Based on an empirical analysis regarding proposed taxonomies, a Key-Value Data Model is proposed as an alternative to the Relational Data Model traditionally used. Furthermore, the mongoDB database, a schema-less document-oriented database, was selected for evaluation. Results based on the experiments suggest a novel approach to provide External or Data-Centric persistence for networks. Similarly, the literature supports programming languages as a better abstraction when it comes to data access and modification by non-technical users such as biologists. Finally, this report lends itself to future work in the area of data persistence in sensor networks.

Keywords: Environmental Sensor Network, Data Persistence, Taxonomies, KVP Data Model, Document-Oriented Databases, KVP Databases, SF-BEAMS, NetBEAMS I certify that the Abstract is a correct representation of the content of this report.

Chair, Thesis Committee Date

ACKNOWLEDGMENTS

"The brick walls are not there to keep us out. The brick walls are there to give us a chance to show how badly we want something. Because the brick walls are there to stop the people who don't want it badly enough." Dr. Randy Pausch

As Dr. Randy Pausch's last lecture reminded me after the first quarter of the program, there are purposes for the difficult things in life as the brick walls will always be standing one after the other in course of your life. As my childhood dream was to become a scientist and study in America, I can say that I might have crossed a good percentage of those brick walls. This program gave me the wings necessary to keep dreaming and believing that anything is possible. You've just got to dream.

I have been fortunate in these twenty-four months to meet many people who have given me more of their personal and professional time, their companionship, their help, and above all, their patience. They all contributed, directly or indirectly, for the success of this work, which I kindly refer to as one of my childhood dreams.

The academic professors were fundamental in different moments during this program. First, I would like to thank my adviser, Dr. Arno Puder, who not only had to put up with the times my laptop PC broke and I delayed the delivery of components, but also the times I was just late for the meetings. He was the first one who believed on my work in the program and accepted me in the team. A big thanks to him, someone whose expertise and professionalism I have learned to admired. Dr. Dragutin Petkovic also played an important role throughout my life during the program, advising me about work schedule, interviews, etc. Finally, Prof. Murphy offered the hardest classes of the program, and I must say that my writing skills were improved because of her class.

There are four fellow researchers whom I would like to specifically thank: Kleber Sales, who have developed the core of NetBEAMS, and for many times explained and helped me during development. Many were the afternoons and weeks of revisions to have a working demo. Teresa Johnson was also important for the development of NetBEAMS and provided her feedbacks. William Murad helped me a lot during the classes of Biology and Algorithms, while Ivan Weiss was a great co-worker during the Operating Systems class developing TOS as one of the best teammates during school time.

A special word of gratitude goes to the ones who made everything possible. My co-workers from CollabNet, in special for Richard, Jack, and Adam because their great recommendations, and to Connie, Edgar, Yiping, Norah, Kathryn and Andres (Facebook) who I had more contact with during the first and second times I worked over there as a full-time and an Intern, respectively. Subversion is part of my life today! Also, Eliot from 10gen for the innumerous days and nights on the #mongodb IRC channel and mailing list, helping me figure out the value of mongoDB. This work was only presented because of the help of three VERY important people I believe are important to me: Niki is the angel that appears from nowhere to salve you! I decided to start the program because I was very welcomed by her. During the entire program she was always there for each of us, I am sure, and now towards the end she stepped up and started reading and fixing the entire chapters. Niki, there are no words to thank you. In addition, Ivan also kindly offered his help to review the technical chapters for the gramatical errors. Man, I'm very lucky to have you around. And finally Kleber, who suffered with the first drafts of this work by reading so many errors and gramatical mistakes. I thank him a lot for the help and I will definitely have a better version for him from now on!

Moving towards to more personal acknowledgements, I would like to execute a big of annotated thanks towards all my family and friends. They are roommates, friends, landlords, distant friends, that I have lived with, had been with, etc. These showed they cared about me (at least I felt it) in what I wanted to accomplish. My gratitude to people here in the California goes to Rhea, Sonia, Demian and Maria, Eduardo, Paula, Amanda and Eric, Leanna, Ivan, Julie, Diane and in special to Michelle, who believed on me since even before being in the program. The ones in Seattle who are also worth mentioning are Eduardo and Renata, for receiving me during the attempts to interview with Microsoft, and Wade and Fran for their friendship and support. My second-degree cousin Thamires with whom I have enjoyed hanging out in the night life of San Francisco! When it comes to the East Coast, a lot of important people are in New York and New Jersey. Patricia is always available to talk about family, all the guys from the Focolare Movement, who I haven't been physically in contact, but online. Ricardo for being the best roommate and close friend, and finally Regina from Boston, who have belived in my potential since I first arrived in this country. Michael Aloi was essential throughout my career and for becaming a great friend. In Brazil, both professor Evandro Costa and Alexandre Vasconcelos for the letters of recommendation and for once again believing in my potential.

I am, for sure, particularly indebted to my parents and my brothers for their monumental, unwavering support and encouragement during these months of dedication. For all my anties, uncles, cousins (in special Kaline, Nicinho and Fernanda), they all have truly always been there for me since even before the program started. Without my mom and dad, or my brothers Leandro and Thiago, none of this dream would have been even possible remotely. I love you all and I will always remember your support during these two years of less phone calls and visits. Some brick walls were hard to climb, some of them were even more difficult, but none of them were impossible, because I have faith in the God of the impossible, who helped me stand up when I fell so many times. That's why I thank God Jesus Christ for bringing another of my oldest childhood dreams alive.

"It's not about how to achieve your dreams. It's about how to lead your life. If you lead your life the right way, the karma will take care of itself. The dreams will come to you." Dr. Randy Pausch

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Sensor networks are commonly used in different areas for different purposes such as accurate measurements during scientific research [RM04] and the general public access such as weather forecasting [ASSC02]. In the scientific community, they serve as tools to watch the state of the environment by storing samples of data at particular periods of time. For example, the NASA Jet Propulsion Laboratory uses the Volcano SensorWeb [SSK+08], a sensor network which aims at collecting data from individual volcanoes in Alaska and South Pacific to be used with heuristics regarding hazardous activities. Similarly, the National Data Buoy Center, a division of the National Oceanic and Atmospheric Administration (NOAA) [EB82], provides online information from different buoys located in different shores around the world regarding water quality, current information, etc, that characterizes sensor network for environmental monitoring [MHO04]. Therefore, sensor networks can be seen as an extremely valuable tool for the scientific and commercial sectors, whose interest is the use of the sensor's produced data.

The first motivation for the inception of this work was to provide a data storage system for NetBEAMS [PJS+09], a component-based sensor network infrastructure that can be used to automate the process of data gathering in sensor networks. In this way, state of the art of data persistence in sensor networks was reviewed for different approaches used to obtain access to the collected sensor data: interrogate sensor devices indirectly by accessing its data in places called network data sinks or simply by connecting directly to the device. While the collected data is only available in-memory as a data stream in the latter approach, it is stored in any sort of secondary memory device such as a hard-disk or a flash memory for archival purposes in the former approach. Along with the location of the collected data, the data model may also influence the process of reusing the collected data. In this way, the introduction of a persistence layer for existing sensor networks requires an understanding about its infrastructure, the nature and the location of where the collected data is temporarily located after it is produced from the sensor device. Similarly related to the infrastructure, the data model used to represent the collected data is another aspect of the data persistence layer implemented to manage the raw data values, and may raise questions. Considering that sensor networks devices may change over time, how can a data model take into account future additions of new entities to the database schema? Should such a solution take into account the end users' skills in database systems or provide access to the data through programming languages [BDD09]? These questions are faced when considering the primary users of the collected data such as Biologists from the the department of Biology at San Francisco State University, who are responsible for maintaining SF-BEAMS [RRGP06], a marine sensor network used by NetBEAMS. While access to the collected data may represent difficulties, other technical issues may also be faced such as growth in terms of the number of sensor devices deployed and, consequently, the amount of data produced. In this way, sensor networks engineered with the purpose of data archival must address questions related to how disk storage can be maximized when storing the collected data. Lastly, the way to read the data for reuse is the most striking feature of a sensor network, taking into account the different data formats used by the primary users of the system. How can individually collected sensor data be exported to other formats such as spreadsheets in order to be shared among research colleagues? For these and other reasons, the inception of a persistence storage for NetBEAMS, and any other sensor network, can be an exceptionally challenging problem to solve.

In view of the fact that the persistence storage of a sensor network imposes many different challenges, the assessment of the current state of the art for data persistence in sensor networks revealed different alternatives to better evaluate database technologies for sensor networks and, specifically speaking, to NetBEAMS. As a matter of fact, the first contribution of this work is the creation of a set of basic taxonomies for data persistence in sensor networks, proposed to drive an empirical analysis of different database systems used by researchers. In this way, based on the findings of the review of current trends in database systems [Bai09, Pri08] and distributed systems such as Cloud Computing [BYV$^+$09], this work proposes a data persistence solution based on mongoDB [mon09], a schema-less document-oriented database system. This selection was based after an empirical analysis of the different characteristics of the relational [Cod70] and the key-value pair [DHJ$^+$07] data models. After designing and implementing a solution whose technology has not yet been used in the sensor network community, experiments were conducted to simulate the workload of real-world scenarios used by SF-BEAMS during one year of data produced by a specific type of sensor. In conclusion, the implementation fulfills most of the requirements and specifications defined by the taxonomies, giving an alternative data model that solves the problem related to constant schema changes, providing a good support for Data Provenance [LNH05] for sensor networks. In addition, the solution provides different data access alternatives such as the use of programming languages or REST Web Services [Fie00] instead of the Structure Query Language (SQL) [Mel96] used in relational databases [Cod70], as well as native support to export data to formats used by the SF-BEAMS staff.

This paper is organized as follows: the next chapter describes the literature review of the state of art on sensor networks and data persistence, providing the underlying foundation for the design of taxonomies in data persistence for sensor networks in Chapter 3. Then, Chapter 4 examines the selected case study and its requirements for a persistence layer, whose technological properties are evaluated against the developed taxonomies in Chapter 5. The culmination of this work is Chapter 6, which details the design and implementation of a data persistence solution for the selected case study. Likewise, Chapter 7 discusses the solution by presenting the measurements and results of the conducted experiments. Lastly, Chapter 8 presents the conclusions of this project, together with the different possible future work not addressed by this report. Chapter 9 presents the appendix with source-code implemented, as well as logs collected from experiments.

Chapter 2

State of the Art of Data Persistence in Sensor Networks

In general, this chapter reviews surveys and related papers on sensor networks, providing a basic introduction about their characteristics to a new audience. Similarly, it reviews the characteristics of related works that discuss and provide data persistence solutions for sensor networks, characterizing the different approaches used to described and model the gathered data from sensor devices.

2.1 Survey in Sensor Networks and Its Infrastructure

According to [ASSC02], sensor networks are specialized types of network systems comprised of nodes with specific goals such as to produce and collect data. The former is seen as a sensor device capable of producing data to the network based on an observation from the environment. On the other hand, the latter is described as the final destination of the raw data in a sensor network and called the network data sink, or simply data sink, whose responsibility is to receive, categorize and store the "sensed" data. In this way, the location of the data sink is determined by the purpose of the collected data: data nodes are placed in-network close to the sensor devices when collected data has a life cycle, being discarded after being utilized. In contrast, data nodes are placed in a centralized node when the collected data is used for historical purposes.

The following sections describe general properties of sensor networks regarding infrastructure and components as described in the literature.

2.1.1 Properties of Sensor Device Nodes in Sensor Networks

Each sensor node performs different tasks for the network and, therefore, influences the method of deployment used. [RM04] identifies that the deployment of sensor devices can use arbitrary methods such as being dropped from an airplane, or manually placed at chosen geographic locations. Moreover, the cited authors also mentioned that the deployment can be done as a one-time event or as an iterative process, where the sensor devices are moved to different places as needed.

As noted by [RM04] sensor nodes can be moved around the deployment location, or remain in one place as a stationary sensor. In the case of a sensor network comprised with different moving sensors, a given sensor node can move occasionally or continuously, as a result of an accidental side-effect or a desired property of the device. As a matter of fact, these sensors are automotive or they can be attached to another moving object. For this reason, [RM04] classifies those types of mobility as active or passive.

Considering the physical design and capabilities of sensor devices, they range from microscopic to devices measuring a few cubic feet [ASSC02]. In addition to their limited size, their computational resources and power availability can also be affected, when they are located in areas of limited access to energy. In this way, [RM04] also distinguishes how sensor devices are usually designed to either store energy in batteries or use energy scavenged from solar cells. Even though a sensor network is designed for a purpose, [RM04] also explains that it can be comprised of different sensor devices playing different roles in what they observe in the environment, characterizing the heterogeneity of the sensor network by the hardware capabilities and what types of data each of them produce.

Finally, [RM04] mentions that sensor device nodes can communicate with each other using different media such as radio, diffuse light or inductive sound, influencing the type of protocols they use to exchange data. They can have knowledge of their physical location if they are integrated with a Global Positioning System (GPS) device, which gives the latitude or longitude coordinate values. Therefore, the cost of these devices may range from hundreds to thousands of dollars, depending on the design capabilities and functionality of a given device.

2.1.2 Properties of the Infrastructure of Sensor Networks

The last section discussed the different properties of the sensor devices when placed in a sensor network. Depending on how they communicate with each other, [ASSC02] states that sensor devices can be organized in a structured, or ad hoc way. For this reason, the infrastructure of sensor networks differ in topology such as single-hop, star, tree or graph. Each of these topologies may influence how the produced data from sensor nodes is collected and routed to their related data sink. While there are only two levels of node connectivity in sensor networks deployed as single-hop and star topologies, sensor networks whose topology is either tree or graph use multi-hop communication among each of their nodes for data communication from the devices to the data sink.

The property of coverage on a sensor network is defined by [RM04] as the physical space "visible" to the sensors devices, which is also directly related to the purpose of the network, its size in terms of the number of nodes deployed and the sensor network lifetime. Therefore, sensors are deployed in sparse or dense manners, according to the specification of what needs to be covered by the sensor devices. Furthermore, considering that some types of sensor devices are very likely to experience regular hardware failures, [ASSC02] described sensor network examples with redundant nodes that provide better data coverage. Finally, the coverage of sensor devices also has influence on the connectivity property of the sensor nodes. In order to communicate with redundant nodes, a physical connection must exist at all times. On the other hand, lower level of coverage is related to sporadic communication among the sensor nodes.

The sensor network size is related to the degree of coverage mentioned earlier, and it is measured by the number of devices deployed. [RM04] reports examples of sensor networks composed by different ranges of deployed sensor devices such as dozens, hundreds or even thousands of nodes. However, the lifetime of a sensor network determines how long these devices are going to be used, as well as how long the sensor network may exist. For example, an environmental sensor network [MHO04] with the purpose of collecting data for historic purposes may have their sensor devices continuously performing their data collection tasks. On the other hand, the lifetime of a sensor network designed to achieve a specific goal will not exist after the sensor devices finish their tasks. Finally, [ASSC02] argues that the performance related to the orchestration of sensor devices in sensor networks is related to the constraints defined for quality of service (QoS). This includes real-time data delivery within a threshold or for robustness such as maintaining live data even though a communication link is not available.

All things considered, sensor nodes play a vital role during the execution of a sensor network, whose infrastructure is characterized by how and where the sensor nodes are deployed and communicate with each other. The next section covers the approaches dealing with the problem of this type of system network.

2.2 Persistence Storage for Sensor Networks

As outlined in the last few sections, sensor networks are comprised of different types of sensor nodes. An important sensor node type is the data sink because it is where the raw data produced by sensor nodes resides. Data collection and access in sensor networks is directly related to the characteristics of the sensor network, its routing mechanisms and where the data sink is placed within the network.

The type of query mechanism used to directly or indirectly interrogate sensor devices in a sensor network is directly related to the aforementioned locations of the collected data. The in-network query processing is used when the location of the collected data is a local storage site [SLM06, SRK^{+}03] within the network; a strategy typically used to maximize the sensor's resources, while minimizing energy consumption [KGH07]. On the contrary, the a typical centralized query processing is used to search for values over the collected data when the location is a centralized data node [DML03].

When it comes to the data representation, the relational data model [Cod70] is one of the most traditional approaches used to describe the collected data of sensor networks, as it is reported in database systems such as the TinyDB [MFHH05]. For instance, [LKK07] proposes a modified version of the SQL language of TinyDB proposed to solve SQL Join problems, so that the query processing takes into account the location of the distributed data. Conversely, [LNH05] proposes the use of Data Provenance as the primary guidelines used to describe collected raw data from sensor devices.

The following sections describe general properties of sensor networks regarding data persistence, as reviewed in the literature.

2.2.1 Data Use and Where Data is Stored

As presented in the previous section, the raw data is the one in-transit from the sensor to the storage device, where it is labeled to be identifiable. It is also referred to as the collected data. Initially, the raw data serves as the main outcome of the sensor network. [LNH05] differentiates the use of **real-time data stream**, when they are generated from a sensor device whenever the data is requested. In this way, the data is still in its raw format specified by the sensor manufacturer. However, it is not retained. In contrast, the authors describe the data as **archivable**, when they a have historical significance and will be processed to be reused by other systems at a latter time. While the real-time data streams are used and potentially discarded, the archivable data is stored into secondary memory such as a hard-disk.

After producing the "sensed" values for the observed parameters from the environment, the sensor node is responsible for sending the raw data to a specific data node defined by the network infrastructure, as delineated in Section 2.1.2. Thus, [SRK$^+$03] describes that the destination for the collected sensor data will be one of two different types of storage devices: the **external storage** and the **local storage**. In the former type, the sensor sends the produced data to another sensor node that contains an external storage device. In the latter the data is stored only within the memory of the device before its reading as a real-time data stream. On the whole, the sensor network infrastructure dictates where the movement performed by the sensor data, for example, [DML03] sees a wireless sensor networks as a many-to-one data collection pattern, when its organization is based on single-hop or star. On the other hand, data may also flow from one intermediate sensor node to another until it reaches its final data sink when the data organization is structured as a graph, tree [SLM06, SRK$^+$03].

Lastly, the use of different external storage devices can be used to partition the collected data. This practice is called **Data-Centric Storage**, and aims at clustering the collected data from different sensor devices based on a given a property of the data such as location or time of creation.

2.2.2 Query Processing in Sensor Networks

The query processing in sensor networks is a direct or indirect method of accessing the collected data stored at data nodes, whose location is determined by different facts described in the previous section. [SRK$^+$03] describes an example of an **in-network query processing** when sensor nodes, acting as data nodes, provides real-time data stream as a response to queries requesting the device's collected data. Likewise, if the collected data is in use for a specific period of time, the data is cached in an external storage device in a close-by data node. In this way, when a query is issued to the sensor network with these characteristics, the query processor considers all data nodes before it returns its final response to the query. In comparison, a sensor network whose topology is in the form of a star, the data sink is usually located in the center of the structure, and thus, receives all the collected data from other sensor devices within the network. For this reason, it can be seen as a **centralized query processing**. Factoring in that a centralized data sink receives all the collected data from the peer nodes, this query processing strategy is responsible for responding to indirect queries about the produced data from the network.

Based on the literature review, there are advantages and disadvantages related to each of the query processing approaches described with regards to the infrastructure of the sensor network. [SRK$^+$03] argues that the primary purpose of the in-network query processing is to minimize the amount of data transmitted from sensor devices to a data sink, as sensor devices are generally limited by energy use. Considering that sensor devices are also used as data nodes, [KGH07] states that this approach can potentially flood all sensor nodes in the network, thereby decreasing its performance. For this reason, [SLM06] reports the use of the data-centric storage by organizing the collected data based on its particular property values. In this way, the query processor hits specific data nodes that contain the types of data in the range of the requested data concluding that this approach decreases the network congestion and maximizes the capacity on the sensor devices and the performance of the sensor network.

Unlike the in-network, the centralized query processing is related to a regular database system, located in one of the network nodes. For this reason, this indirect type of query processing is used when the data purpose of the data is for data storage. As a consequence, [DML03] shows that the result of a many-to-one communication between the centralized data sink and the other data nodes, this query process does not affect any of the sensor nodes in the network and, thus, maximizes the sensor device utilization. Moreover, this strategy better exposes the collected data to the primary users raw data, which follows the process of data processing, compression, aggregation, and so on [ZMEM07].

Although the approach of centralized query processing maximizes the sensor nodes capacity and minimize energy consumption, [DML03] explains that this trade off imposes the creation of the unavoidable development of a point of traffic concentration or the phenomenon of network congestion. For this reason, [KGH07] describes that different techniques used in distributed systems such as the use of master-slave data replication [KB09] or database partitioning [Sch06] can be used to reduce or mitigate the effects of this funneling.

2.3 Data Models and Query Engines for Sensor Networks

Different approaches are used to represent the collected data from sensors. Some research groups lacking database experience arbitrarily assign data descriptors, while others use database technology in a more pragmatic approach. [BDD09] describes several applications using various types of languages for accessing the data designed in different data models. [LNH05] reports on the use of a **Tabular Data Model** [SPBV08] using comma-separated values (CSV) to organize the measurements collected from sensor devices. In addition, [MFHH05] describes the use of the **Relational Data Model** [Cod70] as used in the majority of the projects that involves data persistence due to its popularity in areas such as Software Engineering and Distributed Systems. Finally, [HRN$^+$08, HRGL08] discusses examples of sensor network applications that make use of XML [xml08] to represent the collected data on a **Structured Data Model**.

The following sections describe the application of each of aforenamed data models in the context of sensor networks.

2.3.1 Tabular Data Model and Query

The most basic computer application to process data is through the use of spreadsheets, where data is organized into cells [SPBV08]. When exported to the so-called comma-separated values (CSV) file format, the resulting file is comprised of lines of ASCII characters, separated by a delimiter. The first line represents a header, with the identification of columns of specific data, where the following lines contain the values related to each top column. [LNH05] reports that the tabular data model is commonly used to provide data persistence for sensor networks. The SF-BEAMS environmental sensor networks [RRGP06] uses this approach to distribute the collected data over the Internet using the OPenDAP [ope08] data format.

This data model can be used in different contexts, as [LNH05] describes that the versions of the same document can be easily managed and shared among research colleagues using a version-control system such as Subversion [CSFP09].

2.3.2 Relational Data Model and its Query Mechanism

[LKK07] shows the use of the Relational Data Model [Cod70] as it is used in sensor networks to describe the collected data from sensor devices. The project described by the authors is based on the TinyDB database system [MFHH05], a database system specifically built for data nodes of sensor networks. It uses the same properties and constraints of regular database systems, where the entities of the sensor network are required to be modeled and represented by tables.

In order to query the data collected from sensor devices, the use of the Structured Query Language (SQL) [Mel96] is the main language used to extract the data managed by the system. As mentioned earlier, the default versions of this language have been modified to give support to the constraints of the location of the database such as the addition of new SQL operators for TinyDB [LKK07].

2.3.3 Structured Data Model and its Query Mechanism

XML is an increasing popular data format used widely by the applications developed for the Internet. [HRN^{+}08] reports an efficient way to represent the collected data from sensors using XML, while the authors of [HRGL08] describe a sensor network using XML for data model representation. In general, the collected data is structured into classes of relating elements and validated by an XML Schema [xml04].

In order to extract data from XML-based databases, the XML XPath language [xml07] is often used to query XML data. However, with the inception of hybrid XML-relational database, [SLWK07] reported the use XML as the data model and the use of SQL to query the stored data on the IBM's DB2 database system.

Other examples of the use of the structured data model using XML is found in the use of middleware, as the authors of [OCR^{+}09] reports. Such an approach is used for data exchange by the implementing of a middleware capable of query and extract the collected data from sensor devices [sn-08]. A practical example of the use of XML as a middleware to communicate between nodes is the communication mechanism used by NetBEAMS [PJS^{+}09]. See section the documentation at [dS09] for details.

2.4 Description of the Collected Data

As described in the previous section, the collected data from sensor networks can be represented by different data models, requiring the knowledge of the properties of the data. Not to mention that the low-level or raw data produced by a given sensor device might not include vital details regarding the origin and identity of the collected data. For this reason, the collected data is reported to be described in different ways such as Data Provenance or Annotations.

[LNH05] describes Data Provenance as a technique used to describe those essential properties of the collected data from sensors. One typical example of the application of Data Provenance is the so-called black-box of an aircraft, which carries the collection of spatial-temporal data structures collected from a group of sensors placed in different locations of an aircraft, as shown by [Cam07]. Finally, the author describes the collected data as used in the process of data recovery and analysis in order to identify the probable reason of an aircraft accident, by analyzing the time, location of the aircraft, and other independent variables from the sensors.

In general, sensor devices produce and preserve data onto their local storage or transmit them to data nodes as described earlier in this chapter. It is vital to understand that the majority of these devices output the raw data in streams of values with any type of delimitation between each other. For this reason, the process of parsing and labeling each of the values located in the data stream regularly requires the use of the device's specification documentation. Similarly, in order to have access to each of the properties of collected data by a given property related to time or space, [LNH05] also mentions that Metadata[1] should be included together with the collected data.

An example of a Data Provenance application is the use of the aggregation of data over time to estimate the changing effects of the weather in a given region. For this reason, the collected data from sensor devices need meaningful annotations to better describe the identity of the collected data, as [LNH05] reports. Pertinent questions regarding the data can be asked such as **what was collected?**, **When was the data collected?** or **Where was the data collected?**. Similarly, while Data Provenance provides basic mechanisms to describe sensor networks data in general, others see the importance of the time and position of sensor networks. [YG08] describes strategies used to cluster distributed sensor data as time series of data, as well as to decrease the size of data streams being transmitted in real-time sensor networks when these collected data do not need to be transferred [dAFN+08].

A different way to describe sensor devices is through the use of Annotations. [LCS09] shows the use of tags to describe data collected from a physical area monitoring. The sensor network was designed to monitor moving objects, capturing their location while in movement and their visual appearance using a video camera. In this way, adding tags to particular video frames can describe each of them, and therefore, providing an efficient way to search for a specific frame based on its visual observations. Similarly, the above referenced project also uses data provenance to describe its collected data, since each of the collected data contains the related GPS coordinates of the observed object.

Finally, data provenance provides practical use of the collected data from different sensor

[1]Data about data

devices and different sensor networks in the process called Information Fusison [NLF07]. [LKNZ08] is an example of the aggregation of different data from different devices located in different geographic sensor networks which is capable of processing spatial-temporal queries of real-time or historical data. The following sections describe each of the metadata properties as defined by [LNH05], giving different examples of each of them.

2.4.1 Data as an object

Different sensor devices produce different sets of attributes based on their capabilities and design. In general, these data sets do not have distinct names, which lead to difficulties when indices over the properties of the collected data are required to make them searchable. Specially for the purpose of data archival, the collected data needs to be identified, giving a meaning of independence on its own, the same idea behind the unique identification of entities on the relational data model [Cod70]. Equally, [LNH05] describes that traditional sensor networks use the collected data for analysis long after the attributes have been collected, usually before it is categorized and indexed. Annotations and tags are also names given by [LNH05] as means to describe collected sensor data.

Briefly, the collected data is described by the name of the observation property. For instance, **temperature** = 73.45 expresses the value of the observation of the temperature. In other words, the string "temperature" is the property describing the raw collected data 73.45. On the other hand, the example **temperature-unit** = "Fahrenheit" describes the metadata regarding the temperature.

2.4.2 Time of Data Creation

In order to use the collected data for the purpose of historical analysis, the use of a point in time is the most prominent feature of the data. Also referred to as spatial-temporal data, it is considered a pivotal metadata used in different types of sensor networks such as an aircraft the black-box [Cam07], environmental sensor network [RRGP06], and data communications management systems [Nik05]. In this way, Other projects classify data in time-series [HOE$^+$09], where data is related to all the observations a given sensor device performed over a period of time. The apparent implications is that all the produced data is necessary, but [YG08] argued that some sorts of sensor networks could decrease the creation of “noisy data sets” based on the proper clustering of similar time-series data before they are sent to the data sink.

Based on the guidelines defined by [OS95], the time properties of the collected data can be divided into two different dimensions: **valid time** and the **transaction time**. The former depicts the point in time when the observation took place with respect to the real-world space. An important aspect of this metadata is that since it captures the time when the observation occurred, its value cannot be changed during its existence, as well as placing restrictions on the update operations of this constant. Similarly, the latter type of is the point in time at which a given fact was transferred to the persistence layer. Either types of time can be used depending on the requirements of the sensor network. For instance, [Nik05] reports that all the data stored by its sensor network uses the time of collection, but do not use the transaction time for that purpose. Similarly, [LNH05] shows a practical examples of critical queries performed over the transaction time, when doctors need all the collected data regarding the heart beat rate from the moment the patient entered the ambulance until the moment of arrival at the hospital.

When both temporal dimensions are applied, the collected data is referred to as Bitemporal [OS95]. Usually, the data type of this metadata is a regular timestamps, with any supported precision. To clarify, users may be interested not only when the temperature of 70 degrees happened, but also when it was collected to the data sink. Finally, [LNH05] shows that the collected data for archival purpose is usually saved in self-labeld files described by file names, containing complex naming conventions using the timestamp as a matter of data index.

2.4.3 Data Origin

Depending on the application of the collected sensor data, the location in which the data was collected may be needed. For example, [LKNZ08] shows the geographic location of the collected data is critical to identify the locations of the events of the sensor device [SSK$^+$08]. Similarly, the position of any event captured by a black-box of an aircraft carries metadata for the event. Examples of this are the metadata event = “high-pressure” of 1304, at the altitude of 1,500 feet above the ocean on GPS coordinates of latitude 13.445 and longitude -23,003. This example clearly gives necessary identification of where the event of high-pressure occurred. The precision of the location can be important for the collected data because it helps researchers correlate different measurements based on exact locations of the data. For instance, [LNH05] depicts an example where two different sensor networks covering the same physical area: one covering the reginal traffic and another snowfall. Although each of them has individual functions, collectively they could allow real-time navigation service to answer queries regarding expected traffic patterns based on the current snowfall and advise drivers accordingly by reusing collected data from both sensor networks.

All in all, Data Provenance enriches the value of the collected data from sensor devices by describing its identity creation, time of creation and specific location where the observation occurred. It is important to note, however, that those types of data do not necessarily need to be on the data model used, but the specifications of the use of the data determines which ones are needed. Finally, collected data that are provenance-aware has a direct use on technologies called Information Fusion [NLF07], which aims at using data from different sensor devices.

2.5 Data Load and Management Systems

Depending on the amount of data produced by a sensor network, different approaches are taken to support the data management depending on the amount of data collected. For example, [RRGP06] is an environmental sensor network that produces data regarding marine life. Less than twenty sensor devices produce Megabytes of data on a daily basis, whose data is processed and stored using flat-files directly in the operating system's file system with provenance information regarding the time of collection.

When the amount of data collected grows to other scales, (i.e. Gigabytes) of data on a daily basis, data becomes more difficult to manage in any type of database systems. Examples of sensor networks that produce data under this magnitude are reported in the literature. For instance, [Nik05] describes the creation of a Data Warehouse for the real-time sensor data management using a cluster of MySQL databases used to store data from different optical sensors that collect metrics from physics experiments using eletromagnetic fields. As the author reported, requirements for data management evolved from storing flat files by a monolitic system to a robust distributed system using MySQL database. Moreover, the system targeted from 5 to 10MB of data per second, whose cumulative data reached 1 TB of data every year.

Data management for large scales of data require different techniques to store and retrieve data, since it requires more computational power and data retrieval efficient algorithms. [KB09] describes the creation of a Sensor Data Center designed to store and analyze amounts of data on the range of Terabytes to a future Petabytes of data. Its sensor network deals with not only numerical sensor readings from transportation management, but also radar images, used for historic purposes and data mining that performs traffic pattern analysis. In this way, the authors report the use of parallel data processing techniques in order to provide processing for the collected data, and load balancing when storing data to minimize the sensor data processing delays. For this reason, they suggest the use of concurrent control such as snapshot isolation of the current data into separated replicated data. Therefore, data access from users is not interrupted by the processing of the data that is processed.

In summary, this chapter described a subset of problems and solutions used by the sensor networks community. First, the survey in sensor networks was reviewd to give a foundation of specific properties of sensor networks. Then, different surveys in data persistence in sensor networks were discussed, followed by the data models and query engines. In the end, the most important discovery is that data provenance provides the best guidelines for describing data. The following chapter will classify the properties found in this chapter in terms of taxonomies.

Chapter 3

Proposal of a Data Persistence in Sensor Networks Taxonomy

As reveled in the previous chapter, data persistence for sensor networks require an understanding of not only the properties of the infrastructure of a given sensor network, but also of the properties and nature of the collected data themselves. In order to identify these concepts more clearly, each section of this chapter proposes an individual taxonomy based on the aspects described in the literature review. Then, it briefly discusses the relationships among them.

The root of word "Taxonomy" is the Greek taxis, meaning 'order' or 'arrangement', and, nomos in Latin, meaning 'law' or 'science'). In short, taxonomy is defined as the practice and science of classification and uses the taxonomic units of "taxa", the plural form of "taxon" [tax09]. Similarly, hierarchical diagram is used to relate each of the taxa to their main taxon.

3.1 Purpose of the Sensor Data Taxonomy

As delineated in Section 2.2.1, the purpose of the collected data from sensor devices determines how the collected data is retrieved and used in the network. In such a fashion, this taxonomy relates to how the collected data is used, and should be the primary question answered during the analysis of a sensor network. The taxa related to this taxonomy, depicted in Figure 3.1, is described as follows:

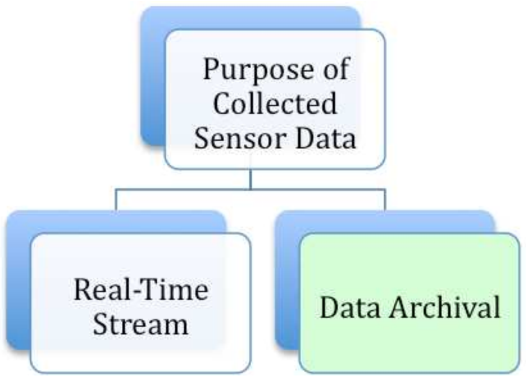


Figure 3.1: The Purpose of Sensor Data Taxonomy

- **Real-time Data Stream**: the collected data is accessed on-the-fly from the sensor device as a real-time data stream, and may have a short life cycle as it temporarily resides in memory;
- **Data Archival**: the collected data is used for historical purposes and is related to the historical data analysis [RM04, ASSC02] or Information Fusion [NLF07].

3.2 Location of the Sensor Data Taxonomy

The location where the collected data is stored plays an important role in the different aspects of the access to the collected data from sensor devices, as shown in Section 2.2.1. Figure 3.2 depicts this taxonomy, which can be described as follows:

- **Local Storage**: characterizes the place the collected data is temporarily located in-memory, or in a secondary storage device on a node located in-network;
- **External Storage**: The collected data is usually located in an external storage device with a data management system such as a relational database. When the collected data occupies different external storage devices, the strategy is called **Data-Centric**

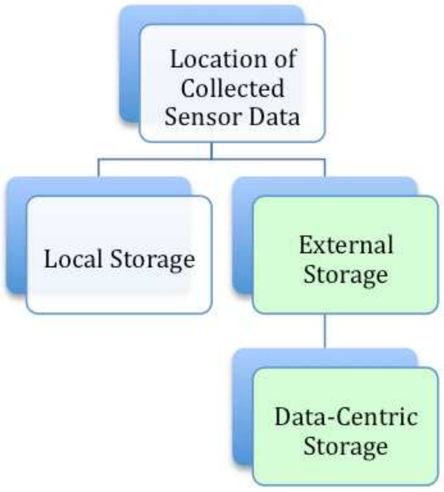


Figure 3.2: The Location of Sensor Data Taxonomy

Storage [SRK+03], being organized by categories based on the collected data keys and values. In other words, what characterizes this taxon is the presence of a data partitioning strategy to store the collected data.

3.3 Data Model Taxonomy

One of the fundamental challenges in the development of a persistence storage in sensor networks is related to the data model chosen to represent the collected data. This taxonomy relates to the data model used to design the collected data, as shown in Figure 3.3, and is described as follows.

- **Schema-Dependent Models**: this data model requires the definition of a master schema that describes the data through a rigorous data modeling process prior to the insertion of the data. Examples of this model is the traditional Relational Data Model[Cod70], as well as the Structured Data Models such as the XML [xml08];
- **Schema-less Models**: contrary to the former taxon, schema-less data models do not require the definition of a data schema or table definition. Instances of such a data model is the use of the Tabular Data model.

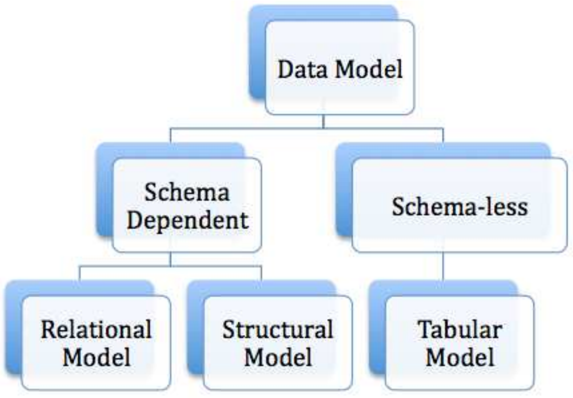


Figure 3.3: The Data Model Taxonomy

3.4 Data Description Taxonomy

Particularly related to how the collected data is described, different projects provide data persistence describing the collected data by using its natural properties, or associating textual descriptions or keywords for the data. The way to describe or label data can follow different guidelines such as data provenance and annotations, discussed in Section 2.4. The description of the properties should take into account the different values of the nature of the data and any other description needed. Therefore, the taxa that compose the Data Description taxonomy, shown in Figure 3.4, are described as follows:

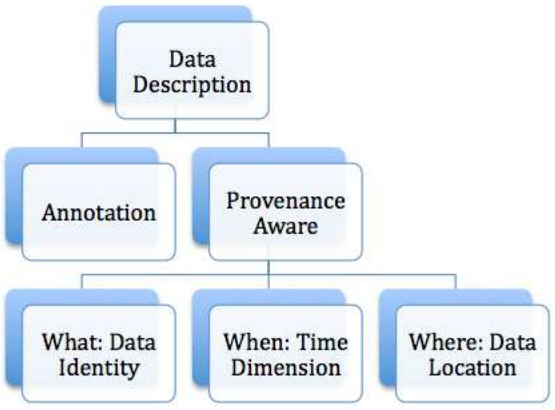


Figure 3.4: The Data Description Taxonomy

- **Provenance-Aware**: when the collected data is categorized based on the nature of the data, **What: Data Identity** is the data type that uniquely identifies the collected data from sensors. Similarly, the **When: Time Dimension**, is related to instances of time relative to the collected data: **valid** and **transaction** times. The former is used to identify when the data was collected, whereas the latter when the data was transmitted to the data sink where the query processing acts. Finally, the **Where: Data Location** identifies the point where the collected data was observed by the sensor device. Usually the latitude and longitude as the GPS coordinates;

- **Annotations**: the annotation of the collected data, through the use of keywords, full-text description, or any other method that helps describe the collected data. It is important to note that this type of data does not makeup part of the collected data from sensors. Rather, they complement the observations of what was perceived by the sensor devices.

3.5 Query Processing Mechanism Taxonomy

The query processing mechanism in sensor networks is directly related to how and where the collected data is used and stored, as shown in Section 2.2.2. The following taxa are related to the different types of query processing mechanisms and summarized in Figure 3.5:

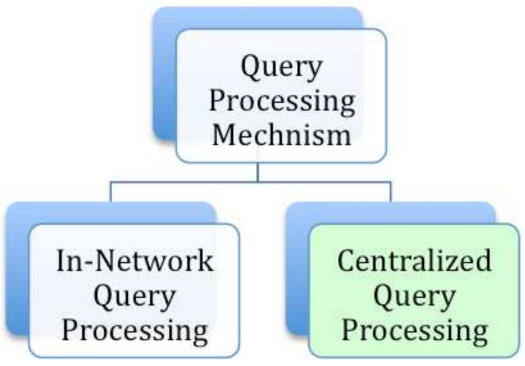

Query Processing Mechnism
In-Network Query Processing
Centralized Query Processing

Figure 3.5: The Query Processing Mechanism Taxonomy

- **In-Network Query Processing**: when the data node is the sensor device itself, and it uses the local storage mechanism, then the sensor node is able to reply to the request related to the observations of the environment defined by the sensor device. In this way, the query takes place in the network itself, and usually returns values related to the current state of the environment;

- **Centralized Query Processing**: as developed by a single relational database server, the data queried from a centralized data network sink in order to be reused. In general, this approach is used when the purpose of data is archival data.

3.6 Data Volume Taxonomy

When it comes to the data volume produced during a time frame, different ranges are reported from different types of sensor networks. Depending on the types of data produced such as numerical or images, they can be in the orders of Megabytes (1MB = 1000KB = 10^3 bytes), Gigabytes (1GB = $1000^3 = 10^9$ bytes) and even Terabytes (1TB = $1000^4 = 10^{12}$) to future Petabytes (1PB = $1000^5 = 10^{15}$ bytes) as shown in Section 2.5. As a comparison of what is processed, Google reports to process 20 Petabytes of data per day in their data centers [DG08]. In this way, the following taxa are related to the different sizes of data produced by a sensor network, summarized in Figure 3.6:

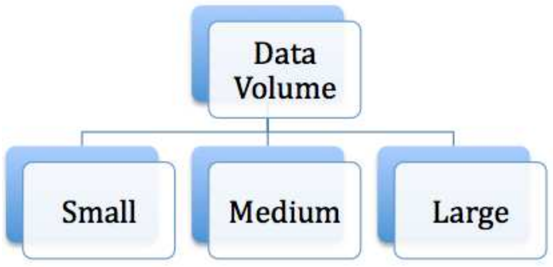


Figure 3.6: The Data Volume Taxonomy

- **Small**: sensor networks whose annual production of data is in the range of MBs or to 1GB of data, usually stored in flat files directly in the operating system. They are usually collected from sensor devices that only generate numerical data;

- **Medium**: amount of data that ranges from GBs to 1TB data and is managed using disk arrays;
- **Large**: the amount of data produced by a sensor device ranging from TBs to 1PT of data necessitates the construction of a data center specializing in the storage of data produced by sensor devices, taking into account the addition of data provenance, among others.

3.7 System Organization Taxonomy

Different approaches have been developed to organize the storage of data produced from sensor networks. Figure 3.7 describes the taxonomy of the database system organization as follows:

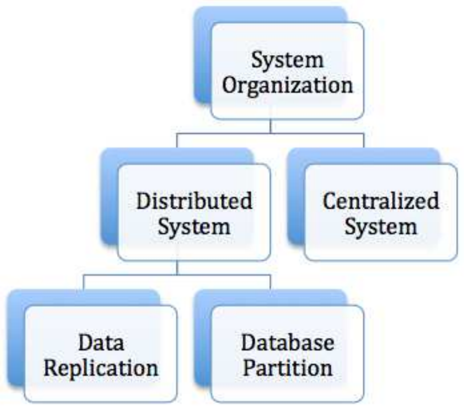


Figure 3.7: The System Organization Taxonomy

- **Centralized System**: a database system that runs in one single centralized host, the focus being the managing of the collected data [RM04]. When the system is organized in a centralized system, usually a centralized query processing is used to process any request to operations on the database management system;
- **Distributed System**: a database system that can be composed of more than one node. Usually, the use of the pattern of Master-Slave mechanism or data replication

[KB09] is used to alleviate “hot spots” during concurrent processes of data collection from the sensor devices and data access from the final users. Unique mechanisms can be used to disaggregate the produced data from a concentrated one: the use of database partitioning, as related to the data-centric storage, where each partition is responsible to handle data produced by a given type of sensor, for instance. Finally, it is the type of system used to support in-network query processing.

3.8 Relationships among the Taxonomies

On the whole, the taxonomies described in this chapter were based solely on the literature review described in the previous chapter. They can be helpful in understanding the requirements of a data persistence layer for sensor networks with different infrastructures. They are summarized on Table 3.1, and briefly highlight these relationships in order to better analyze a problem related to data persistence.

The taxonomy related to the purpose of the sensor data can be directly associated with the taxonomy of the location of the sensor data. While the taxon of the real-time data stream relates to the local data storage, the taxon related to data archival correlates the storage of the collected data with an external storage one. In addition to that, the converse is also true.

The taxonomy related to the data model can be linked to any taxon from the system organization because it is used in both centralized and distributed systems, also associated with

Taxonomy Name	**Associated Taxas**
Purpose of the Sensor Data	Real-time Data Stream, Data Archival
Location of the Sensor Data	Local Storage, External Storage
Data Model	Schema-Dependent, Schema-less Models
Data Description	Provenance-Aware, Annotations
Query Processing	In-Network Query, Centralized Query
Data Volume	Small, Medium, Large
System Organization	Centralized System, Distributed System

Table 3.1: Summary of the Data Persistence Taxonomies

the location of the data and, consequently, the query processing method used. Furthermore, the data description taxon can be used by any of the data models.

The query processing is directly related to the location of the collected data and with the system organization taxa. While centralized systems use centralized query processing methods, in-network query processing is when the collected data is located in different external storage devices organized in a data-centric way, usually supported by a distributed system.

Similarly, the taxonomy related to the volume of data produced by a sensor network usually determines the system organization used to handle and process the collected data. When the volume is small, the use of a single server is required, usually using an external storage device, which relates to the taxonomy for storage systems. However, when the volume is medium to large, the use of distributed systems is required to cope with high volume of data processing. Techniques described for data replication is used to decrease the load in hot spots of the architecture. On the other hand, approaches to separate the data being written and read can be used such as Load Balancing or Database Partition, and the use of a Data-Centric storage approach is used.

In the light of these taxonomies, the classification of the data persistence properties of a sensor networks can be studied and reviewed. As described in the introduction of this report, one the main motivations for this work was to provide means of data persistence for NetBEAMS, a case study which does not contain a data layer. Therefore, the following chapter seeks to describe NetBEAMS and its supporting infrastructure.

Chapter 4

Data Persistence for Sensor Networks: A Case Study for NetBEAMS

As discussed in Chapter 2, different properties of a sensor network and the nature of the collected data must be taken into account in order to provide a data persistence layer for a given sensor network. Similarly, in order to better assist one's analysis of such functionality, Chapter 3 proposed a set of data persistence taxonomies related to different properties of the collected data life cycle. The goal of this chapter is to describe the fundamental properties of the case study used for this work, that is, NetBEAMS, in order to lay the groundwork as it relates to the selection of the database technology. NetBEAMS provides an automated infrastructure solution for SF-BEAMS, and this last component will be covered first. Then, this work proposes a categorization of NetBEAMS according to the defined taxonomies.

4.1 SF-BEAMS: a Marine Sensor Network for Water Quality Monitoring

As described by [PJS$^+$09], NetBEAMS is a joint venture between the department of Biology and Computer Science at San Francisco State University, whose goal is to automate the operational execution of SF-BEAMS. The San Francisco Bay Environmental Assessment and Monitoring Station, or SFBEAMS [RRGP06], is an environmental sensor network system whose primary focus is the study of complex marine and estuarine environments using the SF-BEAMS sensors. The sensors are deployed off a pier located in the San Francisco Bay, in Tiburon, California; and is operated by the Romberg Tiburon Center for Environmental Studies (RTC) a part of San Francisco State University. NetBEAMS is an experimental component that utilizes the SF-BEAMS infrastructure to operate. This section details SF-BEAMS in general, and presents the requirements for data persistence after classifying NetBEAMS based on the taxonomies defined in the previous chapter.

4.1.1 The SF-BEAMS Infrastructure

The SF-BEAMS sensor network is responsible for providing data for water quality monitoring, as well as weather and surface conditions. Figure 4.1 shows a picture taken from the SF-BEAMS web-camera.

In general, the SF-BEAMS network infrastructure contains a varying number of wired and wireless devices attached to pylons at the pier. Each of them is responsible for observing different conditions of the area, having its own mechanisms for internal storage for the collected data. In this way, data can be directly transferred to the labs via Ethernet cables or collected manually with a laptop computer. The current data collection process for SF-BEAMS is described in Figure 4.2 and documented at [sfb08]. Upon collecting data

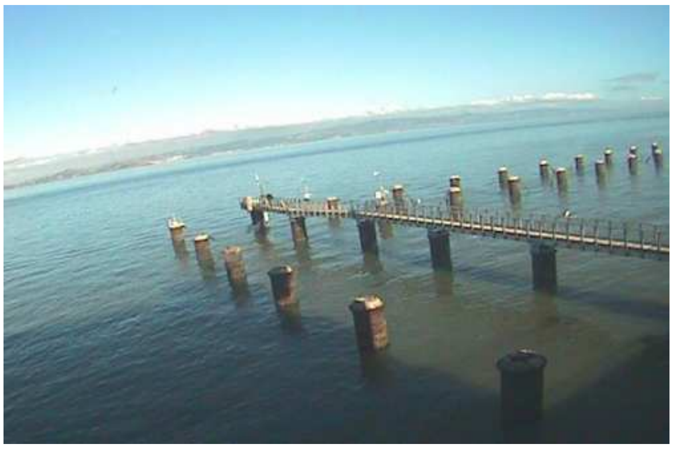


Figure 4.1: Picture of the SF-BEAMS Network, at the RTC pier. Tiburon, CA.

from sensors, the RTC staff uses automation scripts written in Matlab [Duk09] to process, index and distribute the raw data in different formats. One of these formats is the OPeNDAP [ope08], which is widely used at research institutions to promote easier data exchange amongst them. The SF-BEAMS website provides access to the collected data through the Internet at http://sfbeams.sfsu.edu:8080/opendap. An example of an HTTP Request from accessing the data for a specific device is shown in Listing A.37 on page 177.

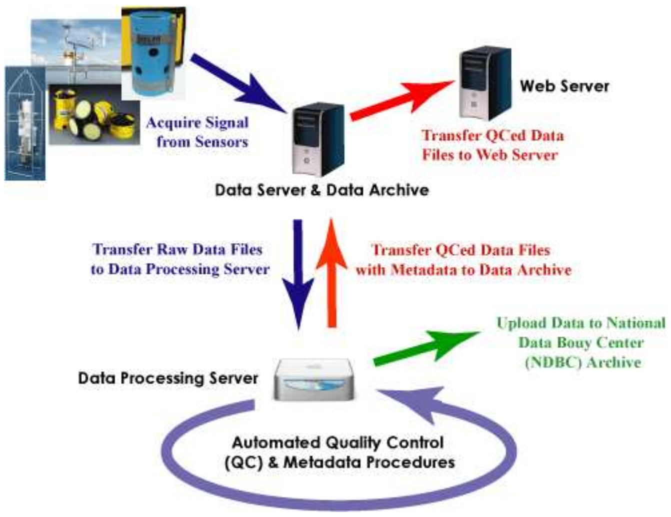

Web Server
Acquire Signal from Sensors
Transfer QCed Data Files to Web Server
Data Server & Data Archive
Transfer Raw Data Files to Data Processing Server
Transfer QCed Data Files with Metadata to Data Archive
Upload Data to National Data Bouy Center (NDBC) Archive
Data Processing Server
Automated Quality Control (QC) & Metadata Procedures

Figure 4.2: The Current SF-BEAMS Data Collection Process

As described in Section 2.1.2, sensor devices produce the observed data based on properties of measurements defined by its manufacturer. SF-BEAMS has used, among other devices, the “6600 ESD V2” Sonde [YSI08], manufactored by YSI Incorporated, as seen in Figure 4.3. Although it is a small device of under 55 cm in length, 8.9 cm in diameter, and weighting approximately 3.18 kg, it is a powerful water quality-monitoring device, capable of producing around 52 bytes on a single real-time data stream reading regarding 12 different parameters such as water temperature, salinity, turbiny, among others. An example of the data stream is as shown in Listing 4.1 on page 29.

Listing 4.1: The Collected Data Stream from the YSI Sonde 6600ESDV2

```
12.20     192     179  5588.40     0.09     0.084     0.059    7.98     -79.6     99.5    8.83    0.4      8.7
```

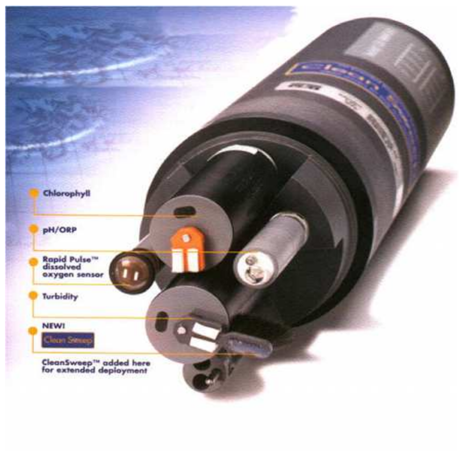

Chlorophyll
pH/ORP
Rapid Pulse™ dissolved oxygen sensor
Turbidity
NEW!
Clean Sweep
CleanSweep™ added here for extended deployment

Figure 4.3: Picture of the case study Sonde device: YSI 6600 ESD V2

In order to have access to the observed data, the RTC network staff downloads the data using one of the device's connections such as the RS-232 serial connector [dAPZJF+01]. Then, they upload the data into their data servers where they are processed and archived for historical analysis. According to one staff member, the infrastructure of the SF-BEAMS is comprised of the following components (as of May 2009):

- 5 YSI Sonde devices, among others, in operation at the RTC pier;
- The sampling frequency rate is configured in ranges of either 1, 6 or 15 minutes, depending on the specification;
- The processed data, used for distribution over the Internet, contains information regarding the time of the data collection.

Considering the infrastructure, the data load in the server-side of RTC's laboratories produced by the YSI Sonde sensors devices can be estimated by analyzing the data distribution over a period of one year, as shown in Table 4.1.

The maximum amount of data produced by the YSI Sonde device is hundreds of Megabytes a year, as **483,840** samples are generated by one single device.

# of YSIs	Rate	Hourly	Daily	Weekly	Monthly	Yearly
1	1 min	3.04 Kb	73.12 Kb	511.87 Kb	1.99 Mb	23.99 Mb
5	6 min	15.23 Kb	365.62 Kb	2.5 Mb	9.99 Mb	119.97 Mb
1	15 min	0.5 Kb	12.18 Kb	85.31 Kb	341.25 Kb	3.99 Mb
5	1 min	2.54 Kb	60.93 Kb	426.56 Kb	1.67 Mb	19.99 Mb
1	6 min	0.2 Kb	4.87 Kb	34.12 Kb	136.5 Kb	1.6 Mb
5	15 min	1.0 Kb	24.37 Kb	170.62 Kb	682.5 Kb	7.99 Mb

Table 4.1: Estimated data volume produced by the RTC's YSI sondes

4.1.2 Taxonomic Classification of SF-BEAMS

This section classifies the sensor network deployed by SF-BEAMS according to the taxonomies proposed in the previous chapter. This classification is necessary to better understand the requirements of a data persistence for NetBEAMS, which indirectly uses the infrastructure of SF-BEAMS.

- **The Purpose of Sensor Data**: the data from SF-BEAMS is exclusively used for the purpose of Data Archival, since they are stored in RTC's lab;

- **The Location of the Sensor Data**: it is clear that the purpose of the data is directly related to the sensors location and, thus, the strategy of **External Data Storage** is used to store its collected data at the network sink at the RTC site;

- **Data Model**: since the format used by the RTC staff to share data with researchers is in the OPeNDAP format, the data model used is the tabular data model, since it uses the comma-delimited files. In this way, the **Schema-less** model is used;

- **Data Description**: once the collected data reaches the RTC lab, a quality assurance process is executed, and the data is transformed into the format used by OPeNDAP. The process to describe the collected data includes the time of the data collection, therefore, one of the **Time Dimensions** such as the valid time is used. Since the files are stored with specific file name structure, using timestamps to describe the data, the use of transaction time may be also considered. Finally, the contents of the files contain the **Data Identity** as shown in the samples of the collected data in Listing A.37 on page 177;

- **Query Processing Mechanism**: similar to the data stored in a centralized way, the query processing is also **Centralized**, as the data is stored on file system;
- **Data Volume**: the volume of data produced is characterized by small. It is related to the volume of data produced by the sensor devices, which is numerical data streams, and the total number of sensor devices;
- **System Organization**: SF-BEAMS runs on a Single System, as the collected data is stored directly in the file-system after previous processing. However, the collected data is provided to the Internet using the OPeNDAP Hyrax system as a middleware.

4.2 NetBEAMS: a component-based approach for SF-BEAMS

Although the execution of RTC's sensor network can be operated as it was mentioned in the previous sections, [PJS+09] described some of the operational challenges faced by the RTC staff during regular activities of the data collection process. It was clear to the research group that SF-BEAMS could have not only its data gathering process automated, but also its data management and distribution. By the use of COTS[1] embedded devices and open-source [vKS07] software, the research group developed NetBEAMS, component-based approach which suggested the improvement of operational activities from SF-BEAMS using systems automation. In brief, one of the ongoing problems of NetBEAMS was regarding Data Persistence, the primary motivation of this report.

The Networked Bay Environmental Assessment and Monitoring System, or Net-BEAMS, offers the Data Sensor Platform (DSP) [PJS+09] as the system architecture that can address the operation of SF-BEAMS. The in-depth documentation describing the DSP Platform, including its architecture and data gathering process is provided at the online documentation [dS09].

[1]Common-Off-The-Shelf designates a product that is produced and sold in bulk

The scope of this work is defined as to provide a persistence layer for NetBEAMS, making a selection of a persistence layer based on the taxonomical classification of SF-BEAMS. Furthermore, the technology to be selected for the persistence layer must take into account the requirements of NetBEAMS primary users such as Computer Science researchers, as well as end users such as Biologists from the RTC laboratories. The requirements for the data persistence specifically for NetBEAMS are described in the following section.

4.3 Requirements for NetBEAMS Data Use

The main reason for the operation of SF-BEAMS using NetBEAMS's automated approach is that it can potentially reduce operational costs and improve the data collection process. In view of this, the scope of a persistence layer can be summarized as a group of functional and non-functional requirements in order to select a database system for the sensor network supported by NetBEAMS.

4.3.1 Functional Requirements

- Reuse the NetBEAMS infrastructure and develop a component responsible for data persistence in a database system;
- The Persistence System must obey to the characteristics of the categorization of SF-BEAMS used by the taxonomies.

4.3.2 Non-Functional Requirements

- The data model used to describe the data must not impose restrictions to the users of the system (i.e. Biologists and students without expertise in Database Systems);
- Data Representation must be similar to those used by RTC, with a more human-readable format;
- The system must be scalable with a low degree of maintenance, in a way that the interruptions to the data gathering process are infrequent;

- Data must be searchable in near-real-time with good performance;
- The system must also support export capabilities, which may be used to match RCT's requirements of the OPeNDAP format;
- The database must be free of charge, following the implementation specifications of NetBEAMS for Open-source software.

In conclusion, after detailing the SF-BEAMS sensor network, and describing the infrastructure of NetBEAMS, its the motivation and goals for this project were presented. The next chapter is the actual analysis of existing database systems mentioned in Chapter 2, which aligns with the requirements and characteristics of NetBEAMS. Meanwhile, more details regarding the architecture of NetBEAMS, as well as the development guidelines can be found in the appendix of Chapter 10.

Chapter 5

Empirical Analysis of Data Persistence Taxonomies for NetBEAMS

Providing data persistence for an existing sensor network requires analysis of the infrastructure and the nature of the data produced. Based on the taxonomies of Chapter 3, and the requirements of a data persistence for NetBEAMS in Chapter 4, this chapter aims at analyzing a group of database technologies selected from the topics covered in Chapter 2. Similarly, supported by the scope of the required implementation presented in this chapter, the comparison of the technologies is summarized, highlighting the selected database system used for the design of a component for NetBEAMS and used for the design of the experiments.

5.1 Scope of the use of the Technology

The scope of the use of the database can involve different functional and non-functional requirements defined in the last chapter. This work does not intend to implement an application based on the database system selected, but to provide the foundation of a database management system that can be used to develop one focused less in refactoring and high scalability and performance. For this reason, the use of the database system will be restricted to the data persistence and use through a native system or through the use of native programming languages or API. In this way, the scope of this technology selection can be summarized as follows:

- Provide data management over the collected sensor data, as a conventional database systems;
- Must be able to scale in any increases to the data load of the sensor network, as described in Chapter 2;
- Must be able to use single or multiple servers to improve performance and optionally deal with Data-Centric approaches to sensor data partitioning;
- Must provide a data model that scales, decreasing the constant data schema changes.

5.2 Database System "Contenders"

This section describes the database systems "contenders" considered to be used as the persistence technology backend for NetBEAMS. As described in the previous chapter, the selection of the technologies must be based on the list of functional and non-functional requirements specified within the scope of this work.

One of the essential questions related to data persistence is regarding the data model and types of the database. That is, the use of the relational or any other model to describe data along with its management. Traditionally, the relational data model has been used to provide data persistence for different types of projects, including sensor networks [Nik05, CZR03, MFHH05]. However, tradition does not always get translated into efficiency in terms of problem solving. For this reason, [Bai09] reflects about the use of alternatives to the relational model when the data persistence problem in consideration is to provide data persistence for dynamic environments such as Internet applications. The author also suggests that the relational model is hard to maintain in ever-changing environments, and thus, proposes the use of the Key-Value-Pair (KVP) data model and a different number of technologies. Similarly, [BYV+09] reports similar trends in academic and industrial research in distributed computing using KVP databases in powerful systems such as Cloud Computing [LKN+09]. For this reason, in addition to the different database systems used by projects in the literature review, this work also includes the review of a KVP database system called mongoDB [mon09], since it was among those cited in the surveys. In such a manner, the list of the systems considered for assessment is the following:

- **MySQL**[mys99]: an open-source relational database system popularized by the development of Internet applications, maintained by Sun Microsystems. [Nik05] reports its use to develop a data warehouse for data collected from sensor devices using this technology;
- **TinyDB**[tin09]: developed and maintained by the Computer Science department of University of California, Berkeley, TinyDB is a relational database system specially developed for sensor devices. As referred in Section 2.2, it is system used in many different sensor networks implementation such as [MFHH05, LKK07];
- **MongoDB**[mon09]: an open-source KVP database system suggested by different surveys to be an alternative to relational databases [Bai09, LKN$^+$09]. It is maintained by 10Gen Incorporated;
- **DB2**[db209]: in the pursuit of an XML database system, DB2 is listed as a hybrid database system that offers both the relational and the XML models [CZR03]. It is privately owned by IBM.

The following sections review these technologies' capabilities to adhere to the specifications of the taxonomies defined in Chapter 3, as well as the requirements and the scope for a data persistence of NetBEAMS, as defined in Chapter 4.

5.3 Analysis of the Purpose of Sensor Data

The execution of the SF-BEAMS sensor network, together with the NetBEAMS automated infrastructure, can be summarized as follows:

- Data is generated by sensor devices, such as the YSI Sonde [YSI08], and manually collected by using a laptop to the data sink at the RTC laboratories;

- Upon data reception, the RTC staff index and archive the data for distribution using the OPeNDAP format;
- An automated approach to the data collection procedures is performed using NetBEAMS [PJS+09], using software components to obtain and send the data to the data sink node. A complete description of this approach is [dS09].

The nature of data from NetBEAMS through SF-BEAMS is used for the purpose of **Data Archival**. After the collected data captured from the sensors, it requires further processing in order to be archived for later reuse. Different types of users are expected to use the collected data, ranging from researchers, students, and the public over the Internet.

5.3.1 Technology Analysis

Since all of developed database systems were implemented with the intention of persisting data in long-term storage devices, every one listed provides support for data persistence and management. Therefore, they are compliant to the purpose of Data Archival for the sensor networks.

5.4 Analysis of the Location of Sensor Data

The advantages of the External Storage approach is the most common way to implement persistence for any type of systems since data management is usually offered by most database systems out-of-the-box. It is basically used to store small volumes of data. On the other hand, the most common problems related to a single External Data Storage approach are related to the capacity of the device and operational problems. If the external storage device is situated in the network, then the capacity of the device may be limited to receive collected data from the sensors in the network, since it may run out of space. Similarly, storage devices are very error prone and its hardware part may fail.

As shown in the previous chapter, NetBEAMS's architecture is based on sensor network whose topology is a single-hop star with a centralized data sink. In this way, all the collected data is transmitted to that single location as described in Section 2.1.2. The simplest location to save the collected data is through the use of at least an **External Storage**. Similarly, the use of a Data-Centric Storage strategy is considered in situations where the data sink is about to reach its storage capacity. The **Data-Centric Storage** separates the collected data into dissimilar locations, based on any property of the data. Therefore, there are two different approaches to data partitioning:

- **Horizontal Partitioning**: the data segmentation is implemented based on the values of the table rows, in the relational model point of view, or the collections elements, on KVP databases. That is, all the values defined by the columns are used in a given table physically located in specific servers. For instance, horizontal partitioning can be defined as a collection of five years worth of historical sensor data, partitioned into five separate servers. In this way, each server contains data related to each of the years;
- **Vertical Partitioning**: data is partitioned vertically in a way that the columns of an entity on the relational model are divided into different database tables. In this way, the particular dataset is broken down to different tables. For example, if the database is used to store pictures of a given sensor camera in a BLOB column, it would be the most preferable to the scenario where the images are not constantly referenced when compared to the textual data. Since KVP uses denormalized data, this partitioning strategy is not used.

5.4.1 Advantages of the Data-Centric Approach

The benefit of the Data-Centric approach can be directly associated with database partitioning mechanisms described earlier. Since the collected data is distributed into different locations, this approach decreases the size of datasets when the database server performs a query processing in a database table. For this reason, this approach helps the database administrators scale their database infrastructure and improve the database performance.

The horizontal or vertical partitioning strategies are also referred to as Database Sharding [sha09], or simply Database partitioning [Sch06]. Both the former and the latter are used in regular schema-dependent models or schema-less models. Different approaches worth mentioning:

- **Hash Partitioning**: a hash key is computed based on different table columns of an entity [Oba09, Sch06], providing an even distribution of the data;
- **Key Partitioning**: is based on the hash partitioning over the values of a specific key [Mer09b];
- **Range Partitioning**: the partition is based on specific ranges of the values of given a given column of a table. For instance, the collected data from the sensors could be separated by the year of its collection. For example, data collected in the year of 2009 would be stored in a different partition than the data collected in the year 2010 [Sch06].

An example of a data-centric storage with different partitions can be seen in Figure 5.1, where the collected data is partitioned by its origin. Each shard is assumed to be placed in different server hosts, as well as each of them uses the same "denormalized" data models.

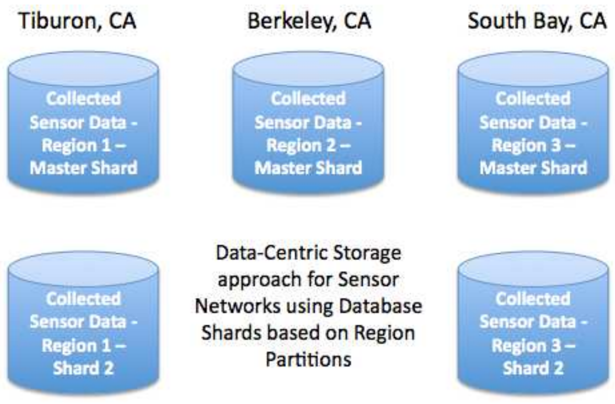

Tiburon, CA
Berkeley, CA
South Bay, CA
Collected Sensor Data - Region 1 – Master Shard
Collected Sensor Data - Region 2 – Master Shard
Collected Sensor Data - Region 3 – Master Shard
Collected Sensor Data - Region 1 – Shard 2
Data-Centric Storage approach for Sensor Networks using Database Shards based on Region Partitions
Collected Sensor Data - Region 3 – Shard 2

Figure 5.1: Example of Data-Centric Storage for Sensor Networks using Database Sharding

5.4.2 Disadvantages of the Data-Centric Approach

The data partitioning is a highly restricted technology, normally available to advanced users of distributed database systems. Despite being offered as a common-off-the-shelf functionality by some popular database systems, this strategy requires closer database management and configuration. Some of the problems related to this strategy are regarding its lifecycle management [sha09]:

- **Rebalancing Partitions**: considering a schema-dependent database system, changes on the schema of one shard requires rebalancing the collected data on each of the shards. As a consequence, the database shards transfers data to new locations whenever necessary. This technique is starting to appear as COTS implementations such as MySQL [mys99] and mongoDB [mon09];

- **Referential integrity**: also considering a schema-dependent database system, any cross-shard query may run into the problem where references do not exist in other shards, since the application layer is responsible for enforcing data integrity. In this way, examples of such include verifying constraints of foreign keys when the partition is done using Vertical table partitioning, considering that data collections are organized in separate spaces, and physically placed in different shard nodes [Sch06].

5.4.3 Technology Analysis

- **MySQL**: Supports External or Data-Centric Storage approaches. When Data-Centric Storage is used, both vertical and horizontal partitions [Sch06] can be chosen, using different strategies to partition the data;

- **TinyDB**: Only supports the External Storage;

- **MongoDB**: Supports External or Data-Centric Storage. When the latter approach is used, the horizontal partition strategy is available with automatic shards [Mer09b];

- **DB2**: Supports External and Data-Centric storage. Only supports vertical table partition for the data-centric storage [CFL$^+$07].

All the selected database systems support the External Storage method. However, they all differ somewhat in terms of Data-Centric Storage support. Therefore, MySQL and mongoDB are promising choices, with the difference of the use of the data model.

5.5 Analysis of the Data Model

One of the most common practices in the area of database system is to use the relational model for data persistence, although the application of the system may not fit to solve the problem. In fact, the main users of sensor networks may not possess any expertise in database systems or data modeling, based on the fact that they are from different science areas. [BDD09] suggested the use of programming languages abstraction, instead of regular database systems, to provide access to the collected data for both technical and non-technical users. For instance, marine biologists manage SF-BEAMS without expertise in data management or modeling systems. For this reason, one of the requirements of the system is the use of a schema-less approach, which describes the collected data without data modeling process used in schema-dependent models. The **Key-Value-Pair Data Model** seems to be an optional candidate that provides such a method of data design.

5.5.1 Analysis of the Schema-Dependent Models

Considering the inception of a relational data model [Cod70] and the use of the YSI Sonde data as the main entity in the system, Figure 5.2 represents a prototype of the relational model, after passing through the process of normalization [Ken83].

When a new type is added into the system, the refactored version of the relational model be depicted in Figure 5.3. Some observations are considered in this scenario:

- Data schema that constantly change requires continuous database normalization processes, changes to structure, database management, and all the code that uses the data. Schema-Dependent databases always require refactoring of the schema to accommodate changes to the entities' properties;

- [LNH05] describes the relational model as problematic when describing collected data from sensor devices;

- [LKK07] suggests changes to the SQL clauses of TinyDB to better support time-series, an important property when describing collected data from with spatial-temporal properties.

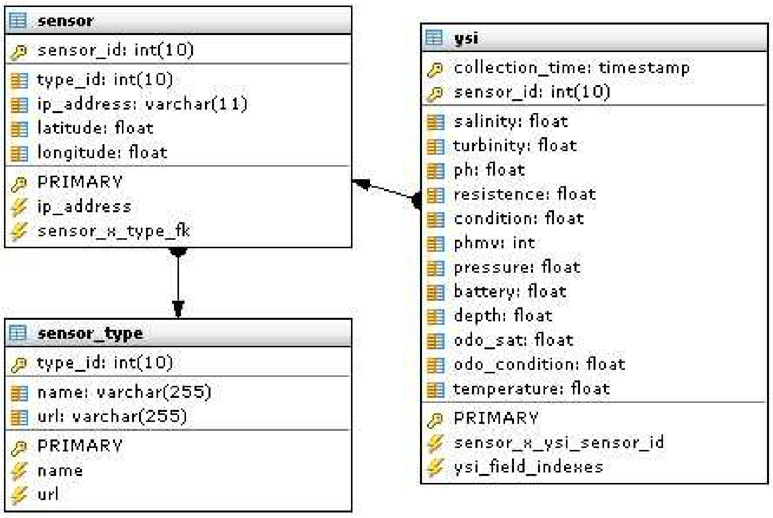


Figure 5.2: Relational Data Model for NetBEAMS - A first prototype

Other projects have used the XML data models for persistence. It falls into the same category as the Relational Model using XML documents [xml08], since XML documents must fulfill the specifications defined by an XML Schema [xml04]. This model uses XPath

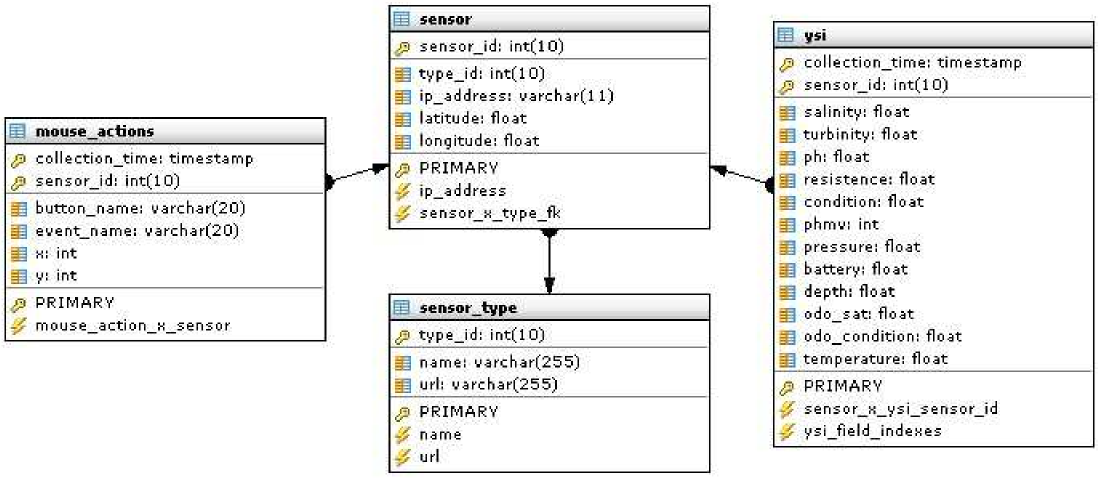
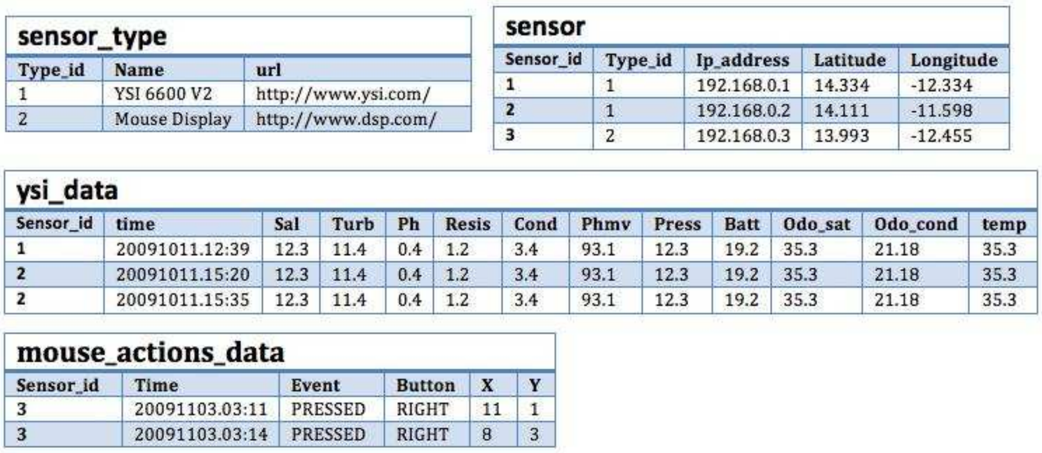


Figure 5.3: Relational Data Model for NetBEAMS - Modified version with new entity

[xml07] technology for querying documents, although some hybrid [db209] technologies may still use SQL [Mel96] for that matter. In this way, the view of an example of database state for the defined previous schema can be seen in Figure 5.4.



sensor_type

Type_id	Name	url
1	YSI 6600 V2	http://www.ysi.com/
2	Mouse Display	http://www.dsp.com/

sensor

Sensor_id	Type_id	Ip_address	Latitude	Longitude
1	1	192.168.0.1	14.334	-12.334
2	1	192.168.0.2	14.111	-11.598
3	2	192.168.0.3	13.993	-12.455

ysi_data

Sensor_id	time	Sal	Turb	Ph	Resis	Cond	Phmv	Press	Batt	Odo_sat	Odo_cond	temp
1	20091011.12:39	12.3	11.4	0.4	1.2	3.4	93.1	12.3	19.2	35.3	21.18	35.3
2	20091011.15:20	12.3	11.4	0.4	1.2	3.4	93.1	12.3	19.2	35.3	21.18	35.3
2	20091011.15:35	12.3	11.4	0.4	1.2	3.4	93.1	12.3	19.2	35.3	21.18	35.3

mouse_actions_data

Sensor_id	Time	Event	Button	X	Y
3	20091103.03:11	PRESSED	RIGHT	11	1
3	20091103.03:14	PRESSED	RIGHT	8	3

Figure 5.4: Instance of a Relational Model Database Prototype

This data model has been used to solve data models for different types of application. One such a problem is the one to represent entities by a list of properties with keys and relating values, called Key-Value data model [Oli06, Gos06]. The author of [Oli06] suggests that this is used as a technique used when the addition of number of properties of an entity constantly changes, and thus, requires constant refactoring of the relational data model. Considering this data model for sensor network devices, whose properties are lists of keys and relating values, an “one-fits-all” attempt to use the KVP data model using the relational data model is depicted in Figure 5.5, and a given example of the state of a database using this schema in Figure 5.6.

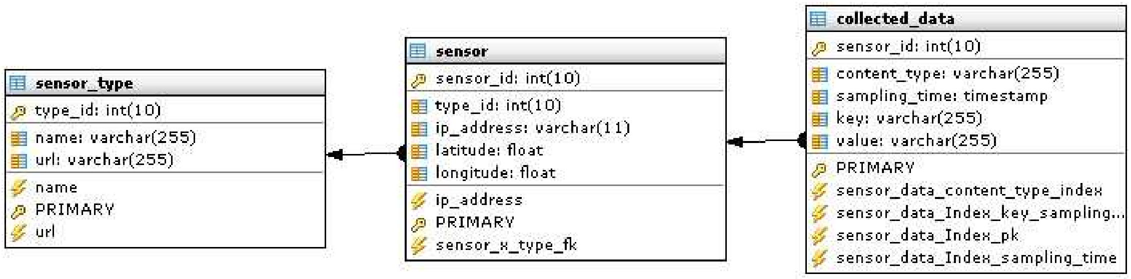
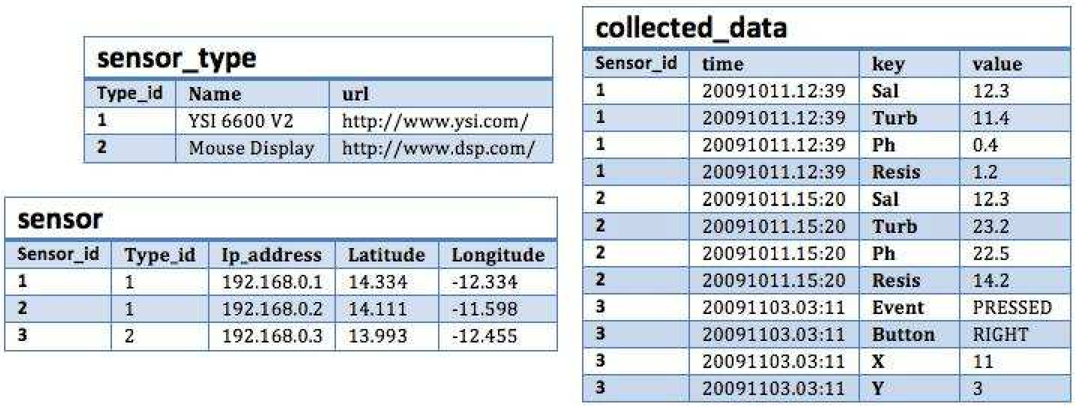


Figure 5.5: KVP Data Model implementation using Relational Model



sensor_type

Type_id	Name	url
1	YSI 6600 V2	http://www.ysi.com/
2	Mouse Display	http://www.dsp.com/

sensor

Sensor_id	Type_id	Ip_address	Latitude	Longitude
1	1	192.168.0.1	14.334	-12.334
2	1	192.168.0.2	14.111	-11.598
3	2	192.168.0.3	13.993	-12.455

collected_data

Sensor_id	time	key	value
1	20091011.12:39	Sal	12.3
1	20091011.12:39	Turb	11.4
1	20091011.12:39	Ph	0.4
1	20091011.12:39	Resis	1.2
2	20091011.15:20	Sal	12.3
2	20091011.15:20	Turb	23.2
2	20091011.15:20	Ph	22.5
2	20091011.15:20	Resis	14.2
3	20091103.03:11	Event	PRESSED
3	20091103.03:11	Button	RIGHT
3	20091103.03:11	X	11
3	20091103.03:11	Y	3

Figure 5.6: KVP over Relational Model Database Prototype

5.5.2 Analysis of the Schema-less Models

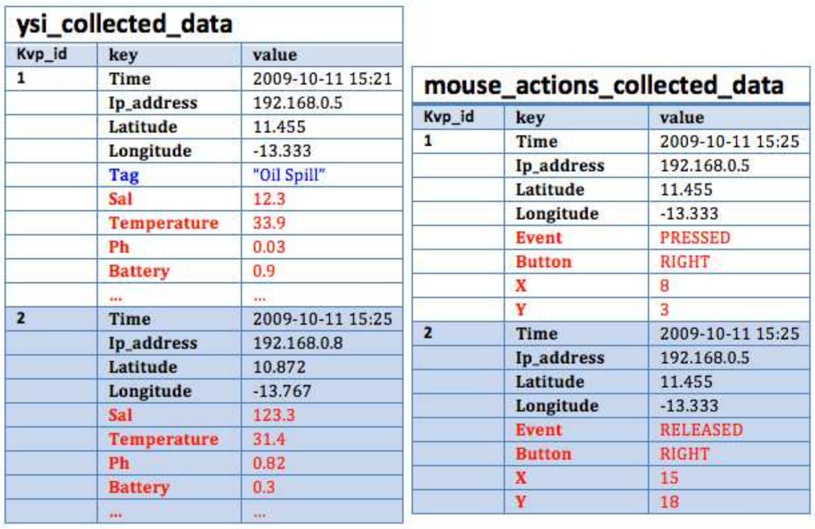
The growth of distributed systems and the Internet have enabled the development of powerful database servers that can be organized in the context of infrastructure and how to represent models. A powerful up-coming approach is the use of database systems that do not use structured language for querying its data. One such example is called Key-Value Pair (KVP) data model [DHJ+07], and it is characterized as follows:

- data is structured in collections of key-value pairs, as it is done in hash data structure. The definition of the key is a given property with its associating value;
- the query process is using mechanics similar to programming or scripting language, that is, the use of APIs in a given programming language.

There are no records of the use of this data model with sensor networks. Data is generally aggregated by given entities and any needed property applied to it. Key may be composed by embedded keys having a dynamic structure and freely used without a schema validation mechanism as used schema-dependent data models, having the following organization:

- entities from the same domains are placed into a “bucket”;
- entities have a set of attributes and relating values;
- entities with different set of attributes may be contained in the same bucket, since there is no schema to govern the bucket items restriction.

For instance, all data needed for the YSI Sonde Data, as well as all necessary provenance data and annotation as defined in Section 2.4, are represented by means of key-value pairs as seen in Figure 5.7.



ysi_collected_data

Kvp_id	key	value
1	Time	2009-10-11 15:21
	Ip_address	192.168.0.5
	Latitude	11.455
	Longitude	-13.333
	Tag	"Oil Spill"
	Sal	12.3
	Temperature	33.9
	Ph	0.03
	Battery	0.9
	...	...
2	Time	2009-10-11 15:25
	Ip_address	192.168.0.8
	Latitude	10.872
	Longitude	-13.767
	Sal	123.3
	Temperature	31.4
	Ph	0.82
	Battery	0.3
	...	...

mouse_actions_collected_data

Kvp_id	key	value
1	Time	2009-10-11 15:25
	Ip_address	192.168.0.5
	Latitude	11.455
	Longitude	-13.333
	Event	PRESSED
	Button	RIGHT
	X	8
	Y	3
2	Time	2009-10-11 15:25
	Ip_address	192.168.0.5
	Latitude	11.455
	Longitude	-13.333
	Event	RELEASED
	Button	RIGHT
	X	15
	Y	18

Figure 5.7: KVP Database instance Prototype

5.5.3 Technology Analysis

The properties of the relational and key-value data models were compared by [Bai09] and summarized in Table 5.1:

In order to have access to the persisted data, both types of data models support methods: one via structured languages, and the other via abstractions of programming languages.

Table 5.1: Schema-Dependent X Schema-less Databases Compared: Properties

Schema-Dependent Databases	Schema-less Databases
1. Real-world model represented by entities, classified in tables;	1. Real-world model by entities, classified in Domains;
2. Tables composed by columns and rows. Rows are comprised by column values, which have the same schema;	2. Domains are similar to tables, but like buckets that contains items without a pre-defined schema, what enables them to have different schemas;
3. Data Model is defined in advance with a schema, which contains relationships and constraints to enforce data integrity;	3. Items are identified by keys, as well as have a dynamic set of attributes attached to it, however with no schema defined;
4. Data represents "natural" entities, not application specific;	4. Items may represent not only the natural representation of data, but also include application-specific data;
5. Data Normalization is the data structuring process to remove data duplication, as well as establishing data associations through table relationships [Ken83];	5. Since data may be repeated, no data normalization is done, so that integrity is done in the application layer;
6. Data Provenance can be provided through data types such as "timestamp" for time or float for the GPS float-based coordinates;	6. Data provenance can be added through the use of keys and relating values for the time information and location of the data.

[Bai09] summarizes the differences shown in Table 5.2:

Table 5.2: Schema-Dependent X Schema-less Databases: Data Access

Schema-Dependent Databases	Schema-less Databases
1. The basic CRUD (Create-Retrieve-Update-Delete) database operations are performed using a structured language such as the SQL or XPath; 2. Query languages can access data from different tables through joins, containing aggregation functions to apply on the returned values.	1. The CRUD operations are performed via an abstraction of programming language. (E.g. "db.collection.find()", "db.collection.insert()"); 2. Some technologies provide basic SQL-like syntax for filter criteria with some predicates like equals to (=), different than (!=), less than or equals ($\leq$), greater than or equals ($\geq$), among others; 3. The data and application integrity logic is placed in the application layer.

Finally, each data model provides its own APIs to the integration with external systems. [Bai09] differentiates each of the application interfaces provided by each of them, as summarized on table 5.3.

The data model used by each of the selected database systems can be summarized as follows:

- **MySQL**: uses the Relational Data Model with the use of SQL;
- **TinyDB**: uses the Relational Data Mode with the use of SQL;
- **MongoDB**: uses Key-Value Pair Data Model with an abstraction of programming language functions calls;

Table 5.3: Schema-Dependent X Schema-less Databases: Application Interface

Schema-Dependent Databases	Schema-less Databases
1. Have their own specific API or through ODBC (Open Database Connectivity); 2. Data is stored in a format that represents its natural structure, and for this reason, in a single or distributed fashion.	1. Tend to provide Web Services Interfaces such as the REST [Fie00]; 2. [Bai09] claims that data is stored in a more effective way, requiring only code plumbing for the relational code;

- **IBM DB2**: uses the Relational and Structured models with SQL and XPath, respectively.

5.6 Analysis of the Data Description

Data Description is the method used to describe the collected data from sensors, especially those produced by real-time data streams. Since that type of data does not include descriptions of its values, data provenance offers guidelines to improve the description of the collected data.

In addition to data provenance, annotation is another technique used to better describe the collected data from sensor devices. Its main purpose is to provide a means of identifying observations, creating indexes based on simple keywords. For an environmental sensor network, this is invaluable used to better manage the collected data.

5.6.1 Use of Provenance-Aware Data

The inclusion of provenance data into the collected data enriches the description of a given event, and is always useful to determine different aspects of the data. As mentioned in

Chapter 3, data provenance helps identify the nature of the data by the identification of the properties of a given sensor device, for example. Similarly, provenance adds spatial-time data in order to correlate the observed data with time a given location and time of occurrence.

Although metadata gives more information about the data, it just represents additional data inserted together with the raw data collected from the sensor devices may contribute with a decreasing amount of disk storage. Therefore, the use of provenance-away data increases the required disk space used to save the collected data from sensor devices.

5.6.2 Use of Annotations

Annotations are used to describe the collect data even more, since the additional "observation" of the data are being stored together with the data. For example, the use of tags to a given entity may help identifying and different observations regarding the data collected from sensor devices. As shown in Figure 5.7, the addition of the key "tag" was used to describe the observations that happened when the event of "oil spill" occurred. In this way, tags can be indexed for better search of the collected data.

On the other hand, the use of annotations behaves the same as the use of Provenance data. It will require additional disk space to store the collected data from sensors.

5.6.3 Technology Analysis

All the database systems technology supports the use of Data Provenance or Annotations. The Key-Value Pair data model has the advantage of not maintaining a schema design, and therefore, additional keys of descriptions can be added at any time without requiring schema changes. On the other hand, the relational data model requires schema changes to support the addition a new table column.

- **MySQL**: supports data provenance and annotations via column types on the entities, requiring schema changes;
- **TinyDB**: supports data provenance and annotations via column types on the entities, requiring schema changes. Different projects have suggested changes to the SQL language used to give better support to spatial-temporal data [LKK07].

- **MongoDB**: supports both data provenance and annotations with the definition of a new key on a given collection. No changes are required to the addition of the keys;
- **IBM DB2**: supports data provenance and annotations via column types on the entities, requiring schema changes.

5.7 Analysis of the Query Processing Mechanism

As a direct result from the previous section, the use of **Centralized Query Processing** is indicated for NetBEAMS, because the SF-BEAMS data goes directly to the individual network sink and, therefore, a centralized point of the data.

An advantage of centralized data management and query processing is simpler than the in-network query method. Data is verified by a straightforward database management system without the risk of data being unavailable, when the data is spread among the network nodes. On the other hand, when the Centralized Query processing is used, problems may occur. Depending on the size of the datasets and the database technology, performance may be a concern due to the alleged Funneling Affect [KGH07], since the point-of-traffic is concentrated in the centralized system.

In order to mitigate the problems originated by this query processing mechanism, techniques such as Data Replication can be used.

5.7.1 Technology Analysis

All the database systems technology support a centralized query system. In addition to that, they can be sorted into different distributed nodes when used in a partitioned environment.

5.8 Analysis of the System Organization

Sensor Network data can be saved into database system that are organized by either a Centralized or Distributed System. While a centralized database system may be considered to be easier to manage, it may fail to adjust to changes of requirements to manage data.

On the other hand, a distributed database system can be used to implement a data-centric solution. Therefore, a database system that can be organized in both approaches is a good candidate to be used a data persistence technology for sensor networks.

Distributed Systems can improve the performance of data management by providing techniques for parallel data processing such as Map Reduce [DG04, CDG$^+$08]. It also helps decreasing the system load and bottleneck problems related data creation and retrieval. It is also seen as an optional mechanism of reducing the funneling effect of a centralized system by directing queries to a given data partition [KGH07]. Data Replication [PGVA08] provides a way to decentralize data access by providing duplicated data that is intended to be used on reading operations.

Although a distributed system may provide better support to the problem being solved, it may be a difficult solution to propose. Since it is designed as set of cooperating subsystems, multiple points of failure exist in this type of system, making it difficult to be operated. Furthermore, it requires more technical knowledge expertise to manage the different systems that composed a distributed database management system.

5.8.1 Technology Analysis

- **MySQL**: provides support to data replication;
- **TinyDB**: it does not provide support to data replication;
- **MongoDB**: supports data replication;
- **IBM DB2**: supports data replication.

5.9 Other Non-Functional Analysis

Other non-functional requirements to be considered are related to the functionalities and type of database systems.

- **Open-Source**: Given the nature of the project, the technology to be used must be free of charge [vKS07], that is, no costs involved in the adoption a such technology;

along with the support from the community;

- **Native APIs**: In order to consider the scope of this work defined in the last chapter, the solution for persisting collected sensor data must be not only limited to a technology that provides data access, such as SQL, but also by other data access mechanisms **data access mechanisms**;
- **Export Feature**: the technology must support export capabilities during its execution such as hot backps.

5.9.1 Technology Analysis

Most of the technologies listed provides access to the data sets through the use of drivers in different programming and scripting languages such as Java, Python, Perl, among others.

- **MySQL**: Is open-source, supports hot backup and contains scripting APIs from the community; from the community. Also APIs are available for the majority of languages;
- **TinyDB**: Not open-source nor does it support hot backup;
- **MongoDB**: An open-source database system with an increasing support from the community, offering native APIs in different programming languages and scripting languages;
- **IBM DB2**: Not open-source, but offers support from its community.

Only MySQL and mongoDB supports the non-functional requirements defined in this section.

5.10 Global Analysis Results and Technology Selection

Each of the databases selected was scored according to the support provided to the different types of taxonomies and requirements defined for a selection of a persistence technology for NetBEAMS. Table 5.4 summarizes these scores using the (+) and (-) notation. Since they all provide support to data archival for a small volume of data, these parameters are not included in the table.

In general, all the selected databases can be used for the purpose of data storage, also associated with the location of the collected data being in an external storage. Since the data volume produced by NetBEAMS is small, a typical centralized server can be used to provide management of a database management system, and consequently a centralized query mechanism. Additionally, a distributed system can also be used in the case of future system expansion. All technologies, but TinyDB, fit to these requirements.

As used in different projects, schema-dependent models such as the relational data model can be used to design the collected data entities. As shown in Figure 5.4, the specification of a schema for the YSI Sonde Data is achieved. However, what if a new sensor device needs to be included into the model? A refactoring would be necessary to accommodate this new requirement. All database presented technologies, but mongoDB, use the relational model.

Considering the nature of sensor devices as a list of properties, where the collected data values are labeled and represented by keys, the Key-Value Pair data model is proposed in Figure 5.6. Considering that this technique can support any different device with different properties, this would be a good candidate to be used as a persistence model to support any properties list of any sensor device. However, this approach results in different problems.

Table 5.4: Technologies Selection Scoring

Database	**MySQL**	**TinyDB**	**MongoDB**	**IBM DB2**
Centralized Query	+	+	+	+
Schema-less	-	-	+	-
Non-Structure Query	-	-	+	-
Provenance Support	+	+	+	+
Distributed System	+	-	+	+
Data Partition	+	-	+	+
Export Capabilities	+	-	+	-
Native APIs	+	+	+	+
Open-Source	+	+	+	-

Taking into account the support to data provenance, the use of time-series for the solution results in data repetition, as shown in the values of the column "time". Thus, it represents an insertion anomaly on the First Normal Form of the relational model [Cod70], where the collected data metadata must be repeated. Furthermore, [Gos06] describes that the use of this strategy in regular relational databases as one that presents poor retrieval performance, due to the concentration of data in a single database table. Therefore, this strategy is not well suited to provide persistence for data collected from sensor networks, described by the guidelines of Data Provenance.

Although the use of schema-less database systems in sensor networks has not been reported by the literature review, the schema-less data model can be used as opposed to the schema-dependent because it does not require the definition of a static schema. As shown in Figure 5.7, different instances of the YSI Sensor are represented not only by the properties of the sensor device, but also by data provenance and annotations. Data provenance is prepresented by the keys "time, "latitude" and "longitude", while Annotations by the key "tag". Furthermore, since the the method of data access of the KVP databases is through the use of an abstraction of programming languages, it may also offer a better support to non-skilled users such as the RTC marine biologists. mongoDB is the only technology that provides such support.

All the discussed tools provide access to data through an additional API, but mongoDB also provides it as a query method. Without no reason to further discussions, mongoDB was considered as a good candidate to be explored as a data persistence solution for NetBEAMS. Moreover, it also provides all the other requirements as a technology as such being an open-source application and that it can be used as a single server or a distributed system, requirement necessary to develop a Data-Centric.

To conclude, this chapter evaluated different database systems filtered from Chapter 2. After presenting the database system "contenders", the analysis of each of the taxonomies was conducted for each of the technologies, culminating with the technology selection comparison. The following chapter describes the design of the solution using mongoDB as the primary solution for the backend.

Chapter 6

DSP Data Persistence: Component Design and Architecture

The mongoDB database system was selected, as stated in Chapter 5, to store the collected data from the SF-BEAMS sensor network. NetBEAMS is based on a component-based architecture, where each of the components can play different roles during their lifecycle. It uses the abstraction of Data Producers and Data Consumers: while a component produces data to the system, the latter consumes [PJS+09]. In this way, a DSP Data Component was designed and implemented to provide the integration between NetBEAMS and the mongoDB database. This integration is based on the Data Sensor Platform (DSP), the kernel software stack provided by NetBEAMS, as documented in [dS09].

This chapter follows the steps taken in Software Engineering guidelines to analyze and design a software component using Object-Oriented Analysis and Design [Lar04] using the techniques from the Agile and XP Scrum [Els08] processes. First, Section 6.1 gives an overview of the problem by analyzing a basic scenario of the data collection process used in NetBEAMS. Then, the Section 6.2 describes the requirements for the design of the new component, followed by the specifications of the data model used for the mongoDB database in Section 6.3. Lastly, the high-level architecture of Section 6.4 gives the foundation of the DSP Data Persistence component design of Section 6.5.

Details about the implementation of this design are located in the Appendix Section A.1.

6.1 Business Process Analysis

As described in [dS09], the DSP Platform was designed to support SF-BEAMS. As shown in Figure 6.1, the event of data collection starts by the DSP Component deployed on each DSP Gateway Node [dS09], by the event "Measurements X Collected". When this DSP Client component finishes the data extraction process, a new one starts. First, each of the co-located DSP Client processes the collected measurements and creates a DSP Message, embodying the collected data from the associated sensor device. Each DSP Message is added into a local temporary outbound queue that waits to be transmitted. Then, an asynchronous data transmission event occurs at a specified rate, packaging all DSP Messages into the body of a DSP Messages Container in order to be serialized into XML (shown in Listing A.19 on page 136) and transferred to the DSP Server host, a centralized server that receives all the collected data embodied into each of the DSP Messages.

The goal of the new DSP Data Persistence component is to proxy the payload from the DSP Messages to the new suggested Measurements Database. Once the DSP Server receives all the messages in a container, it processes the messages and sends their payloads to the Measurements Database. This process requires a new Database Configuration to be provided

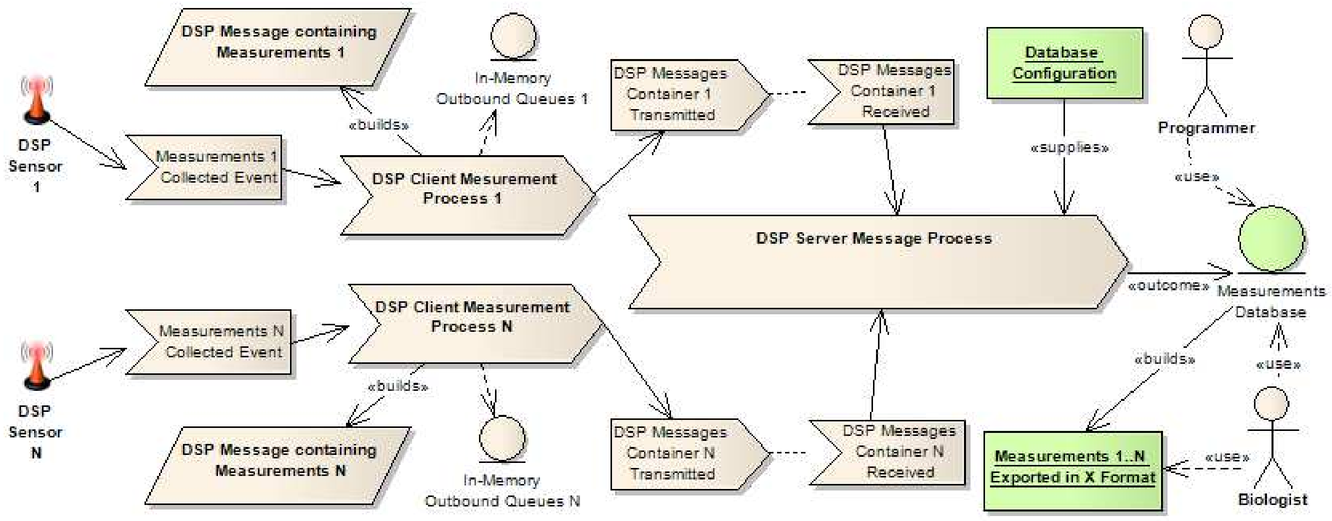

DSP Message containing Measurements 1
In-Memory Outbound Queues 1
DSP Messages Container 1 Transmitted
DSP Messages Container 1 Received
Database Configuration
Programmer
DSP Sensor 1
Measurements 1 Collected Event
«builds»
DSP Client Mesurement Process 1
«supplies»
«use»
DSP Server Message Process
«outcome»
Measurements Database
DSP Client Measurement Process N
Measurements N Collected Event
«builds»
«use»
DSP Sensor N
«builds»
DSP Message containing Measurements N
In-Memory Outbound Queues N
DSP Messages Container N Transmitted
DSP Messages Container N Received
Measurements 1..N Exported in X Format
«use»
Biologist

Figure 6.1: UML Business Process Model diagram - Adding Persistence for NetBEAMS

to the DSP Server host, as well as the installation and setup of the new Measurements Database. In this way, different users of the collected data, such as Programmers and Biologists, can access the collected data directly from the Measurements Database and reuse them by Exporting specific observation parameters to different formats of their interest.

6.2 Requirements Specification

This section outlines the functionalities and constraints that the developed solution must perform and follow, respectively, obtained by analyzing the main data collection scenario described in the previous section and reviewing the requirements defined by the RTC staff members.

6.2.1 Functional Requirements

The primary goal of the system is to allow users to have easy access to the collected data from *NetBEAMS* using a schema-less database technology. mongoDB provides access to the stored data without requiring the use of a Structured Query Language. Instead, that technology provides a programming language abstraction that makes the access to the stored data easier [BDD09]. Using Agile User Stories to describe Use Cases Diagrams [use07], Figure 6.2 enumerates these functionalities as customers conversations, starting with a fictitious persona, the action it desires to perform and the outcome of the action, a requirements gathering pattern defined in Scrum [Els08]:

- **Search Measurements by Sensor Properties**: "As a marine biologist, I would like to search observations by filtering values of the sensor device's properties such as temperature or salinity, so that I can find specific associated values to the observation.";
- **Search Measurements by Sensor Location**: "As an oceanographer, I would like to search observations that took place at the geographic coordinates (x latitude, y longitude), so that I can assess the area around the given coordinates.";
- **Search Measurements by Data collection date**: "As a marine biologist, I would

like to search observations that took place last week, so that I can assess past environmental conditions";

- **Annotate Existing Measurements**: "As a estuarine ecologist, I would like to annotate observations from the time the oil spill occurred in the San Francisco Bay, so that I can maintain historical evidence of the impact of such event.";
- **Export Data to Spreadsheets**: "As a scientist from RTC, I would like to analyze of the observed data from yesterday using a spreadsheet, so that I can verify measurements using Microsoft Excel.";
- **Export Data to OPeNDAP format**: "As a biologist, I would like to export the collected data produced during this month using the OPeNDAP data format, so that I can collaborate with other research groups that use this data format.";
- **Access the data through API, Programming Languages**: "As a marine biologist who learned the Python scripting language, I would like to write a software

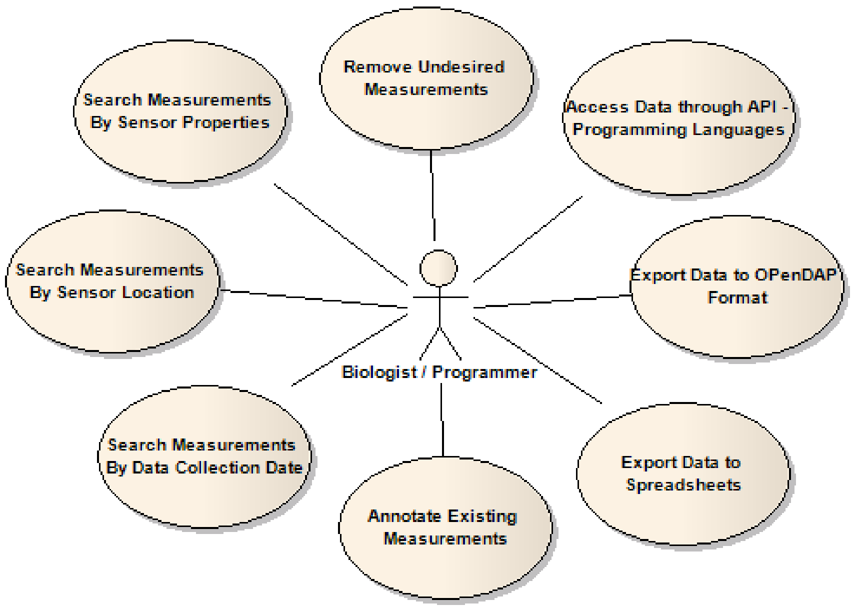

Remove Undesired Measurements
Search Measurements By Sensor Properties
Access Data through API - Programming Languages
Search Measurements By Sensor Location
Export Data to OPenDAP Format
Biologist / Programmer
Search Measurements By Data Collection Date
Export Data to Spreadsheets
Annotate Existing Measurements

Figure 6.2: UML Use Case diagram for Persistence Functions

that connects directly to the database, so that I can implement a customized analysis system for NetBEAMS.";

- **Remove Undesired Measurements**: "As an oceanographer, I would like to remove specific observations collected yesterday, so that the research group does not use 'junk data'.".

As one can see, the basic users' functionalities are related to the access to the collected data using different parameters related to the each observation's properties or implied meta-data such as time of collection or location. In order to do so, the functions to be performed were developed following the specifications of the Data Sensor Platform (DSP) architecture, as described in [dS09]. In order to collect data sent to the NetBEAMS server, the new DSP component must act as a Data Consumer. The functionalities provided by this component can be seen in the UML Use Cases diagram [BRJ99] in Figure 6.3, and summarized as follows:

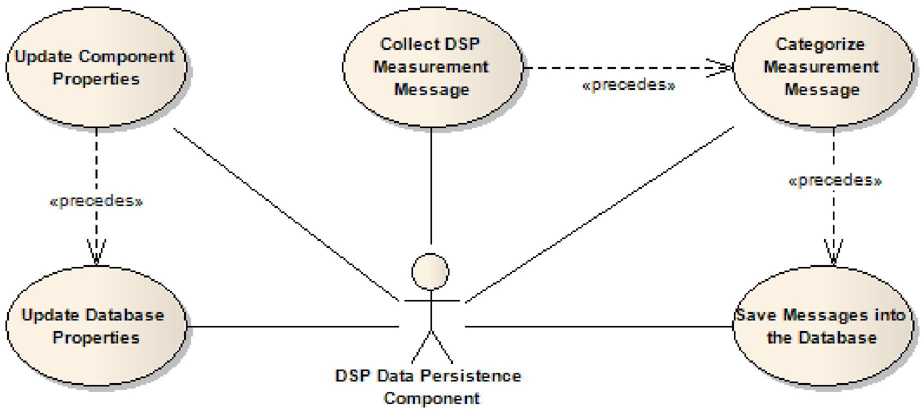

Update Component Properties
Collect DSP Measurement Message
«precedes»
Categorize Measurement Message
«precedes»
«precedes»
Update Database Properties
Save Messages into the Database
DSP Data Persistence Component

Figure 6.3: UML Use Case diagram for the Persistence Component

- **Update Component Properties**: the component must be configurable, with its own set of properties, such as the flush rate used to send data to the database system;

- **Update Database Properties**: the component must be able to update the mongoDB database settings such as IP Address and communication port;

- **Collect DSP Measurement Message**: the component must be able to receive and temporarily retain the collected messages from different the sensors devices;
- **Categorize Measurement Message**: the component must be able to categorize the DSP Message, as it arrives, by its properties such as transaction time, and originating DSP Message ID;
- **Save Messages into the Database**: the component must be able to send the payload of the DSP Messages to the database.

6.2.2 Non-Functional Requirements

Different constraints were required to execute the functional requirements described in the previous section. Those constraints are listed as different non-functional requirements as follows:

- The solution must maintain the received DSP Measurement Messages in-memory for a specific period of time in order to decrease the write load on the database (flush rate);
- The persistence storage system must be able to cope with the Data Volume produced by SF-BEAMS on daily basis;
- The persistence storage system must be able to scale without minimal architectural and schema changes;
- The persistence storage system must be able to provide access to the persisted data using an programmatic abstraction interface, as the authors of [BDD09] proposed.

Driven by the requirements defined, the definition of the keys of the key-value pairs model, used to describe the collected data from the sensor devices and their associated metadata, is the foundation for the integration between the DSP Data Persistence component and the mongoDB database. Next section details the data model design that specifies those keys.

6.3 Data Model Design

MongoDB is defined as a Document-Oriented Database system, which is based on the abstraction of the Key-Value-Pair data model, using a binary version of JSON [Ste08] as its main data representation. MongoDB does not require the creation of a database schema, rather the definition of keys that comprise a document on the database organized into separate collections of relating documents. In order to describe the keys of a document, the guidelines provided by the Data Provenance taxonomy was used. The key-value pairs of any instance of the collected data must include the following key properties:

- **Location**: properties that describe the location of the data. For example, the host machine name, IP address, or the GPS coordinates [LSPS07] from the sensor device;
- **Identity**: what uniquely identifies the data among others, produced by other sensor devices in the network such as the DSP Message ID using UUID[1];
- **Time-Dimensions**: when the data was recorded [MMLM07] and when the transaction occurred [YG08].

After analyzing the DSP Message Structure, as well as the YSI Sonde Data Handler of the DSP Platform architecture documentation [dS09], the design of the document developed to identify both the data and metadata of the collected data is organized as follows:

- **message_id**: Defines the DSP Message ID that carried the observation from the sensor device to the DSP Server. This ID can be used during network and database inspections in order to track messages for the DSP Platform;
- **sensor**: Defines the properties of the sensor device;
  - **ip_address** Defines the IP Address of the sensor device that produced the data;
  - **location**: Defines the geographical coordinates of the sensor device, using the Decimals Degree format [dec09];

[1]Universally Unique Identifier

    - **latitude** the GPS latitude coordinate of the position of the sensor device;
    - **longitude**: the GPS longitude coordinate of the position of the sensor device.

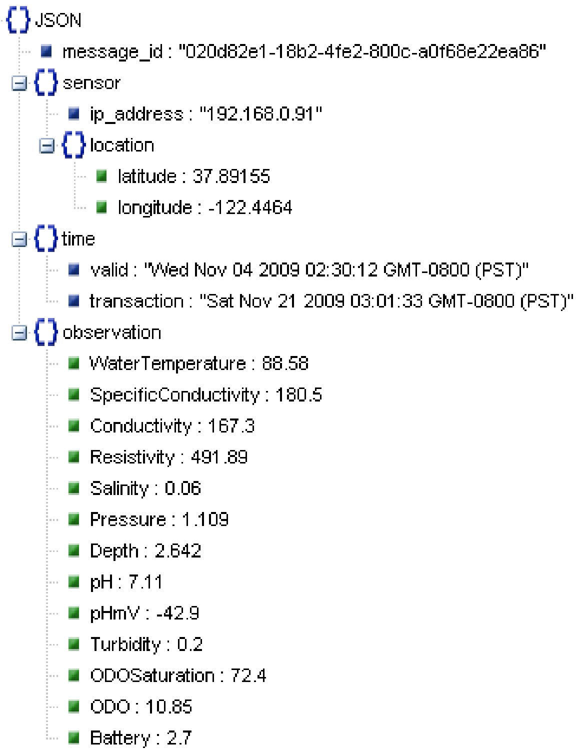
- **time**: Defines different time dimensions according to Data Provenance;
    - **valid**: it is extracted from the container's date and time of creation, which was collected during the sensor device's reading;
    - **transaction**: it is extracted from the DSP message container's creation time and it is used to identify when the transaction, which transferred the message from the DSP Gateway Client to the DSP Server, occurred;
- **observation**: carries the observations of a given sensor device. For example, the observation key of for the YSI device [YSI08] may contains all the subsets of the properties defined for the device, such as battery, temperature and salinity, among others.

MongoDB uses JSON data structure [Ste08] to represent a document, as the example shown in Figure 6.4 on page 64. Any number of keys can be in the root branch of the tree, as well as in sub branches of the document structure. The values of the tree are concentrated in the leaves of the tree. Listing A.36, on page 176, shows the instance of this design on mongoDB database. Based on this foundation, the design of the DSP Data Persistence component evolved from a top-down approach to meet this data model design.

It is important to note that mongoDB does not use any structure language, like the use of the SQL in relational databases, to access the persisted data. In order to access the values of the documents, the "dot notation" [Mer09a] is used to reach each of the tree nodes. For example, the use of "observation.WaterTemperature" and "sensor.location.latitude" give the values of the data "water temperature" and the metadata "latitude" of the collected data, respectively.

6.4 High-Level System Architecture

Considering the business analysis described in Section 6.1, the addition of a data persistence layer to the DSP Platform can be achieved by adding a Data Consumer (DC) as described



```
{} JSON
   message_id : "020d82e1-18b2-4fe2-800c-a0f68e22ea86"
 {} sensor
     ip_address : "192.168.0.91"
   {} location
       latitude : 37.89155
       longitude : -122.4464
 {} time
     valid : "Wed Nov 04 2009 02:30:12 GMT-0800 (PST)"
     transaction : "Sat Nov 21 2009 03:01:33 GMT-0800 (PST)"
 {} observation
     WaterTemperature : 88.58
     SpecificConductivity : 180.5
     Conductivity : 167.3
     Resistivity : 491.89
     Salinity : 0.06
     Pressure : 1.109
     Depth : 2.642
     pH : 7.11
     pHmV : -42.9
     Turbidity : 0.2
     ODOSaturation : 72.4
     ODO : 10.85
     Battery : 2.7
```

Figure 6.4: Data Design using JSON Data Structure

by [dS09]. Based on the Requirements specification, this component must be responsible for the extraction process that takes the payload of the body of a filtered DSP Measurement Message and converts it to the key-value data model defined in the previous section. This process guarantees that the collected data is inserted to a running mongoDB database system and covers the use cases defined. Figure 6.5 provides an architectural overview of the DSP Server node with the loosely-coupled components, highlighting the inclusion of a new DSP Component called “DSP Data Persistence”.

As it is depicted the diagram 6.5, the DSP Data Persistence can be added into the existing architecture as an independent plug-and-play component. The database system and the necessary drivers are also added without any changes to the existing DSP infrastructure. First, the DSP Messages are delivered to the DSP server through the DSP Wire Transport Server, as shown by flow 1. Next, the DSP Platform is responsible to forward a copy of

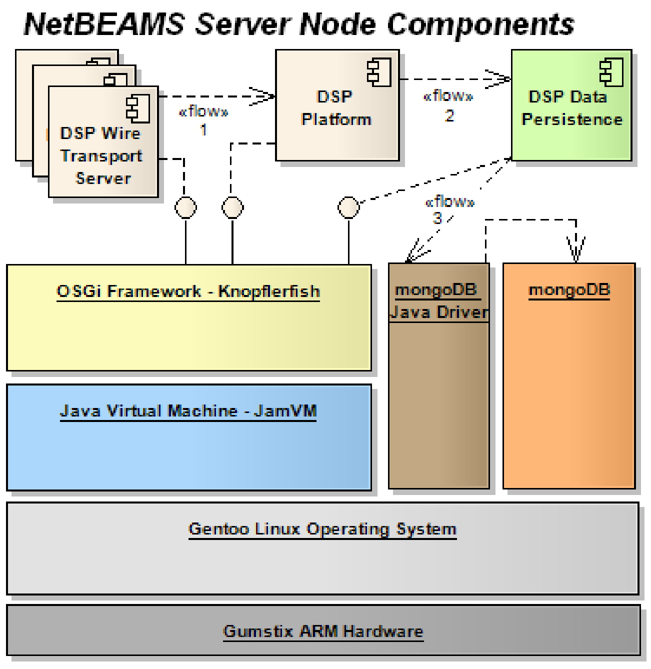

NetBEAMS Server Node Components
DSP Wire Transport Server
«flow» 1
DSP Platform
«flow» 2
DSP Data Persistence
«flow» 3
OSGi Framework - Knopflerfish
mongoDB Java Driver
mongoDB
Java Virtual Machine - JamVM
Gentoo Linux Operating System
Gumstix ARM Hardware

Figure 6.5: UML Components diagram of the DSP Data Collector

the Measurement messages directly to the DSP Data Persistence as shown by flow 2. Then, the sole responsibility of the DSP Data Persistence is to transfer the received measurement messages to the database system by using the connection drivers, as shown by flow 3.

Any DSP Component is essentially an OSGi Bundle [TV08], a unit of deployment in a Java JAR format [CWH00] containing the Java classes and a descriptor manifest artifact that identifies the major components of the package, which services the component requires to use and which ones it will optionally publish during its activation and lifecycle. Details about the deployment of the DSP Data Persistence component, as well as about the OSGi environment used, are given in the Appendix section A.1.1.

The following sections focus on the DSP Data Persistence component design in the context of an OSGi bundle [osg08].

6.5 DSP Data Persistence Component Design

The proposed Data Persistence component must follow the DSP Component requirements, which specifies the design-pattern of Data Producer and Consumer mentioned earlier in this chapter. As seen in the UML Package Dependency Diagram in Figure 6.6, a new DSP Component has dependencies on three different packages: the OSGi Framework and the DSP Platform responsible to integrate a new component into the micro-kernel, and the mongoDB Java driver API that connects to the mongoDB server.

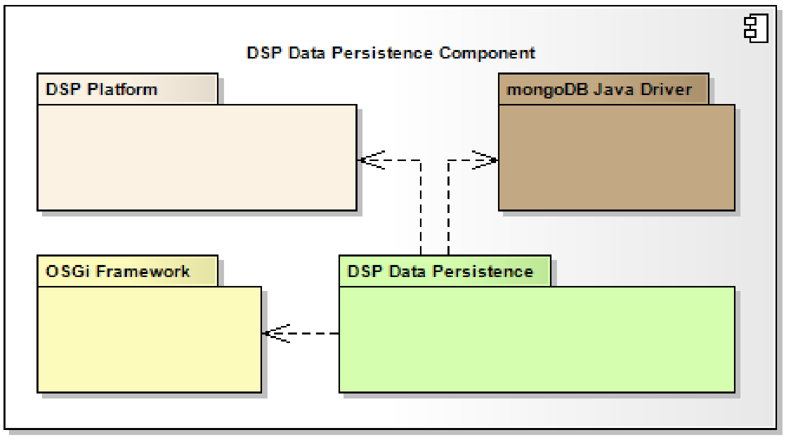


Figure 6.6: UML Components diagram of the DSP Data Collector

In order to use a DSP Component, one must first activate it into the OSGi Framework the DSP Platform runs on. Both the activation and the design of how the new component acts as a Data Consumer is explained in the following sections.

6.5.1 DSP Data Persistence Activation and Message Delivery

The DSP Data Persistence component starts with an OSGi Activator class, and the implementation of the DSP Component, responsible for providing services to the platform [dS09]. Based on these specifications, the following set of classes, shown in Figure 6.7, depicts the implemented classes for the new DSP Data Persistence component.

The class DSPDataPersistenceActivator is the main class responsible for the activation of the class DSPDataPersistence, as it extends from the OSGi class BundleActivator. The

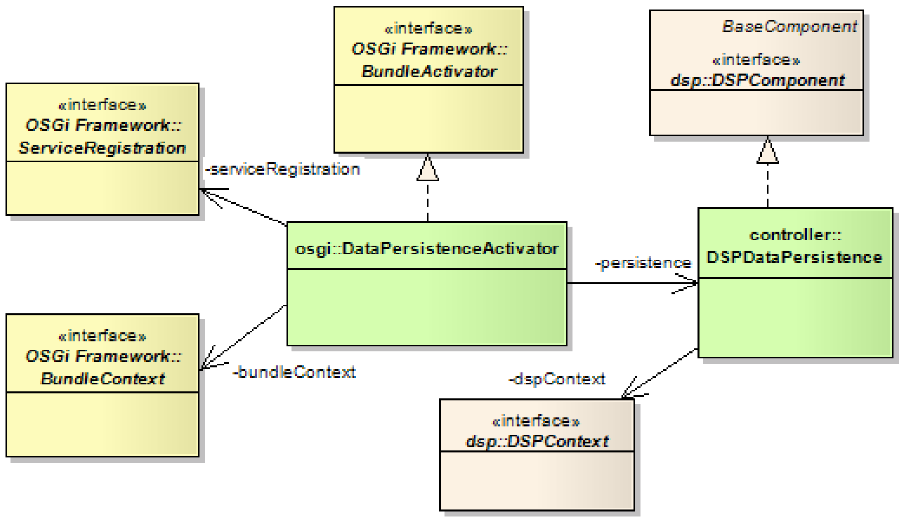


Figure 6.7: UML Class diagram for the DSP Data Persistence Activator and Component

BundleActivator maintains references to the classes BundleContext and ServiceReference, both from the OSGi Framework, in order to maintain the component managed by the OSGi Framework and register the class DSPDataPersistence as a service to the OSGi environment. Therefore, the class DSPDataPersistenceActivator is the main communication interface between the DSP Platform and the OSGi Framework, are responsible for the orchestration of the application lifecycle.

The class DSPDataPersistence implements the interface DSPComponent, responsible to provide the contract between the component and the DSP platform. As a consequence, this class inherits all the behavior described in [dS09] necessary to be executed in the DSP Platform in a given context (through the instance of class DSPContext). The different roles played by the class DSPDataPersistence are defined as follows:

- **Data Consumer**: receives any measurement message from remote sensors;
- **Data Producer**: transforms any received measurement message into the format defined in Section 6.3, in order to be sent to the database.

The class DSPDataPersistenceActivator is responsible for starting and stopping, the services provided by the DSPDataPersistence component, as this behavior is inherited from the

OSGi class BundleActivator from the OSGi Framework. Following the specification of the DSP Architecture [dS09], Figure 6.8 shows a UML Sequence Diagram that depicts the steps of the activation of the DSP component, and summarized as follows:

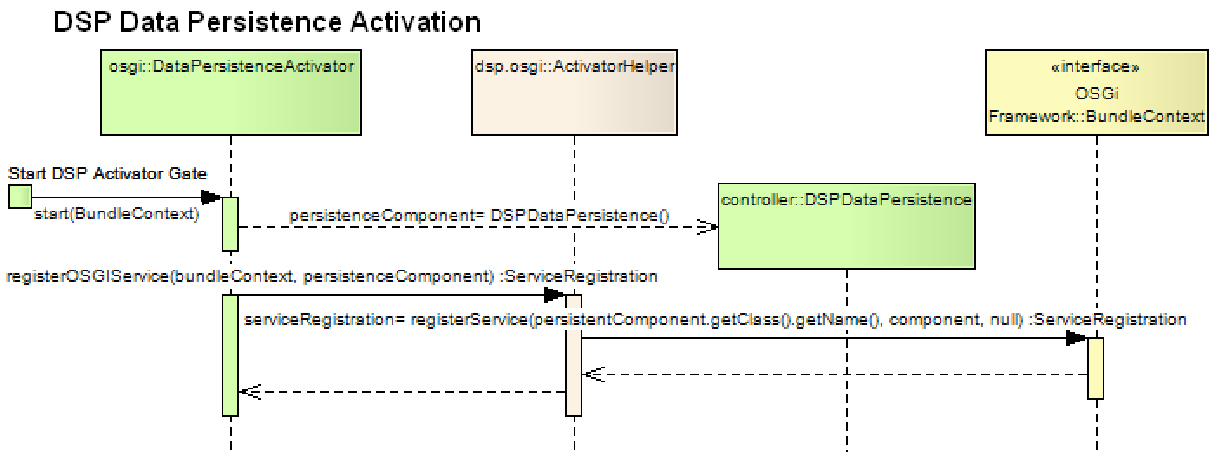

DSP Data Persistence Activation
osgi::DataPersistenceActivator
dsp.osgi::ActivatorHelper
«interface» OSGi Framework::BundleContext
Start DSP Activator Gate
start(BundleContext)
persistenceComponent= DSPDataPersistence()
controller::DSPDataPersistence
registerOSGIService(bundleContext, persistenceComponent) :ServiceRegistration
serviceRegistration= registerService(persistentComponent.getClass().getName(), component, null) :ServiceRegistration

Figure 6.8: UML Sequence diagram describing the DSP Data Persistence bundle activation

1. During its activation, the DSP Platform will get the name of the DSP Data Persistence component from the configuration artifact config.xml as seen in Listing A.12 on page 119;

2. After being selected based on the configuration priority, the DSPBundle artifact is identified and installed by creating an instance of the class DSPDataPersistenceActivator, making a call to the method start();

3. During the activation, an instance of the class DSPDataPersistence is created and registered as an OSGi Service;

4. Upon registering the DSP Data Persistence component, the DSP Platform listens for the event serviceUpdate() and triggers the last operations to bootstrap the DSP Component:

   (a) Initialize the component by calling the method initComponent();

   (b) Start the component by calling the method startComponent();

(c) Bootstrap messages by calling the method delivery(). This is where both the DSP Measurements Messages and Update Messages are delivered by the DSP Broker. This concept is based on the listener of Observer Design-Pattern [HK02].

Any configuration parameter required by a DSP Component is sent during its bootstrap process through the use of an optional DSP Update Message, as shown in Listing A.15 on page 122. The initialization of the DSP Component Activator and the DSP Component registration are also shown in Figure 6.8.

As soon as the process "Start DSP Activator Gate" is executed, an instance of the class DSPDataPersistence component is created to be registered into the OSGi Framework using the DSP Platform's helper class ActivationHelper. It is responsible to proxy the method call to the class BundleContext, which makes the actual registration of the DSPDataPersistence component as an OSGi Service. At this point, the DSPDataPersistence is installed and on the state "Active", as defined by the OSGi Framework (Refer to [dS09] for details).

Once the DSP Data Persistence component is concurrently running with other components from both the DSP Platform and the OSGi Framework, it is able to receive DSP Messages. The following section will explain how to bootstrap the DSP Data Component in order to receive Measurement Messages.

6.5.2 Delivering Bootstrap and Measurement Messages

Although the DSP Data Persistence component has been instantiated as of Figure 6.8, the process of delivering messages to the DSP Data Persistence component has yet to be completed. As defined by the requirements, the DSP Component must execute its function concurrently, using the configuration parameters during its bootstrap process. The DSP Platform delivers any bootstrap message defined in the DSP Platform runtime environment, as explained in the Appendix section A.1.2. However, to deliver DSP Measurement Messages, a new filter must be provided. As explained in [dS09], the DSP Broker uses the DSP Matcher's rules to select to Data Consumers configured in a given DSP. For this reason, a rule that filters any remote measurement message received by running DSP Server is used, as shown in Listing A.13 on page 120. The rule defined in line 63 express that all messages created by any producer on any DSP Host, either client or sever, has as delivery target the DSP

Data Persistence component. In this way, when DSP Measurement Messages are delivered, the DSP Broker delivers the DSP Message to the DSP Data Persistence Component.

The UML Class Diagram for the participating classes of the DSP Data Persistence component is shown in Figure 6.9. The class DSPDataFlusher is responsible for running as a worker thread, whose function is to verify if there are messages to be flushed to the Database by contacting the Singleton class [HK02] TransientPersistenceLayer, responsible to keep track of the DSP Messages received when the DSP Broker delivered them to the DSP Data Persistence component.

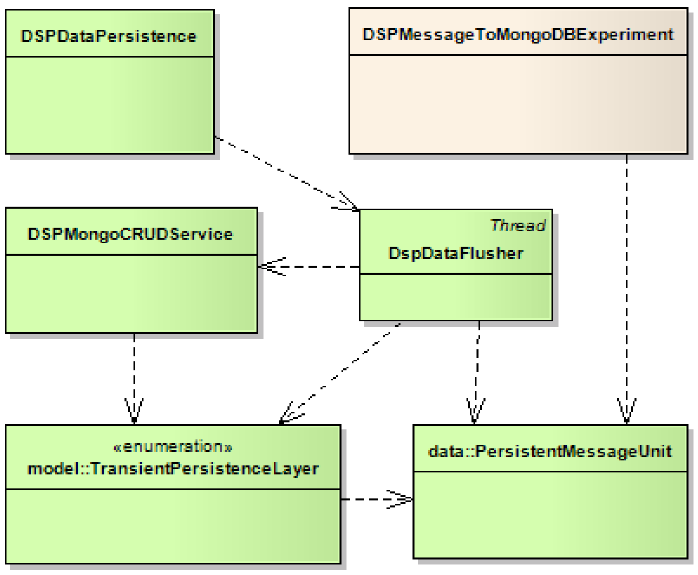

DSPDataPersistence
DSPMessageToMongoDBExperiment
DSPMongoCRUDService
Thread
DspDataFlusher
«enumeration»
model::TransientPersistenceLayer
data::PersistentMessageUnit

Figure 6.9: UML Class diagram with DSP Data Persistence and Participating classes

Upon receiving a new DSP measurement message, the DSP Data Persistence component keeps any received message in a temporary local cache, as described by the non-functional requirements. In this way, the DSP Data Persistence component maintains a dependency on the Singleton class TransientPersistenceLayer, responsible for encapsulating the DSP Message into an instance of the class PersistentMessageUnit, to be stored in the defined local cache.

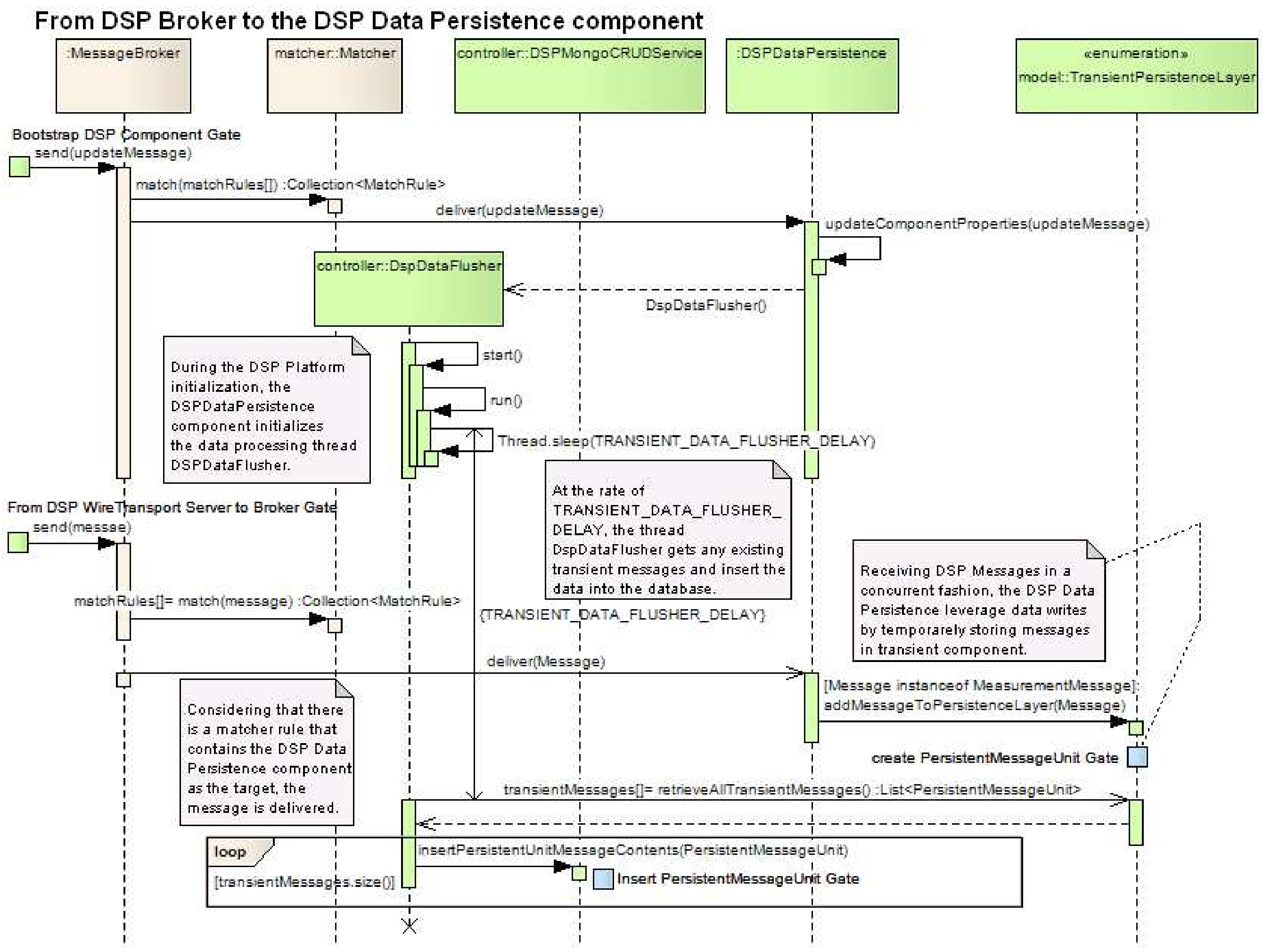
According to the DSP Broker model described in [dS09], the DSP Matcher needs to be updated with the addition of a matching rule that filters a copy of any Measurement Message to the DSP Data Persistence component. As it is shown in Figure 6.10, the worker Thread DSP Data Flusher is the link between the Transient and Persistent layers, as it depends on the references from the classes TransientPersistenceLayer and the DSPMongoCRUDService.The class DSPMongoCRUDService is responsible for extracting the collected data from the body of the DSP Message and creating the keys designed to be persisted in the mongoDB Database.

The DSP Broker retrieves the list of matching rules by contacting the Matcher. As the DSP Platform has already loaded all the Matching rules, a copy of an analyzed DSP Message is delivered to the DSP Data Persistence component specified by a matcher rule. However, the UML Sequence Diagram depicted in Figure 6.10 shows two different moments of message delivery:

- A bootstrap DSP Update Message is delivered to the DSP Data Persistence component when the DSP Platform is initializing the component. For example, the class DSP-DataPersistence can initialize the class DspDataFlusher to flush temporary messages into the database at the rate of TRANSIENT_DATA_FLUSHER_DELAY, defined in seconds. Listing A.15, on page 122, shows an implementation of a bootstrap message for the DSP Data Persistence;

- Any DSP Measure Message can be persisted by the DSP Data Persistence component. The component will add the message to a temporary memory location to decrease I/O on the database, when the DSP Broker delivers a DSP MeasureMessage to this component, as depicted by the UML Gate element [gum08] "create PersistentMessageUnit Gate". This last Gate describes the simple procedure to add the message into the transient persistent layer, as shown in Listing A.19 on page 136.

The class PersistenceMessageUnit is the major transient persistence unit that carries references regarding the originating DSP Message that was collected by the DSP Data Persistence component. It is composed by an instance of the class SensorLocation and contains a reference to the PersistentMessageState enumeration. The former identifies which sensor produced the collected data from the DSP Message by the IP address, while the latter identifies whether the PersistenceMessageUnit has been saved into the Database or not as depicted by the UML State Diagram in Figure 6.11. The class SensorLocation was developed


From DSP Broker to the DSP Data Persistence component
:MessageBroker
matcher::Matcher
controller::DSPMongoCRUDService
:DSPDataPersistence
«enumeration»
model::TransientPersistenceLayer
Bootstrap DSP Component Gate
send(updateMessage)
match(matchRules[]) :Collection<MatchRule>
deliver(updateMessage)
updateComponentProperties(updateMessage)
controller::DspDataFlusher
DspDataFlusher()
start()
run()
Thread.sleep(TRANSIENT_DATA_FLUSHER_DELAY)
From DSP WireTransport Server to Broker Gate
send(messae)
matchRules[]= match(message) :Collection<MatchRule>
{TRANSIENT_DATA_FLUSHER_DELAY}
deliver(Message)
[Message instanceof MeasurementMessage]:
addMessageToPersistenceLayer(Message)
create PersistentMessageUnit Gate
transientMessages[]= retrieveAllTransientMessages() :List<PersistentMessageUnit>
loop
[transientMessages.size()]
insertPersistentUnitMessageContents(PersistentMessageUnit)
Insert PersistentMessageUnit Gate

During the DSP Platform initialization, the DSPDataPersistence component initializes the data processing thread DSPDataFlusher.

At the rate of TRANSIENT_DATA_FLUSHER_DELAY, the thread DspDataFlusher gets any existing transient messages and insert the data into the database.

Receiving DSP Messages in a concurrent fashion, the DSP Data Persistence leverage data writes by temporarely storing messages in transient component.

Considering that there is a matcher rule that contains the DSP Data Persistence component as the target, the message is delivered.

Figure 6.10: UML Sequence Diagram - Flow 1: From the DSP Broker to the Data Persistence Component

as a to facilitate the identification of sensor devices that are not capable of sampling their geographical location by giving their references during the bootstrap process.

When a DSP Message is delivered to the DSP Data Persistence component, the message is directly transferred to the Singleton class TransientPersistenceLayer, as described before. As described in the UML Sequence Diagram in Figure 6.12, the first point of execution is the "create PersistentMessageUnit Gate", where an instance of the SensorLocation is acquired

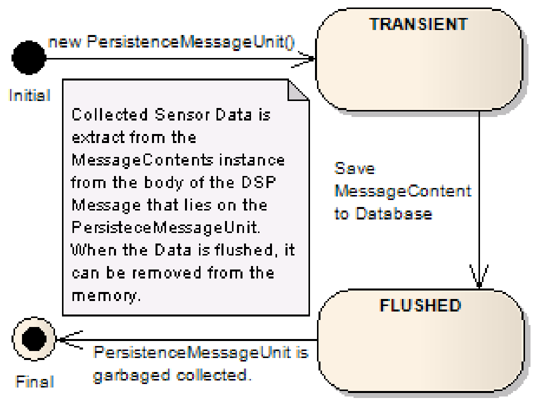

new PersistenceMessageUnit()
Initial
TRANSIENT
Collected Sensor Data is extract from the MessageContents instance from the body of the DSP Message that lies on the PersisteceMessageUnit. When the Data is flushed, it can be removed from the memory.
Save MessageContent to Database
FLUSHED
Final
PersistenceMessageUnit is garbaged collected.

Figure 6.11: UML State Diagram for the instance of the class PersistentMessageState

for any DSP Message received, in order to be added into the local cache. Before that occurs, the PersistentMessageUnit is placed in the state "TRANSIENT".

At this point, the DSP Data Persistence component maintains a set of DSP Messages in

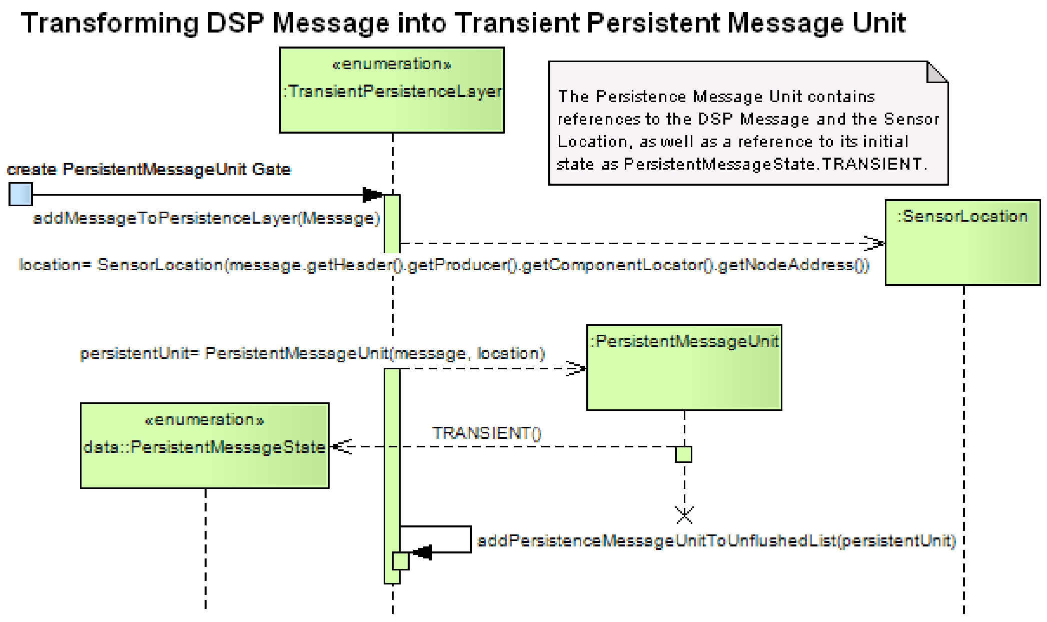

Transforming DSP Message into Transient Persistent Message Unit
«enumeration»
:TransientPersistenceLayer
The Persistence Message Unit contains references to the DSP Message and the Sensor Location, as well as a reference to its initial state as PersistentMessageState.TRANSIENT.
create PersistentMessageUnit Gate
addMessageToPersistenceLayer(Message)
:SensorLocation
location= SensorLocation(message.getHeader().getProducer().getComponentLocator().getNodeAddress())
persistentUnit= PersistentMessageUnit(message, location)
:PersistentMessageUnit
«enumeration»
data::PersistentMessageState
TRANSIENT()
addPersistenceMessageUnitToUnflushedList(persistentUnit)

Figure 6.12: UML Sequence diagram - Adding a DSP Message into the Transient Persistence layer

a local cache during the specific "flush rate" specified in the bootstrap message parameter. When the class DSPDataFlusher finds any number of instances of PersistentMessageUnit in cache, it starts the process of flushing them into the mongoDB database, described in the following section.

6.5.3 Flushing data into the Database

The class DSPDataFlusher is a thread designed to be in two different states, as depicted in Figure 6.13:

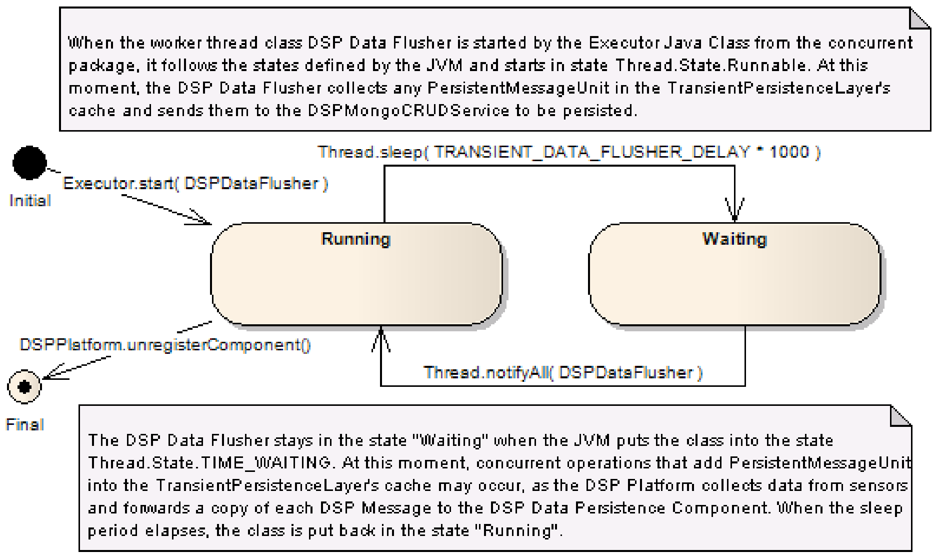

When the worker thread class DSP Data Flusher is started by the Executor Java Class from the concurrent package, it follows the states defined by the JVM and starts in state Thread.State.Runnable. At this moment, the DSP Data Flusher collects any PersistentMessageUnit in the TransientPersistenceLayer's cache and sends them to the DSPMongoCRUDService to be persisted.
Initial
Executor.start(DSPDataFlusher)
Thread.sleep(TRANSIENT_DATA_FLUSHER_DELAY * 1000)
Running
Waiting
DSPPlatform.unregisterComponent()
Thread.notifyAll(DSPDataFlusher)
Final
The DSP Data Flusher stays in the state "Waiting" when the JVM puts the class into the state Thread.State.TIME_WAITING. At this moment, concurrent operations that add PersistentMessageUnit into the TransientPersistenceLayer's cache may occur, as the DSP Platform collects data from sensors and forwards a copy of each DSP Message to the DSP Data Persistence Component. When the sleep period elapses, the class is put back in the state "Running".

Figure 6.13: UML State diagram for the DSP Data Flusher Worker Thread

- **Waiting**: the initial state of a Java thread after its initialization, waiting to be on the "runnable" state. This period of time is defined by the "flush rate" specified in the bootstrap message of the DSP Data Persistence component;

- **Running**: the class DSPDataFlusher contacts the class TransientPersistenceLayer, in order to verify if there are any instance of the class PersistentMessageUnit waiting to be transmitted from the temporary cache to the Database service. If the cache is not empty, the DSPDataFlusher acquires the lock over the TransientPersistenceLayer and retrieves the existing set of PersistentMessageUnit to be sent to mongoDB. Then, it sends all of the PersistentMessageUnit classes to the Singleton Service class DSP-MongoCRUDService, which is responsible to extract the collected data from the DSP Message's Body and convert it to the Key-Value model specified in Section 6.3.

The main participating classes from the DSPMongoCRUDService are shown in the UML Class Diagram 6.14. It keeps references to all types of sensor devices. Furthermore, it is responsible to implement the CRUD methods to modify the mongoDB database state. The class Mongo is used to acquire an instance of the connection with mongoDB server and provide Factory Methods [HK02] to create instances of the classes DB, DBCollection and BasicDBObject. The class DB is used as a reference to a database namespace, such as "netbeams", the one suggested and implemented by this work. Similarly, class DBCollection offers an API to manipulate similar documents via the abstraction of a collection of items. Finally, the class BasicDBObject is used to represent the keys and values to be persisted in the database.

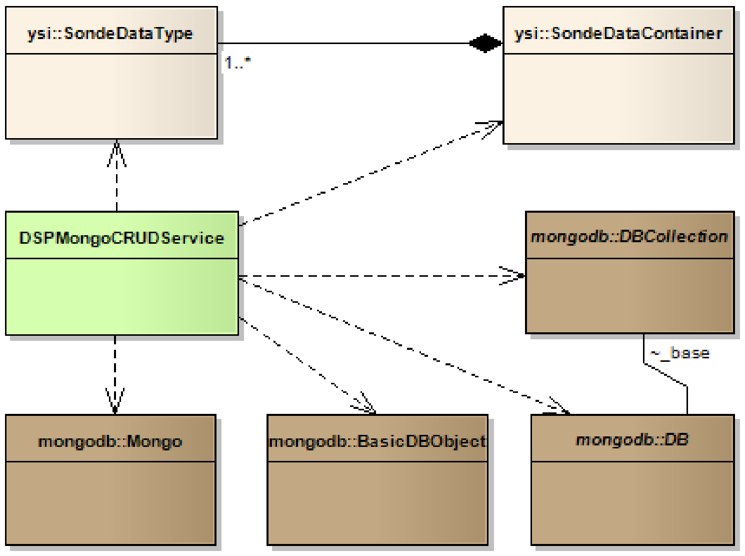


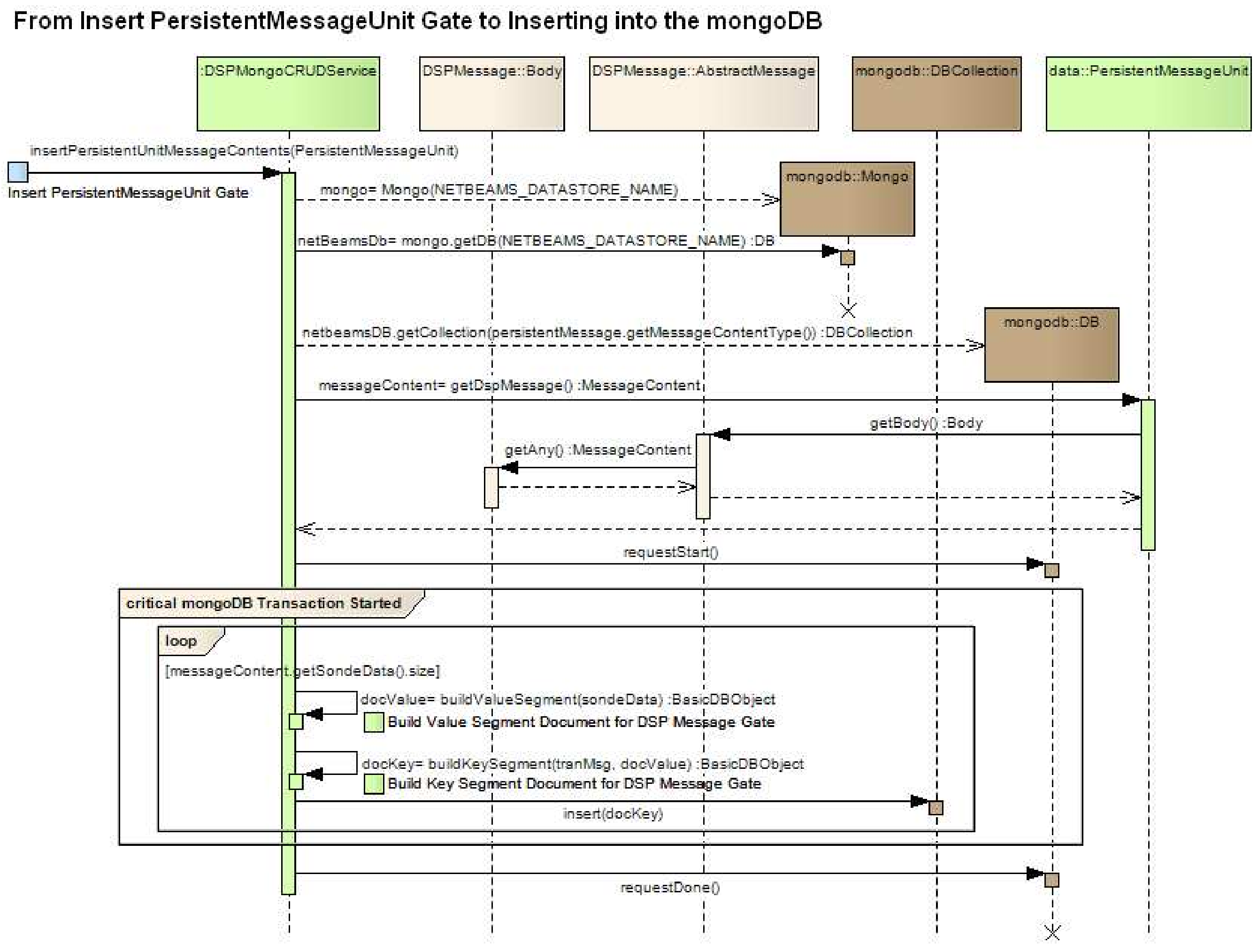
Figure 6.14: UML Class diagram with Persistence Service for MongoDB

The last method call of the UML Sequence Diagram, depicted in Figure 6.10, shows the method call to the DSPMongoCRUDService method call to persist each of the PersistentMessageUnit (see execution point “addPersistentMessageUnit Gate”). In short, the class DspDataFlusher continuously wakes up at the rate of “TRANSIENT_DATA_FLUSHER_DELAY”. Its purpose is as simple as to iterate over the list of PersitenceMessageUnit on the state of TRANSIENT, and send each of them to be saved on the persistence storage.

Finally, this sequence of events is continued by ”inserting” “PersistenceMessageUnit Gate”, which triggers the persistence service from the chosen database system as shown in Figure 6.15.

6.5.3.1 Creating mongoDB Document instance of Key-Values

In order to build the mongoDB document described in section 6.3, the reference to the DSP Message is used to extracted the metadata, while the reference to the DSP Message Content from the body of the DSP Message is used to extract the collected data from sensors.



From Insert PersistentMessageUnit Gate to Inserting into the mongoDB
:DSPMongoCRUDService
DSPMessage::Body
DSPMessage::AbstractMessage
mongodb::DBCollection
data::PersistentMessageUnit
insertPersistentUnitMessageContents(PersistentMessageUnit)
Insert PersistentMessageUnit Gate
mongo= Mongo(NETBEAMS_DATASTORE_NAME)
mongodb::Mongo
netBeamsDb= mongo.getDB(NETBEAMS_DATASTORE_NAME) :DB
netbeamsDB.getCollection(persistentMessage.getMessageContentType()) :DBCollection
mongodb::DB
messageContent= getDspMessage() :MessageContent
getBody() :Body
getAny() :MessageContent
requestStart()
critical mongoDB Transaction Started
loop
[messageContent.getSondeData().size]
docValue= buildValueSegment(sondeData) :BasicDBObject
Build Value Segment Document for DSP Message Gate
docKey= buildKeySegment(tranMsg, docValue) :BasicDBObject
Build Key Segment Document for DSP Message Gate
insert(docKey)
requestDone()

Figure 6.15: UML Sequence Diagram - MessageContent instance saved by the class mongoDB service

After the keys for the collected data are created, the keys for the metadata are captured as shown in the UML Sequence Diagram in Figure 6.17.

To summarize, the DSP Data Persistence Component is added into the DSP Platform by introducing a new component that provides service to the transfer any collected data from sensor devices to mongoDB, the Key-Value Pair database technology. The general dependencies of the system can be seen in UML Class Diagram depicted by Figure 6.18,

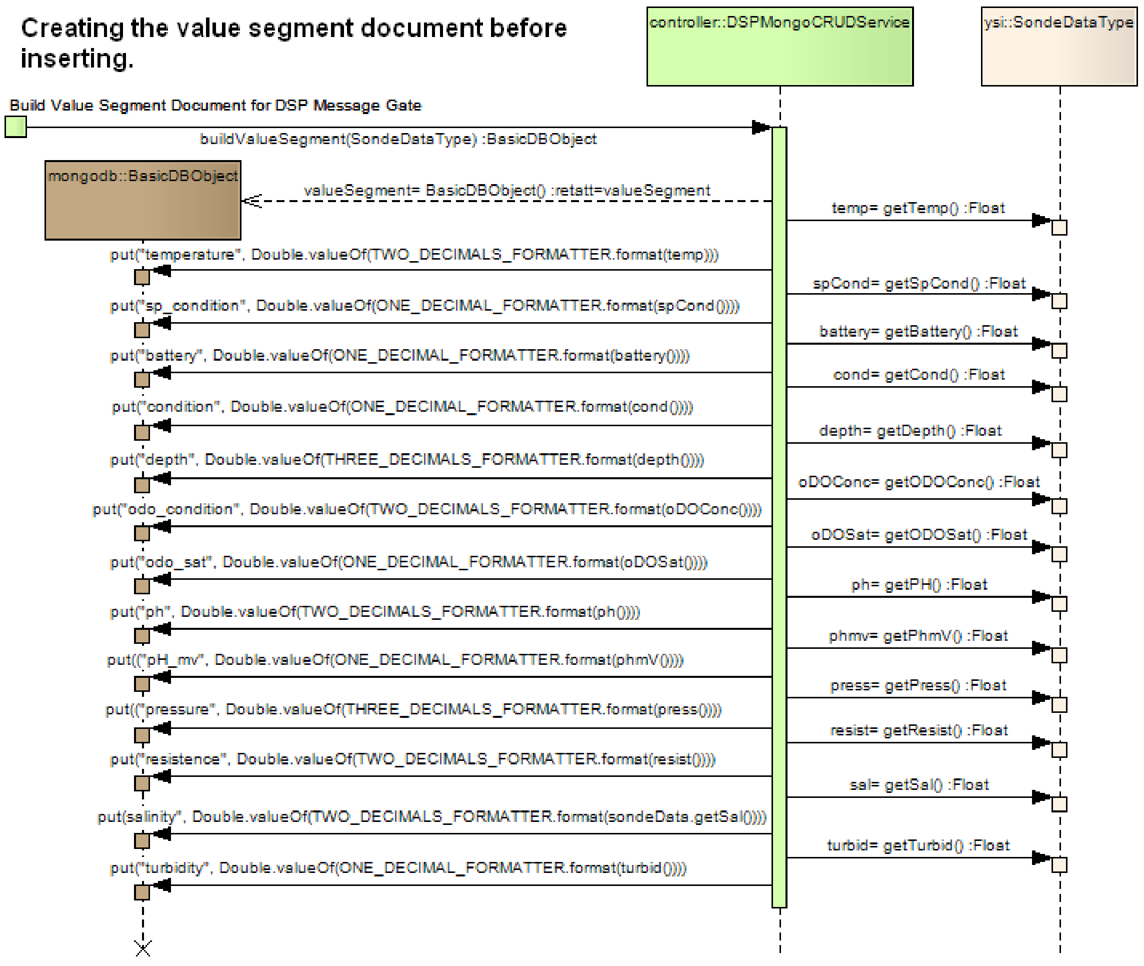

Creating the value segment document before inserting.
controller::DSPMongoCRUDService
ysi::SondeDataType
Build Value Segment Document for DSP Message Gate
buildValueSegment(SondeDataType) :BasicDBObject
mongodb::BasicDBObject
valueSegment= BasicDBObject() :retatt=valueSegment
temp= getTemp() :Float
put("temperature", Double.valueOf(TWO_DECIMALS_FORMATTER.format(temp)))
spCond= getSpCond() :Float
put("sp_condition", Double.valueOf(ONE_DECIMAL_FORMATTER.format(spCond())))
battery= getBattery() :Float
put("battery", Double.valueOf(ONE_DECIMAL_FORMATTER.format(battery())))
cond= getCond() :Float
put("condition", Double.valueOf(ONE_DECIMAL_FORMATTER.format(cond())))
depth= getDepth() :Float
put("depth", Double.valueOf(THREE_DECIMALS_FORMATTER.format(depth())))
oDOConc= getODOConc() :Float
put("odo_condition", Double.valueOf(TWO_DECIMALS_FORMATTER.format(oDOConc())))
oDOSat= getODOSat() :Float
put("odo_sat", Double.valueOf(ONE_DECIMAL_FORMATTER.format(oDOSat())))
ph= getPH() :Float
put("ph", Double.valueOf(TWO_DECIMALS_FORMATTER.format(ph())))
phmv= getPhmV() :Float
put(("pH_mv", Double.valueOf(ONE_DECIMAL_FORMATTER.format(phmV())))
press= getPress() :Float
put(("pressure", Double.valueOf(THREE_DECIMALS_FORMATTER.format(press())))
resist= getResist() :Float
put("resistence", Double.valueOf(TWO_DECIMALS_FORMATTER.format(resist())))
sal= getSal() :Float
put(salinity", Double.valueOf(TWO_DECIMALS_FORMATTER.format(sondeData.getSal())))
turbid= getTurbid() :Float
put("turbidity", Double.valueOf(ONE_DECIMAL_FORMATTER.format(turbid())))

Figure 6.16: UML Sequence diagram showing the creation of the keys for the collected data

that expands the UML Package Dependency Diagram shown in Figure 6.6.

Next section analyzes the deployment details for the DSP Platform and the mongoDB integration.

6.5.4 DSP Data Persistence Deployment

The deployment of the persistence layer is accomplished by the addition of two new components to the DSP Platform: the DSP Data Persistence component and MongoDB Database

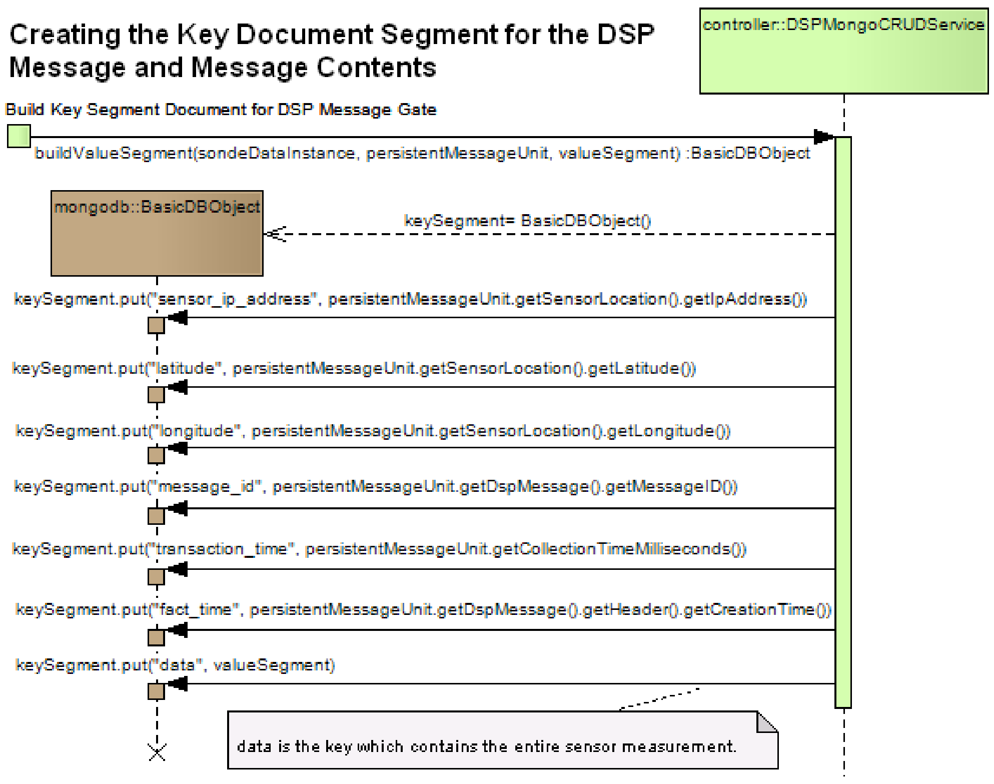

Creating the Key Document Segment for the DSP Message and Message Contents
Build Key Segment Document for DSP Message Gate
controller::DSPMongoCRUDService
buildValueSegment(sondeDataInstance, persistentMessageUnit, valueSegment) :BasicDBObject
mongodb::BasicDBObject
keySegment= BasicDBObject()
keySegment.put("sensor_ip_address", persistentMessageUnit.getSensorLocation().getIpAddress())
keySegment.put("latitude", persistentMessageUnit.getSensorLocation().getLatitude())
keySegment.put("longitude", persistentMessageUnit.getSensorLocation().getLongitude())
keySegment.put("message_id", persistentMessageUnit.getDspMessage().getMessageID())
keySegment.put("transaction_time", persistentMessageUnit.getCollectionTimeMilliseconds())
keySegment.put("fact_time", persistentMessageUnit.getDspMessage().getHeader().getCreationTime())
keySegment.put("data", valueSegment)
data is the key which contains the entire sensor measurement.

Figure 6.17: UML Sequence diagram showing the creation of the keys for the metadata

Server. The DSP Data Persistence component is responsible for capturing any data produced by sensors in the data sink, while mongoDB is where the data is going to be stored, as a result of the technology selection on the previous chapter.

The deployment of the DSP Data Persistence follows the deployment specifications of a DSP Component over an OSGi Framework. For the development of this work, the Knopflerfish OSGi Container [kno08] was used. The details of the implementation are described in Section A.1. The deployment of the mongoDB can be accomplished in different approaches, summarized as follows:

- **Single Server**: a regular deployment of a database system as it is done in general applications, where data is managed by one single process;
- **Sharded Server**: a deployment approach where data is sharded into different servers.

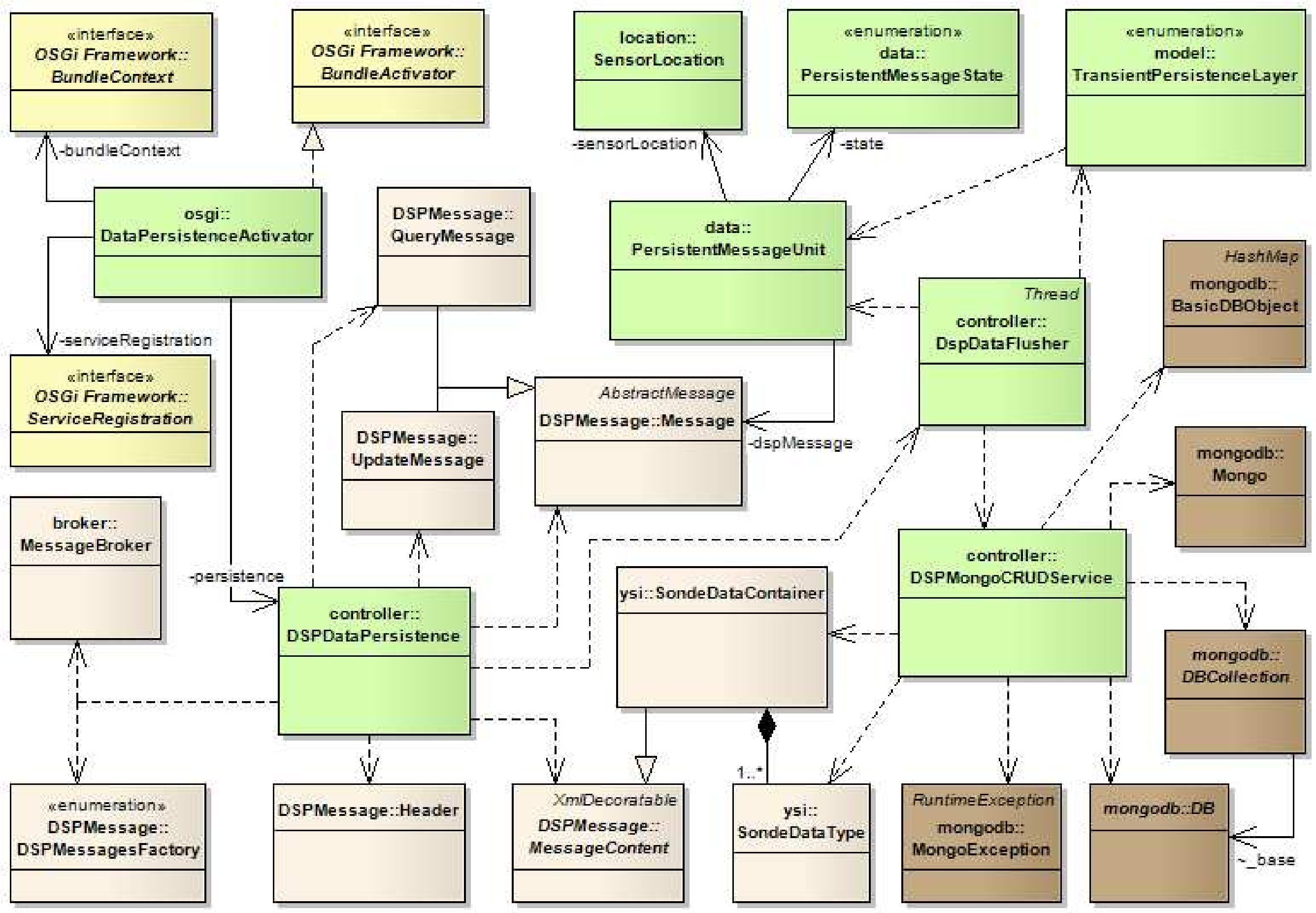

«interface» OSGi Framework:: BundleContext
«interface» OSGi Framework:: BundleActivator
location:: SensorLocation
«enumeration» data:: PersistentMessageState
«enumeration» model:: TransientPersistenceLayer
-bundleContext
-sensorLocation
-state
osgi:: DataPersistenceActivator
DSPMessage:: QueryMessage
data:: PersistentMessageUnit
HashMap mongodb:: BasicDBObject
Thread controller:: DspDataFlusher
-serviceRegistration
«interface» OSGi Framework:: ServiceRegistration
AbstractMessage DSPMessage::Message
-dspMessage
DSPMessage:: UpdateMessage
mongodb:: Mongo
broker:: MessageBroker
-persistence
controller:: DSPMongoCRUDService
ysi::SondeDataContainer
controller:: DSPDataPersistence
mongodb:: DBCollection
1..*
«enumeration» DSPMessage:: DSPMessagesFactory
DSPMessage::Header
XmlDecoratable DSPMessage:: MessageContent
ysi:: SondeDataType
RuntimeException mongodb:: MongoException
mongodb::DB
~_base

Figure 6.18: UML Class diagram for the DSP Data Persistence component.

The single server approach maintains all data located in a simple space. Distributed Systems applications such as Master-Slave, Data Replication, etc can be used in this approach for scalability 6.19.

Given that mongoDB provides support for Database Partitioning by using the concept of Shards, the implementation of a Data-Centric approach can be done by deploying different mongoDB instances in a distributed cluster, where each node can hold different collections of data. mongoDB provides Horizontal Partition with automatic distribution based on the ranges of a given key of the document. For NetBEAMS collected data, a good candidate for partitioning is the time when the data was created, and separate each shard in groups of

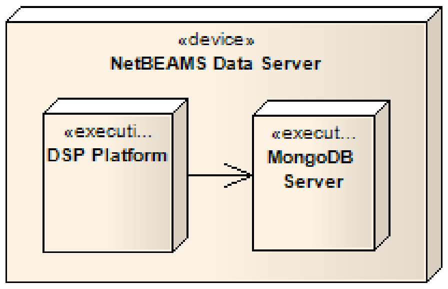


Figure 6.19: UML Deployment diagram with a single server

months of the year.

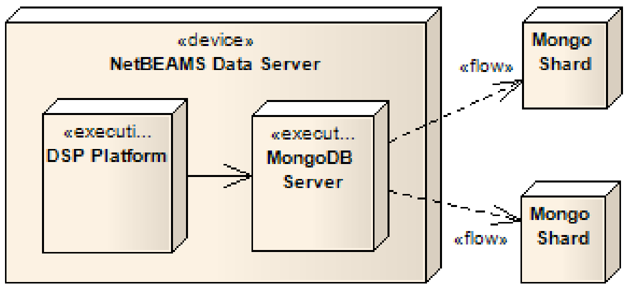


Figure 6.20: UML Deployment Diagram for a sharded server

The design of the DSP Data Persistence uses the DSP Component specification defined by the DSP Platform. First, the analysis of the business process helped to understand the requirements of the system. The data model design followed the specifications defined in Chapter 5. Each of the layers of the system was described using UML Diagrams, having partial details regarding the OSGi implementation. The next chapter entails the experiments conducted to evaluate the designs discussed in this chapter, and walks through the experiments conducted to verify the implementation of using mongoDB.

Chapter 7

Experimental Results: Correct Behavior and Performance

As discussed in the previous chapter, the implementation of a data persistence layer for NetBEAMS followed the requirements defined in Chapter 4 in regards to the classification of the SF-BEAMS sensor network and the software infrastructure provided by the Data Sensor Platform. The implementation was developed using the infrastructure provided by the online version-control system repository from NetBEAMS, whose documentation is detailed in Section A.1.

This chapter details the experiment setup designed to evaluate the proposed implementation, showing the measurement results collected from the simulation log and based on the user experience while reusing the collected data. The pertinent discussion of this work is divided up into different sections, each of them analyzing different aspects related to data persistence for NetBEAMS. Section 7.1 introduces the experiments setup and methods used for the evaluation such as the artifacts used, the definition of the workload to be used during the evaluation of the planned scenarios of evaluation. Then, Section 7.2 shows the measurement results of the experiment detailing the operations performed by each of the scenarios and user experience, to finally proceed with the final analysis and discussions.

7.1 Experiment Setup

The experiment was designed to support different scenarios of regular use of the mongoDB system infrastructure, simulating the activities of data collection from the SF-BEAMS network using the NetBEAMS infrastructure. First, the setup for the data collection of randomly generated data was implemented, which directly uses already existing NetBEAMS classes. The CRUD scenarios were implemented in different programming languages, from the data insertion scenario using the Java Programming Language, to the remaining operations using the Javascript Scripting language [jav98] and the Python Programming Language [pyt09]. Similarly, an open-source web application, developed in the Ruby Programming language [FM08], was also used. The experiment scripts were written using Shell Bash Script Language [bas01] and is responsible for orchestrating the execution of one single experimental "run".

7.1.1 Experiments Artifacts

Different artifacts were created to perform the different experiments regarding the CRUD operations. The "Create" operation' was developed in Java, isolating the services provided by the class DSPMongoCRUDService (Listing A.18, page 130). It is responsible for the connection handling with mongoDB, as well as the insertion operation from NetBEAMS. As described [dS09], the current version of NetBEAMS supports the data collection for the YSI Sonde device [YSI08], among others used for demo purposes.

In order to exercise mongoDB with the same data collected from NetBEAMS, the existing Java class TestSondeData (Listing A.27, page 151), was refactored to provide a random instance generator for the YSI Data handler implemented in the Java class SondeDataType (Listing A.26 on page 144). When the Insert function is executed, the created instances are transferred from the JVM to the mongoDB, and stored in the collection "SondeDataContainer". Specific details about the execution of mongoDB are depicted in Section A.2. Finally, the implementation of the experiment in the Java class DSPMessageToMongoDBExperiment (Listing A.28, page 154) was responsible for managing the creation and insertion of any workload composed by instances of the class SondeDataType in a bulk manner.

Similarly, the support for the Retrieve, Update and Delete operations against the inserted workload were implemented using Javascript, as shown in Listing A.24 on page 141, respectively by the function calls “find”, “update” and “remove” over the collection “Sonde-DataContainer”. Similarly, in order to implement a customizable export system that converts from mongoDB into the format OpenDAP, a simpler prototype script was developed in Python, as shown in Listing A.25 on page 142. As a result of the execution of this script, the exported artifact has the same format as described in Listing A.37, page 177.

On the other hand, the execution of the main experiment shell script requires the user to provide the size of the workload, as shown in listing A.3.

7.1.2 Used Workload

The workload selected for the experiments reflects the current use of the collected data at the SF-BEAMS infrastructure. In this way, the experiment runs for 5 different times to investigate different scenarios. This number is related to the total number of sensor devices reported to be in operation, as discussed in Section 4.1.1. The volume of the workload prepared after the experiment setup is as follows:

- **First Round**: 483,840 documents, worth of production of the YSI Sonde device during one year, at the rate of 1 observation per minute;
- **Next Rounds**: 483,840 * 4, or 2,419,200 documents, representing 5 different YSI devices similar to the first round.

7.1.3 System Environment

Two different system environments were prepared: one for the External Storage environment, or Single Server; and another for the Data-Centric Storage, or Distributed Server. Since the single server is a regular centralized database system, mongoDB just requires the execution of the database process “mongod”, which is started by specifying the directory where the inserted data will be managed. The distributed server setup requires a different list of processes (see Section A.3.3): one “mongod” process for each mongoDB shard running on the different mongoDB cluster, one “mongod” process to manage the cluster metadata, and one “mongos” process, which is the main one responsible to orchestrate the cluster. The single system environment is symmetric to the External Storage used to persist the collected data, while the clustered one simulates the Data-Centric Storage one.

The simulation of both the single and distributed system used both a Mac OS 10 Snow Leopard 64bit, with a total of 8GB of physical memory. However, in order to simulate different shards in different machines, the use of a VirtualBox [Li09] provided the use of guest operating systems to run and act as the mongoDB shards. In this way, two different versions of the Ubuntu Linux were used: one 8.04 32bits and another 9.04 64bits. The configuration used for the single server only uses the process “mongod” to be started, while the clustered setup must be enabled in the database, as in Section A.3.1.

7.1.4 Planned Scenarios

In order to verify the feasibility of the solution proposed, all CRUD operations were exercised through the implementation of the practical scenarios based on the use cases described in Section 6.2.1. In brief, the use cases were chosen to run with random values such as follows:

- (R0) Insert a volume of collected data from sensors using NetBEAMS service compatible with 1 year of observations;
- (R1) Find all documents whose observations were collected between two different dates, say the days between December 8 and 15 of 2009;
- (R2) Find all documents whose observations were collected from a given known sensor device such as one whose IP address is equals to ”192.168.0.102”;
- (R3) Find all documents whose observations contained specific values read from the environment, say Salinity equals to 0.01 and the water temperature equals to 46.47;
- (U1) Update all the documents whose observations were collected during the day of December 2nd, 2009, by adding a tag = “oil spill”;
- (D1) Remove all the documents whose observations were collected on December 7th, 2009.

- (R4) Export the produced data of the week to CSV files, using the same tabular format sequence used in the files distributed by SF-BEAMS using the the OPeNDAP format, as in Listing A.37 on page 177.

7.2 Measurement Results

As described in the previous section, the design of the experiments were based on the specifications of a data persistence for NetBEAMS, using the mongoDB database system. After setting up the experiment environment, the execution of the workload was performed against the mongoDB database using its mechanisms used to manipulate the data generated. It uses the programming language abstraction to provide access and modifiy the data collected from the NetBEAMS component. For this reason, this section details these mechanisms used to implement each of the scenarios defined in the previous section, associating numerical response time collected from the logs.

7.2.1 Insert Operation

The first set of measurements collected were the ones regarding the use case R1, shown in Listing A.18 page 130, defined as the insertion of instances of the collected data from the sensor devices using Java. The document follows the specification of the keys designed in Section 6.3. The construction of the keys and respective values were done using a Hash-like object "BasicDBObject" such as the ones shown in the method "buildKeySegment". Similarly, the creation of database indexes was performed using the same approach, as shown in the Java method "setupMetadataIndexes". Then, the insertion of the documents was performed on the service method call "insertPersistentUnitMessageContents", which delimits the bulk creation using the transaction management mechanism over the collection. In this way, different insert times were collected from the logs of the different experiments as summarized on Table 7.1.

After executing the experiment, the claimed disk space for the representation of the randomly generated data was separated into two different sizes: one related to the amount of data produced by one YSI device, and another produced by five different YSI devices, as

x	**Linux VirtualBox Server**	**Mac OS Host Server**
Single Server	25K docs/sec	51K/sec
Clustered Server	17K docs/sec	39K/sec

Table 7.1: Insertion averages on Virtual and Host in Single or Clustered mongoDB Server

summarized on Table 7.2.

x	**1 YSI**	**5 YSIs**
Indexed Keys	278.33 MB	1.35 GB

Table 7.2: Amount of disk storage used

7.2.2 Retrieve, Update and Delete Operations

The simulation of the use case scenarios for the implementation used the mongoDB shell client, which uses Javascript, for most of the operations for “retrieve”, “update” and “delete” shown in Listing A.24. The implementation of the use case to export the collected data to OPeDAP format used Python, depicted on Listing A.25. The access pattern to use the collected data is “db.collection_name.operation”, where “collection_name” is the name of the collection used for the collected data, and in this case, the name of the DSP component that represents the collected data. Therefore, mongoDB uses dynamic object binding for the collection object, and executes the CRUD operations as method calls with a list of the needed parameters. Table 7.3 relates method calls used to implement each of the use cases defined, as implemented in Listing A.24.

The execution time for each of the use cases were collected using the logs collected during the execution of the experiment script manager. The retrieval operations ran in average for 300ms, while the updates took more than 10 seconds when it involved larger data sets with more than 300,000 documents. Similarly, the delete operation took less than 13 seconds to perform.

Scenario Type	Language	Method Call	Use Case
Create	Java	db.SondeDataContainer.insert()	R0
Retrive	Javascript	db.SondeDataContainer.find()	R1, R2, R3
Update	Javascript	db.SondeDataContainer.update()	U1
Delete	Javascript	db.SondeDataContainer.remove()	D1
Export	Python	db.SondeDataContainer.find()	R4

Table 7.3: mongoDB method calls used for the use cases

7.2.3 User Experience

In general, the user experience of mongoDB is related to the methods used to access and modify the state of the database using a different approach when compared to traditional Relational databases. Since most of the methods to create, access and modify the database were performed using the programming language abstraction of objects and method calls, the transition from the use of a relational data model is very easy. The creation of data was the first contact with mongoDB in terms of data creation in Java. Later, different methods of data access were used after the randomly generated data was inserted into mongoDB: using the shell commands mongo and “mongoexport”, command-line execution developed in Python, through the REST Web Services API, and using a web application, written in the Ruby Programming Language [FM08], providing the view over the collected data using a web browser. More details about the execution of these commands during the experiments are shown in Section A.3.3.

The primary method used to access the mongoDB database is using the shell command. Once the user requests access to the database, he or she can issue an operation over the collection of documents using regular the abstraction programming language, as the shown in Listing A.24 on page 141. The CRUD operations are available to the user and can be used at any time to any database collection. Aggregation operations such as to count the returned result set is also provided via programming languages abstraction.

In order to export the collected data to other data formats, the command “mongoexport” and the developed export script written in Python were used. The “mongoexport”, exports the collected data from the database using the native support to comma-delimited values.

As shown in Listing A.38, page 177, the actual execution describe how mongo is querying the data from the database. As a result, the total number of documents exported is shown in on line 8. In addition to the regular command that exports the entire database, the user can provide a query string during the execution of the command in order to filter the dataset returned and, consequently, the number of documents exported. Similar to the use of the default export capabilities of mongoDB, a script written in Python to export data from mongoDB was written as shown in Listing A.25, page 142. It uses the package "pymongo" to create a comma-delimited file with the same header and line format as shown in Listing A.35, page 176. The results can be compared with the example the output of the HTTP Response body of Listing A.37 on page 177, requested directly to the OPeNDAP server of RTC. Therefore, the experiment is capable of reproducing the targeted OPeNDAP format used by the RTC server.

While the access to the database using the shell command gives full access to the CRUD operations, the access to the data was also performed using the REST Web Services API, as shown in Listing A.8 on page 113. As one can see, the IP address and the default port number of the mongoDB REST middleware are provided. Then, the REST operation uses the name of the database and the collection to correlate with the ones in the database system. Thus, querying the collections from mongoDB can be performed using the HTTP Request Parameter variables, by adding the prefix "filter_" before the name of the key of the document. The given example shows the query performed in the collected data for the key "observation.Conductivity" with the value "104.5". Similarly, the HTTP Request variable "limit" is used to reduce the number of documents returned by the query. The HTTP Response from mongoDB returns the result using the JSON format, similar to the ones printed in the command-shell.

Finally, the access of the collected data was performed using Futon4Mongo, an open-source web-based application written in Ruby that provides access to the mongoDB database collections and documents. The screenshots of the session used to browse through the collection of collected data is shown in Figure A.5.

7.3 Discussion

The implementation of the data persistence for NetBEAMS follows the specifications explained in Chapter 4. The purpose of the sensor network for NetBEAMS is to support data archival from any collected data, and for this reason, mongoDB was selected after an empirical analysis described in Chapter 5. The requirements described in Chapter 4 were expressed as examples of use cases in Section 7.1.4. This chapter discusses the experiments from different angles: analysis of the taxonomies, the infrastructure, the use cases evaluation and applicability in other projects.

7.3.1 Taxonomic Evaluation

After selecting mongoDB as the best candidate for the persistence infrastructure for NetBEAMS, the development of the persistence layer met the requirements of archiving data for different types of sensor devices from the DSP Platform such as the YSI Data Handler. As it is shown in Section 7.2, the collected data is archived in files using the BSON format [bso09], a binary representation used to store instances of the document described in Listing A.36 on page 176. Along with the purpose of Data Archival, the use of mongoDB also met the specification of External Storage, since the data persisted in the file system, as well as Data-Centric for running mongoDB with different distributed mongo shards (cluster nodes).

SF-BEAMS uses Tabular Data model, a schema-less data model that represents the collected data using comma-delimited files, without a database system to manage those files, just the OPeNDAP middleware. In contrast, the data persistence method proposed by this work suggests mongoDB as a schema-less database system to manage the data collected using NetBEAMS. As described in Chapter 5, the schema-less data model can provide better support to Data Provenance because the data model allows the change of records without affecting the others. In this way, the metadata used by the implementation clearly describes the three fundamental questions regarding the collected data from sensor devices, as described in Section 6.3. The identity of the data is given by the key “message_id”, in a way that it uniquely identifies the data produced. Thus, the message can be tracked back to the sensor that produced the data for different purposes such as tracking the data producer. Then, time dimensions were used in order to track two different aspects of the data collection: the valid time can be used to correlate observations to the exact moment it occurred, while the transaction time can be used to verify when the data was collected, and make decisions such as deleting the instances of data collected in the previous two days, etc. Similarly, the use of the key "sensor.location" can give the exact location of the data, since researchers can be interested in finding data based on a specific location. The most important aspect of the schema-less data model is that the observed data is added to the key "observation", as any sensor device carries its set of attributes of key-value pairs. A good example of the applicability of the schema-less model is the scalability regarding changes to the data structure. For example, if an existing sensor device is flushed with a new version of the firmware[1], changing the format of data types or adding new values, the following instances of collected data may very from the previous one by simply adding the new additions of data. For this reason, the DSP Data Persistence gives administrators the ability to choose which fields to persist by declaring it using a bootstrap message for the component, as shown in the parameter "YSI_DESIRED_PROPERTIES" of Listing A.15, page 122.

In order to access the collected data, the implementation used different APIs provided by the mongoDB community. The Centralized Query Mechanism was satisfied by the use of those APIs, which also fulfills the requirements defined by NetBEAMS. For example, the accesses of the database using Python or Java were based on the needs and the use of the languages for different purposes. In this way, the implementation of the persistence model captures the instances of any known data collected from NetBEAMS and saves it using the document pattern as shown in Listing A.36 on page 176. Any programming language used was capable of accessing the same collected data by their own.

In the end, the implementation of the data persistence for NetBEAMS was developed in both Centralized and Distributed Systems. The use of regular single server execution provided enough computational power for a compatible volume of data when compared to the one produced from the SF-BEAMS sensor network. The implementation suggested by this work fulfills the needs of a persistence system for a Small volume of data being created at the same time in a Single Server. Furthermore, mongoDB also provided support for a Distributed System, when the use of database shards, in forms of different mongoDB nodes,

[1]Primary machine instructions that run in the hardware

was used as a clustered environment.

7.3.2 Infrastructure Evaluation

Overall, the implementation of the DSP Data Persistence component meets the specification of the use cases defined in Section 7.1.4. From the perspective of disk-space utilization for the compatible volume of data for five YSI devices is that it requires a small amount of disk space. According to the mongoDB documentation [mon09], the claimed amount of disk space is related to how mongoDB uses a binary representation of the JSON documents, which are composed only of strings, together with the definition of the indexes. The size of the keys defined for the YSI Sonde data suggests additional disk space was used for the storage.

The performance to insert data into the database system using the Java driver is documented by mongoDB as the fastest approach, providing persistence for thousands of documents per second. This capacity is more than adequate for NetBEAMS, which can use the Single System environment. Improvements over the response time can be achieved through the implementation of a Data-Centric approach on a Distributed Database environment, or clusters. In this approach, the inserted data is stored in different servers based on a partition key. Given that less data is concentrated in each partition, the centralized query mechanism used is faster, and consequently improves the performance as described in the Data Location taxonomy in Chapter 3. In this way, the infrastructural performance during the inserts suggests that persisting data collected from the NetBEAMS sensors using a clustered environment is an option when the single server capacity is reached.

Different problems and limitations were faced during the execution of mongoDB, which were related to the infrastructure used: the limitations regarding disk storage in 32bit operating systems and very slow performance when using virtualized servers. During the execution of the experiments, mongoDB used memory accordingly, without taking control of the operating system's resources. However, when mongoDB is used in a 32bit server architecture, there are limitations on the disk space used to store collections. After reaching 2GB of collected data during many different runs, mongoDB crashed and did not allow the use of the collected data. For this reason, the setup of a new VirtualBox using Ubuntu Linux 64bits as a guest to a Mac OS X 64bits was prepared and used throughout the completion of these experiments. On the other hand, the performance was drastically decreased using mongoDB in virtualized servers due to the fact of dynamic partition chosen. The performance shell application "iostat" was used to verify the behavior, and the use of mongoDB was switched to the host operating system, whose hardware specifications are summarized on Table 7.3.2.

Table 7.4: Experiment Host Machine Hardware Specifications

Model Name	MacBook Pro
Model Identifier	MacBookPro5,3
Processor Name	Intel Core 2 Duo
Processor Speed	2.66 GHz
Number Of Processors	1
Total Number Of Cores	2
L2 Cache	3 MB
Memory	8 GB
Bus Speed	1.07 GHz

7.3.3 Use Cases Evaluation

[BDD09] suggested that programming languages are easier abstractions to researchers that use collected data from sensor devices without a background in computer science or database systems. The authors have suggested the implementation of a programming language for this group of biologists, for instance. This suggestion has good intentions, but limits the users with a single mechanism to access the collected data from sensors. For this reason, this report proposes the use of key-value pair databases, such as mongoDB, to provide transparent access the data through the abstraction of any programming language that has a driver written for mongoDB. During the user experience section, different programming languages were used to access the collected data, a novel mechanism not yet reported in the sensor networks community.

The development of the scenarios were clearly written in different ways, such as the ones defined in Listing A.24 on page 141, which implements the use cases for data Retrieval, Update and Deletion operations. The main advantage of this approach is that users do not use the SQL language to extract data from the database, but use method calls defined for each the abovementioned operations. For example, based on the description of the collected data by keys and values shown in Listing A.36, page 176, different use cases can evaluate documents in this structure. Usually researchers may search the collected data by a given range of dates, for their regular activities of data analysis. The scenario implemented by R1 in the above cited listing shows it clearly. Similarly, the search for data in a given sensor device is expressed by the search for the IP address for them on the use cased defined by R2. Finally, the search for the raw data obtained by the sensor devices is used from the key "observation", as shown by the R3 use case implementation.

Allowing for non-predictable events that might occur during the data collection process of sensor devices, which will influence their values, the support to annotations is a feature that can be used to describe the data. mongoDB indirectly supports the use of annotation because of its method of updating documents and adding new keys to any of them without requiring design changes. Supposing an environmental event occurs, such as the recent oil spill in the San Francisco bay in October 2009 [sfb09], the values of the collected data, and data analysis, was influenced by different presence of different objects in the water. The implementation of the scenario U1 provided the addition of a new key called "tag", with the annotation "oil spill". This new key, after properly indexed, provides another option to meaningfully search the collected data from the sensor devices. Old data then can be deleted as expressed by the scenario D1, as a regular operation supported by any database system.

Commonly during scientific research, data is shared. Researchers can choose the use of spreadsheets to analyze collected data with simpler tools than database systems or programming language. For this reason, OPeNDAP is used in the community to share data using a comma-delimited format. In this way, the implementation of the use cases for R4 was developed in two different ways. First, the use of the export capabilities to a common-delimited values was performed (Listing A.38, page 177), taking proximally 3.3 minutes to export one year's worth of data for 5 YSI devices without indexing. Second, the implementation of a custom export utility using Python, as shown in Listing A.25, page 142, enabled the creation of the comma-delimited files with the same format as the ones used by OPeNDAP. The comparison can be done by analyzing the output of the OPeNDAP request from SF-BEAMS on Listing A.37, page 177, and the output provided from the script written in Python, show in

Listing A.35 page 176. Furthermore, considering research teams with skills in Web Services, an easy ubiquitous integration with mongoDB can be achieved by using the REST Web Services APIs provided by mongoDB, still in alpha version. A practical application of this API was the created Apache Futon4mongoDB web application, used to give user friendlier access to the collected data as shown in Figure A.5, on page 115.

As discussed in Chapter 5, the differences between a traditional database system, which uses the relational data model, and document-oriented database systems were clearly noticed during the development of this work. As mentioned in Section 7.2.3, the user experience was mostly related to the definition of the keys that describe the sensor device, developed as discussed during the design of the solution in Chapter 6. Therefore, based on the use of a different type of data model in the scope of data persistence for sensor networks, the taxonomies defined in Chapter 3 could be updated to the one shown in Figure 7.1, including the Key-Value Pair Data Model.

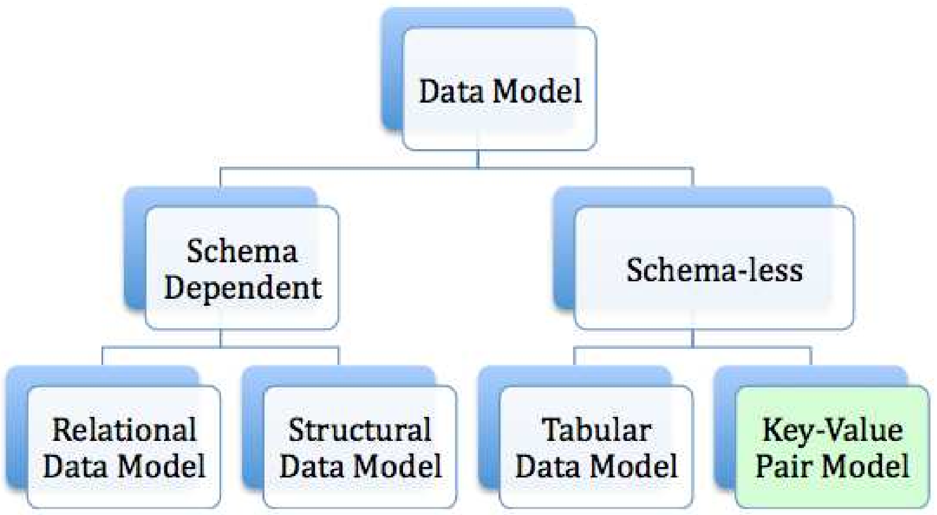


Figure 7.1: Updated Data Model Taxonomy, adding the KVP Data Model

7.3.4 Applicability and Recommendations

This work can be used to evaluate data persistence in sensor networks in different ways. The use of the data persistence taxonomies through this work guided both the evaluation and implementation of the data model using mongoDB. In order to use the guidelines proposed by this work, one may first determine the properties of the sensor network in question. Considering that sensor devices produce streams of non-described data, the use of key-value pairs data model did not require changes throughout different application layers.

Focusing on the different layers of an application, consider the different multi-tier application approaches where each layer implements a functionality of the system such as the Model-View-Controler (MVC) [KBK+05]. Usually, the use of traditional database systems and object-programming languages require "data plumbing" from the table relations from the database system to the object instances. In order to implement all the CRUD operations to a relational database without an ORM [orm02] offering, the implementation of the data model potentially uses the Structured-Query Language (SQL) for the problem of data plumbing, which results in a tightly-coupled implementation. On the other hand, mongoDB eliminates the use of an additional query language because it uses declarative approach on top of the programming language drivers it support, decreasing the complexity of the application.

Sensor networks with medium or large volumes of data are implemented nowadays using clusters and parallel programming. Google's BigTable [CDG+08] uses the principle of structured data in distributed tabular format, processing large volumes of data using the MapReduce parallel approach [DG04]. The current version of mongoDB offers its first version of the MapReduce approach, but it is still imature. However, it can be useful to process large amount of data collected from sensor devices, or process historical data for any type of analysis.

7.3.5 Difficulties Experienced

Particularly speaking about the learning curve, different situations contributed to the success of the first experiments. The first problem to be solved was the selection of the data model to be used during the technology selection. Since relational database systems are the "traditional" way to provide data persistence, the investigations started by creating a relational model described in Chapter 5. Then, the technique using KVP data models in relational databases were attempted without success. Only after reviewing the technical article [Bai09], suggested the advisor of this work, the search for a database system that uses the KVP data model was considered, and finally successfully implemented.

Since the technology is new, it brought new challenges regarding the usability and user experience. The technology is relatively new and still under development, many different bugs were be reproduced at random during the execution of the experiments, and therefore, reports were sent to the open-source community through various communication channels. To address this, the fastest way to solve an issue was through the mongoDB's IRC channel located at the Freenode server (www.freenode.org, channel mongodb). Second, the users group, located at http://groups.google.com/group/mongodb-user also helped troubleshooting problems.

Physical memory is limited and, for this reason, it may not be enough to run the experiments for total workloads. In this case, the Java Virtual Memory was totally consumed by one of the revisions of the experiments setup. Therefore, it was learned that large amount of data loads require different techniques to manage data in memory accordingly, such as creating different object pools as temporary caches for objects whose internal state are similar to others in the JVM. Also, operations that involve writing in disk, such as during the bulk "Create" operations, may lead to different database performance. First, the experiment seemed to stop inserting objects in one shard while it ran for over 2 minutes without inserting any documents. The reason was that the partition was "/home", completely full, leading to the problem. The mongos cluster header did not realize that it was the time to switch to another shard. Similarly, any global "Retreave" operation on a cluster blocks all the shards. For this reason, dividing the write from the read load was necessary in order to decouple these two operation.

Also related to the difficulties of this work, mongoDB is not yet stable, and therefore, harder to implement. The main production version of mongoDB is on version 1.2 at this time, while the shards support is still in alpha version. For instance, different work-arounds were performed to have shards experimented as described, but the execution was not successful.

To sum up, this chapter described the experimental results of the implementation of a data persistence component for NetBEAMS, showing the experiment results, the measurement results and finally the discussion about the solution in general. The following chapter brings the conclusions and future works for this report.

Chapter 8

Conclusions and Future Works

There were significant trends covered in this work, including the importance that sensor networks are becoming ubiquitous, having their produced data available online and other types of media. In the scope of Environmental Sensor Networks, the collected data is the most important parameter to real-world applications based on sensor networks. For example, the data collected in the water may determine if a river is polluted, while the data collected during a heavy snowfall can indicate hazard alerts to the population. However, in order to provide access to the collected data, one must first assess the different types of infrastructural characteristics of the studied sensor networks. The selection of a technology that provides such a persistence solution capabilities can be part of a process of analysis of taxonomies proposed by this work, characterized as the first contribution of this report.

Based on the findings of the taxonomies, together with the empirical analysis of the technologies, this work revealed a novel approach to provide persistence and access to collected data from sensor devices using the Key-Value-Pair data model. Besides the contribution of a new component for the Data Sensor Platform, the evidence that the system improves productivity was confirmed during the execution of the experiments, writing less source-code to obtain better functionalities when compared to other solutions. Similarly, this work suggests that the abstraction of programming language should be used with research groups whose primary expertise is not in Computer Science or Database Systems.

The DSP Data Persistence component improves NetBEAMS's support to SF-BEAMS, incorporating a persistence layer for the sensor network that can be used in different ways. The solution for data persistence using Key-Value Pair databases is a novel way to save collected data from sensor devices into a single or multiple servers of a database technology. mongoDB's ability to cope with the uncertainty of any properties collected from any sensor specification is the most compelling reason to use the Key-value pair data model for collected data from sensor devices.

As far as future works, there are many different directions for exploration. The first major release of mongoDB with complete support to shards is scheduled for 2010. An important mechanism of mongoDB is that it can manage different mongoDB servers spread out in a cluster. This feature will give better support to the Data-Centric Storage approach. This approach can reduce the retrieval time by focusing on centralized selection of data over a given database shard. Moreover, the overall search through the collected data attributes can be drastically decreased by a parallel search over the database shards using the MapReduce [DG04] tools. At the time of development of the experiments, the MapReduce API from mongoDB was also in the alpha version, and therefore, presented many bugs. One suggestion for the implementation of such functionality is based in Figure 8.1 ([Buc08]), which shows an architectural idea of the deployment of a Data-Centric storage approach. One scenario of data retrieval is to process search results required to be executed in parallel. The task of querying broken down into two phases: the definition of the reduced problem as the parts, and the map of the final result calculated with the response from each part. If "Task" is considered to be a query (arrow 1) to the server, then the execution of the same query occurs in all different partitions (arrow 2), or mongoDB shards. After processing the request, each node returns the result of the query to the server (arrow 3), whose responsibility is to process each individual result and return the final one to the client application (arrow 4).

In addition to the infrastructure, the use scheduling techniques for data gathering in sensor networks can help the SF-BEAMS sensor networks decrease the data load into the system. While the period of times used by RTC seems to cover the basic requirements of data, the data storage may be seen as overloaded with repeated data. Since environmental conditions might not change drastically over time, the use of data scheduling and data clustering approaches can be used. Considering that SF-BEAMS contains a cluster of sensors, the data load could be decreased by using techniques such as clustering the data in the data

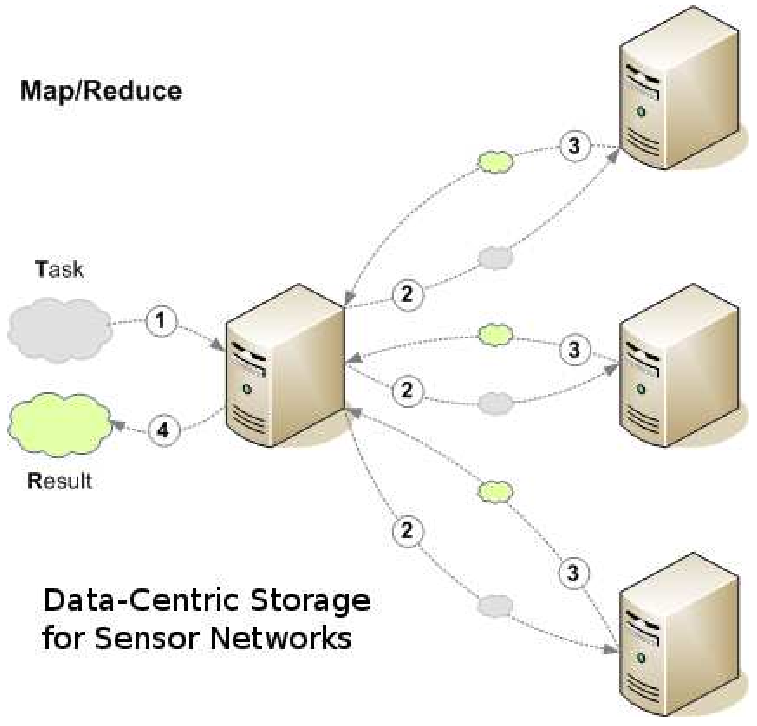


Figure 8.1: Data-Centric approach using MapReduce ([Buc08])

sink before they are persisted [YG08, CHZ09]. Since the infrastructure of the SF-BEAMS sensor network is a one-hop start design, the knowledge about the data can only be achieved in the data sink. One suggestion is the implementation of a DSP Data Clustering component that is responsible for the data clustering before they are inserted into the database. Similarly, the use of schedulers on the sensors could also be used to decrease the amount of data to be sent to the data sink, as seen at the survey [WX06].

Finally, taking into account the use of the database system to complement NetBEAMS execution, the design of event-based applications could add different functionalities to the management capabilities of the sensor networks. For instance, the collected data from a YSI device [YSI08] carries the information regarding the battery life of the device, seen at the collected attribute "observation.Battery". A new DSP Component responsible for observing these attributes could define the threshold of the event of low-battery. With an event-based application in mind, events regarding the SF-BEAMS network could be better managed through NetBEAMS where scheduled sites visits would depend on such events to happen and, thus, avoiding unnecessary operational costs.

Appendix A

Appendix: Implementation and Experiments Artifacts

This chapter includes references to the implementation Data Persistence Implementation covering its design and architecture. Also, some of the implementation details of the DSP Data Persistence component designed in Chapter 6 are presented, showing the generated experiment artifacts. While this section does not include all the related source-codes, the remainder can be downloaded directly from the Subversion repository branch at http://code.google.com/p/netbean

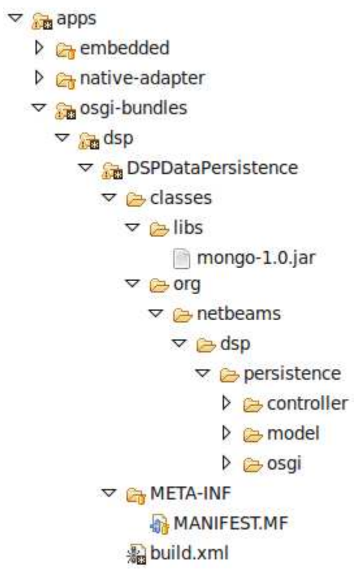
A.1 DSP Platform Deployment Details

The DSP Data Persistence component was implemented using the Java Programming language, on top of the OSGi Framework. This section describes the setup process of the implementation, as well as the artifacts use during the implementation, which will be listed in the appendix section.

The structure of the artifacts implemented for the DSP Data Persistence Component follow the conventions of the NetBEAMS implementation [dS09], where the main directory **PERSISTENCE-DIR**, depicted by Figure A.1, holds each of the development artifacts.

- **DSPDataPersistence**: Main directory with the build.xml (Listing A.10) and other Eclipse-related artifacts. The other major directories are also listed;

- **META-INF**: this directory contains the descriptor file for the OSGi Framework, in this case, the MANIFEST.MF;
- **src**: the main directory structure for the source-code implemented. Note that it follows the Java specification for packaging, and therefore, lists the package org.netbeams.dsp.persistence as the main package, including the layers controller, model and OSGi.



```
apps
  embedded
  native-adapter
  osgi-bundles
    dsp
      DSPDataPersistence
        classes
          libs
            mongo-1.0.jar
          org
            netbeams
              dsp
                persistence
                  controller
                  model
                  osgi
        META-INF
          MANIFEST.MF
        build.xml
```

Figure A.1: The DSP Data Persistence Directory Structure

A.1.1 OSGi Deployment Process

Since each NetBEAMS component is managed by an OSGi component and its infrastructure, this section describes the basic functionality of the OSGi platform [osg08], a framework was conceived to support modularity in terms resources-limited environments such mobile devices and vehicles, but it was first widely deployed on Eclipse IDE[1]. It promotes reusable loosely-coupled modules to be integrated using the Java Programming Language [AGH05]. The most basic layers of the OSGi Framework depicted in Figure A.2 are as follows:

[1]Integrated Development Environment

- **Module Layer**: manages the OSGi bundles deployed on the OSGi Platform, providing the necessary "wiring" of the components. In other words, the modules, called OSGi Bundles, can export and/or import packages in the level of a Java Class managed by the OSGi Framework;
- **Service Layer**: responsible for the interoperability between two or more bundles, enabling the bundles to register services offered by the its specification;
- **Execution Layer**: executes the bundles, managing and changing their lifecycle through the bundle execution.

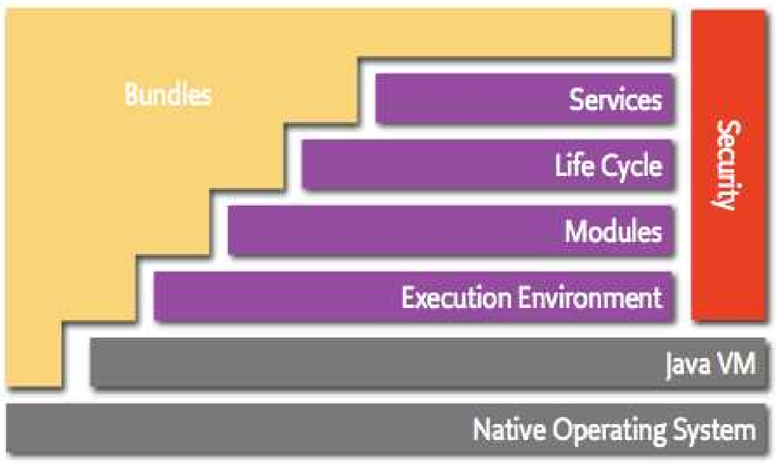


Figure A.2: The OSGi Framework Layers

NetBEAMS uses the concept of the modularization through the use of the Producer-Consumer paradigm described in the previous section. Since the DSP Components are essentially OSGi bundles, the interoperability between them is given by the OSGi Framework, described by an artifact called MANIFEST.MF (Listing A.11 on page 119). In general, an OSGi bundle must provide specifications that describe the module to be published into the OSGi Framework. In this way, the main properties of the OSGi MANIFEST.MF artifact can used by the DSP Data Persistence can be summarized as follows:

- **Bundle-Activator**: The name of the instance of an OSGi Activator class, responsible to manage the bundle. The implemented class “DataPersistenceActivator” provides the activation mechanisms for the services of the DSP Data Persistence Component;
- **Bundle-ClassPath**: the necessary Java Jars list needed to run the bundle. It lists the mongoDB Java driver as one of the required dependencies deployed in the same package;
- **Import-Package**: the Java Packages needed by the DSP Data Persistence Component Bundle. These Java Packages must be provided by other DSP components, forcing the execution to be dependent on those packages. They are provided through a section called “Export-Package”, in other OSGi bundles and as it is shown in Listing A.11, page 119, they are can be from different packages.

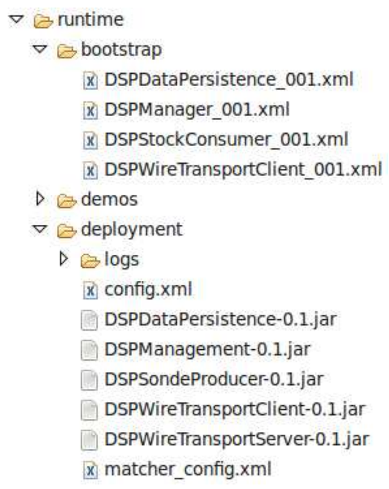
Once the OSGi bundle is installed into the OSGi Platform, it will be managed by the OSGi Execution layer and change the bundle state according to its lifecycle. Therefore, the packaging of the OSGi source-code is done using a JAR[2] artifact specification [CWH00]. The DSP Platform is bundled into a JAR created by the Apache ANT build script [Moo05], as shown in Listing A.10. The task “dsp-data-persistence.all” compiles all the source-code created from the design of Chapter 6 and packages everything into an artifact name “DSPDataPersistence-x.x.jar”, where x.x is the version of the bundle. Figure A.3 shows the DSP Data Persistence file under the directory “**RUNTIME-DIR**/deployment/”.

A.1.2 DSP Data Persistence into the DSP Platform

When the new component was developed and deployed by editing two main configuration descriptors:

- **config.xml**: the DSP Data Persistence is added to the DSP Platform by adding an entry for the component’s OSGi Bundle artifact. This artifact can be seen in Listing A.12, page 119;

[2]Java Archival Repository



```
runtime
  bootstrap
    DSPDataPersistence_001.xml
    DSPManager_001.xml
    DSPStockConsumer_001.xml
    DSPWireTransportClient_001.xml
  demos
  deployment
    logs
    config.xml
    DSPDataPersistence-0.1.jar
    DSPManagement-0.1.jar
    DSPSondeProducer-0.1.jar
    DSPWireTransportClient-0.1.jar
    DSPWireTransportServer-0.1.jar
    matcher_config.xml
```

Figure A.3: The DSP Runtime Dir

- **matcher_config.xml**: the addition of the rules that filters DSP Messages to the DSP Data Persistence component;
- **DSPDataPersistence_001.xml**: depicts the bootstrap message for the DSP Data Persistence Component, as shown in Listing A.15.

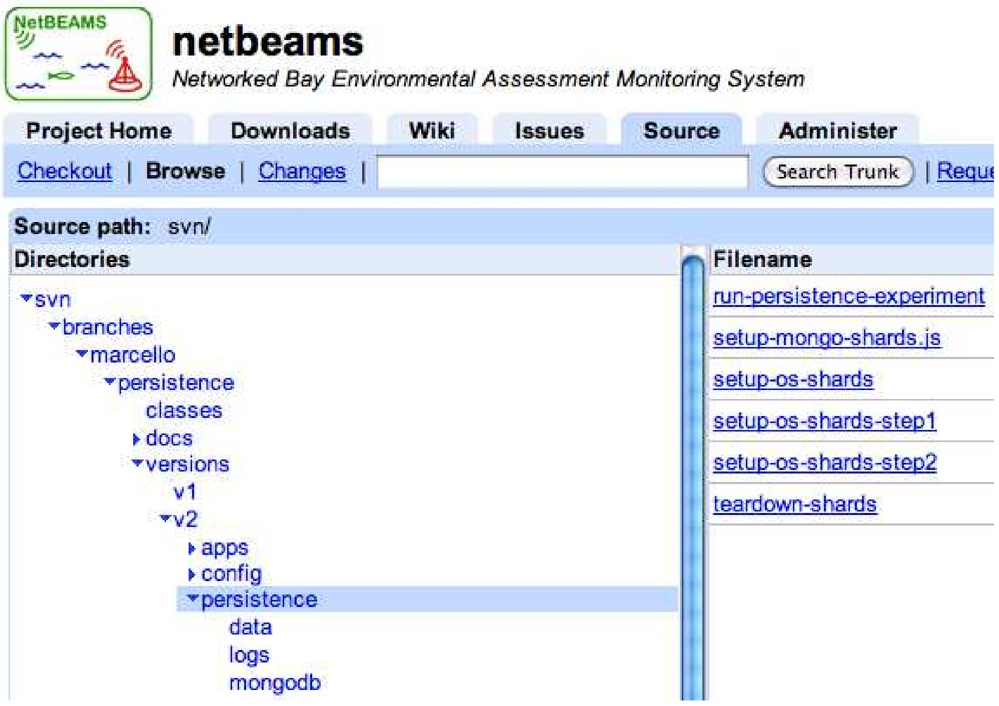
The first configuration artifact enables NetBEAMS to initialize the descriptor of the component, as shown in Listing A.12, page 119. The component is added with the highest priority, since it contains dependencies to other DSP Components.

A.2 mongoDB Deployment Details

MongoDB was the selected technology described in Chapter 5 because of its features. It provides data persistence based on collections of data, stored using BSON [bso09], a binary representation the JSON data representation format. mongoDB uses the abstraction of programming language to represent the basic CRUD operators, as well as the regular functionalities provided by any type of database system such as indexing and users management. As it's stated in their web site, mongoDB "bridges the gap between key/value stores, which are fast and highly scalable, traditional RDBMS systems (deep in functionality)".

- mongoDB implements a document-oriented structure, which is similar to Key-Value Pairs;
- mongoDB is written in C++, and therefore, can is available in any major platform, as well as offers a broad range of API drivers written in different languages such as Java, Python, Perl and Ruby;
- mongoDB has support to distributed systems such as Master-Slave Replication, and horizontal database partitioning, referred to as Database Shards.

The artifacts of the mongoDB are located in the “thirdparty” directory of the NetBEAMS resources directory. The build script from the DSP Data Persistence component can be executed to produce the persistence directory structure for NetBEAMS, as described in Figure A.4.


NetBEAMS
netbeams
Networked Bay Environmental Assessment Monitoring System
Project Home
Downloads
Wiki
Issues
Source
Administer
Checkout | Browse | Changes |
Search Trunk
| Reque

Source path: svn/

```
Directories
svn
  branches
    marcello
      persistence
        classes
        docs
        versions
          v1
          v2
            apps
            config
            persistence
              data
              logs
              mongodb
```

Filename
run-persistence-experiment
setup-mongo-shards.js
setup-os-shards
setup-os-shards-step1
setup-os-shards-step2
teardown-shards

Figure A.4: DSP Data Persistence directory

A.2.1 Using the Client and Server

In order to start the server, the command-line shown in Listing A.1 is used, where the data is configured to be stored at the directory "./data/", relative to the directory shown in the shell. Also, the port number where the server listens to the client connections is give as port "20000". Once the server is running, any client application can access to the server. For example, in order to connect the DSP Data Persistence component, the port 20000 is an optional value be provided in the DSP Data Persistence bootstrap message, as shown in Listing A.15. Although mongoDB offers the option to run with its specific default port number, it is recommended to assign different ports, once the use of shards requires different port numbers for each of the shard servers. An example of accessing mongoDB using a shell client is shown in Listing A.2 page 109.

Listing A.1: Starting the Server

```
marcello-de-saless-macbook-pro:persistence marcello$ mongod --dbpath data/ --port 20000 &

Wed Dec  9 01:39:51 Mongo DB : starting : pid = 10096 port = 20000 dbpath = data/ master = 0 slave = 0
    64-bit
Wed Dec  9 01:39:51 db version v1.1.4, pdfile version 4.5
Wed Dec  9 01:39:51 git version: ad7fc8908396a3351a0f07bb1e2fc4baba2802ad
Wed Dec  9 01:39:51 sys info: Darwin erh2.10gen.cc 9.6.0 Darwin Kernel Version 9.6.0: Mon Nov 24 17:37:00
    PST 2008; root:xnu-1228.9.59~1/RELEASE_I386 i386 BOOST_LIB_VERSION=1_37
Wed Dec  9 01:39:51 waiting for connections on port 20000
```

Listing A.2: Starting the Server

```
marcello@netbeams-dev:~/workspaces $ mongo 192.168.1.2:20000/netbeams
MongoDB shell version: 1.1.4
url: 192.168.1.2:20000/netbeams
connecting to: 192.168.1.2:20000/netbeams
type "help" for help
```

The description of a distributed system can be seen next section.

A.3 Experiments Execution

The experiments conducted using mongoDB can be summarized by the different data access methods provided by the technology. Each subsection describes these methods. The server could be started using the commands specified in the previous section.

A.3.1 Setting Up the Single and the Distributed Cluster Servers

mongoDB can be used in two different ways: as a single server, using the single process “mongod”, or as a distributed cluster where each cluster node runs on “mongod” managed by a cluster head “mongos”.

Before starting the main cluster head, all cluster nodes must be started. Listing A.20 shows the manual setup process performed in the Mac OS X 64bits Host. It starts two different shards, with their own respective directory, and starts the process “mongod” for each of them, forwarding the execution output to their respective log artifacts. Note that the IP address is capture in the beginning of the process because it will be used while setting up the cluster head.

In order to automate the manual process, different automation scripts were written using the Linux shell bash execution, as well as the ones used to setup mongoDB using Javascript, the language used by mongoDB shell client. First, the scripts to setup the mongoDB distributed cluster is shown in Listing A.22. It enables shard support for the different shards that must be already running, and then enable it on the collection “SondeDataContainer” for the database “netbeams”. As it is shown in the script, the declaration of the shards are executed in the “admin” database, while shards are in the database “config”, since it is the main cluster metadata server.

The first script created was the automated shell bash script to setup local shards directories and processes, as well as the cluster head metadata server (config) and the start of the cluster head “mongos”. Each local shard can be candidates to store the collected data, depending on the decisions made by the mongos cluster head. Listing A.21 expects to be executed by passing the number of local shards as a parameter to create the local directories and start the processes described. Listing A.29 shows the output of the exeuction of the setup script, incorporating the output of the Listing A.22. Note that the same IP address, shown in the setup of the cluster nodes, is also shown in the list of the shards.

A.3.2 Running the Experiments

After setting up the infrastructure, the main experiment script can be executed in the client-side. Considering the distributed server approach, the process “mongos” must be running in the server host. The infrastructure for the execution of the experiments used Sun Microsystem's VirtualBox to run different operating systems at the same time. The cluster head "mongos" was running on a Ubuntu Linux 9.04 64 bits together with the cluster metadata server "mongod" referred to as "config" server. The cluster nodes were running on the Mac OS X 64 bits on the host server, running two different shards. Finally, the execution of the experiments used Ubuntu Linux 8.04 32 Bits.

As developed for the infrastructure setup, the execution of the experiments were performed using automation scripts written in different languages. First, the experiments for the "Create" operations was written in Java as shown in Listing A.28. Similarly, remaining operations "Retrieve, Update and Delete" were written in Javascript, as shown in Listing A.24. Then, the execution of this Java class is performed by the main experiment shell script displayed in Listing A.23. It requires the number of randomly created YSI Sonde data to be executed, as shown in Listing A.3.

Listing A.3: Running main experiment shell script

```
marcello@netbeams-dev:~/workspaces/netbeams/versions/v2/persistence$ ./run-persistence-experiment
####### NetBEAMS Experiments - Persistence on MongoDB ########

Usage: ./run-persistence-experiment X, where X is the size of the workload to be inserted from NetBEAMS
    YSI Data Handler to mongoDB
```

The first experiment conducted was running the main experiment script of Listing A.23 (page 139), as one of the execution runs shows the results in Listing A.30, page 163. The total number of inserted documents measured every 20 seconds is displayed. Then, the execution of the use cases was performed, as shown in Listing A.31 page 167. Additional execution logs are shown in Section A.5.1.

A.3.3 Data Access Using Database Shell

The iterative mongo client shell offers users to verify and navigate on a given database and its collections. This first section shows the connection of the mongo client to the database "netbeams" as on Listing A.1 on page 109. The execution of the operation to count the existing number of documents in the mongoDB collection "netbeams" is shown in Listing A.4, (page 111). It highlights the query for the collections available in the netbeams database.

Listing A.4: Starting the Server

```
...
...
connecting to: 192.168.1.2:20000/netbeams
type "help" for help
> show collections
SondeDataContainer
system.indexes
> db.SondeDataContainer.count()
2419200
```

An example about data retrieving of a random document from the collection “netbeams”, “findOne()” is documented to provide this support as shown in Listing A.5, page 112. The output is the actual instance of a document, showing all the collected data in the key “observation”, and the provenance metadata in the keys “sensor.location” and “sensor.time”.

Listing A.5: Querying the database: one item

```
> db.SondeDataContainer.findOne()
{
    "_id" : ObjectId("5d6f40078ec8074bba980900"),
    "message_id" : "020d82e1-18b2-4fe2-800c-a0f68e22ea86",
    "sensor" : {
        "ip_address" : "192.168.0.91",
        "location" : {
            "latitude" : 37.89155,
            "longitude" : -122.4464
        }
    },
    "time" : {
        "valid" : "Wed Nov 04 2009 02:30:12 GMT-0800 (PST)",
        "transaction" : "Sat Nov 21 2009 03:01:33 GMT-0800 (PST)"
    },
    "observation" : {
        "WaterTemperature" : 88.58,
        "SpecificConductivity" : 180.5,
        "Conductivity" : 167.3,
        "Resistivity" : 491.89,
        "Salinity" : 0.06,
        "Pressure" : 1.109,
        "Depth" : 2.642,
        "pH" : 7.11,
        "pHmV" : -42.9,
        "Turbidity" : 0.2,
        "ODOSaturation" : 72.4,
        "ODO" : 10.85,
        "Battery" : 2.7
    }
}
```

The query based on attributes can be done using the ”dot” notation, as the user can navigate through the JSON documents. Additionally, the use of internal functions is provided by mongoDB on the result of others. The example in Listing A.6, page 113 counts the number of documents with the key "data.ph" equals to "5.64".

Listing A.6: Execution of mongo client

```
> db.SondeDataContainer.find({"data.ph":5.64}).count()
1226
```

The following example is the output of the first 2 documents from the same previous query using the function "limit()", as shown in Figure A.7. The function "limit()" is used to decrease the number of items returns by the given number.

Listing A.7: Query Element with specific projection limiting the result set size

```
> db.SondeDataContainer.find({"data.ph":5.64}).limit(2)
{"_id" : ObjectId( "d36f4007b7e7ac4a03c60000") , "sensor_ip_address" : "192.168.0.136" , "message_id" :
    "7b6624d6-0ca1-4cba-a343-f166e88da73b"
, "transaction_time" : 1252845473412 , "fact_time" : 1252845346000 , "data" : {"temperature" : "45.01" , "
    sp_condition" : "37.6" ,
"condition" : "145.8" , "resistence" : "159.77" , "salinitude" : "0.0" , "pressure" : "0.391" , "depth" :
    "0.46" , "ph" : "5.64" ,
"pH_mv" : "-62.1" , "odo_sat" : "89.7" , "odo_condition" : "59.34" , "turbidity" : "0.0" , "battery" :
    "9.4"}}
{"_id" : ObjectId( "d36f4007b7e7ac4a1fc80000") , "sensor_ip_address" : "192.168.0.136" , "message_id" :
    "7b6624d6-0ca1-4cba-a343-f166e88da73b" ,
"transaction_time" : 1252845473412 , "fact_time" : 1252845346000 , "data" : {"temperature" : "46.71" , "
    sp_condition" : "60.8" ,
"condition" : "160.6" , "resistence" : "1399.4" , "salinitude" : "0.01" , "pressure" : "1.057" , "depth" :
    "2.485" , "ph" : "5.64" ,
"pH_mv" : "-16.3" , "odo_sat" : "58.8" , "odo_condition" : "19.29" , "turbidity" : "0.2" , "battery" :
    "9.2"}}
>
```

A.3.4 Data Access Using APIs

The mongoDB server offers different drivers to access the data, as well as the REST Web Services. In order to get the results using the REST Web Services interdace, the request can be issued using a web browser. For instance, the request to the documents using the HTTP Request Variables to filter by the fields as shown in Listing A.8, page 113.

Listing A.8: REST HTTP Interface Example

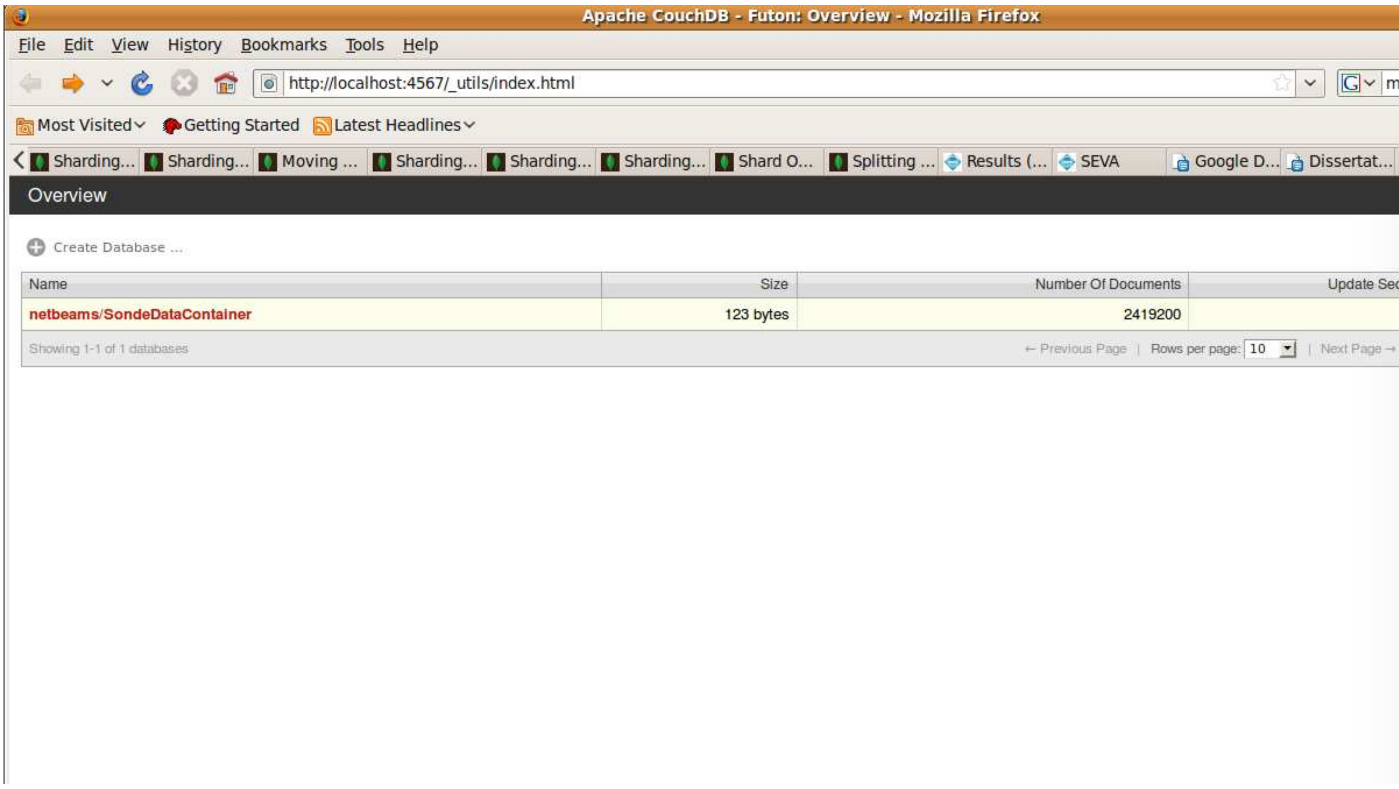
```
http://192.168.1.2:28017/netbeams/SondeDataContainer/?limit=-3&filter_observation.Conductivity=104.5

HTTP/1.0 200 OK
x-action:
x-ns: netbeams.SondeDataContainer
Content-Type: text/plain;charset=utf-8

{
  "offset" : 0,
  "rows": [
    { "_id" : { "$oid" : "616f400784ba194bc2dee200" },
      "message_id" : "c76a9898-1eee-4463-bf5d-d40a7a8af6e3",
      "sensor" : { "ip_address" : "192.168.0.60",
                   "location" : {
                       "latitude" : 37.89155, "longitude" : -122.4464}
                 },
      "time" : {
               "valid" : { "$date" : 1261229067000 },
               "transaction" : { "$date" : 1259977271704 }
               },
      "observation" : {
                "WaterTemperature" : 51.29, "SpecificConductivity" :110.7,
                "Conductivity" : 104.5, "Resistivity" : 1710.07, "Salinity" :0.01,
                "Pressure" : 1.094, "Depth" : 1.808, "pH" : 3.04,
                "pHmV" : -46.4, "Turbidity" : 0.2, "ODOSaturation" :59,
                "ODO" : 10.66, "Battery" : 7.1
                }
                ......
                ...... More records....
  }
   "total_rows" : 3 , "query" : {} , "millis" : 91 }
```

A.3.5 Visualizing Data on the Browser

Data visualisation tools for mongoDB is slowly being developed by open-source developers. Futon4mongodb, one of the open-source tools developed to visualise mongoDB data, is an adaptation of another software developed for CouchDB [cou09]. Figure A.5 shows one single collection for the YSI data containing of 1 million objects.

The collection is composed by instances of documents that represents the YSI Sonde type, being indexed by a document ID and the values being the keys defined, as shown in Figure A.6.

By clicking in one of the items, the document properties are shown for navigation purposes as shown in Figure A.5.



Name	Size	Number Of Documents	Update Se
netbeams/SondeDataContainer	123 bytes	2419200	

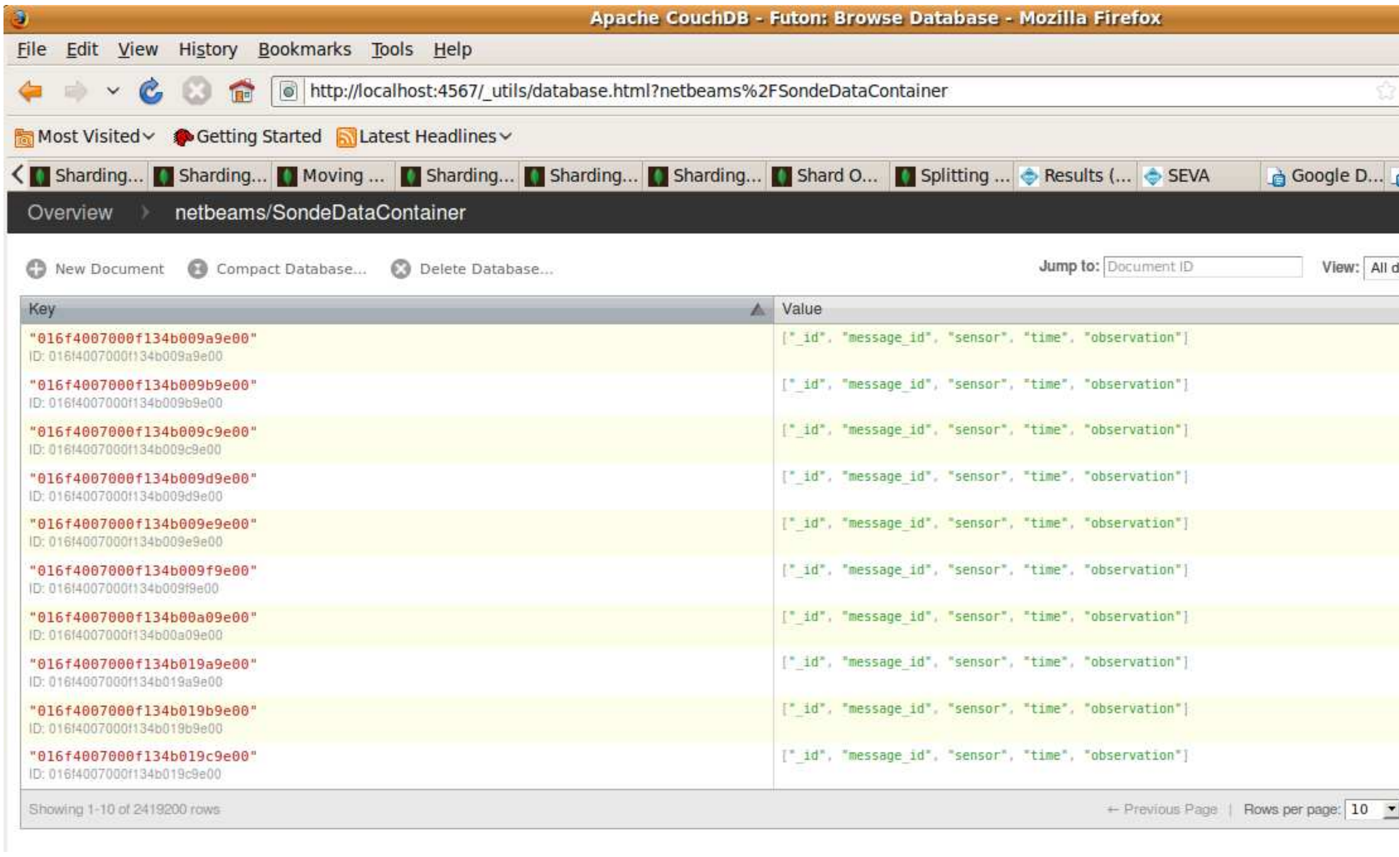
Showing 1-1 of 1 databases

Figure A.5: Viewing a partial list of data using the Futon for CouchDB/MongoDB

A.3.6 Exporting Data to Spreadsheets

mongoDB has an export facility shell called "mongoexport". It can export the data in JSON format or CSV. One may also write its own export tool in any of the languages such as Java, PHP, Python, Perl, Ruby, among others. A list of the existing drivers in different languages is provided athttp://www.mongodb.org/display/DOCS/Drivers. The following command can be executed to have the exported version of the data in CSV (read the help output of the command for details).

An example to export the data in CSV format can be seen in listing A.38, and the resulting example containing, 1 million objects can be downloaded at http://netbeams.googlecode.com/files/e
1000000-data-exported-20090913-053538.csv.tar.gz.



Key	Value
"016f4007000f134b009a9e00" ID: 016f4007000f134b009a9e00	["_id", "message_id", "sensor", "time", "observation"]
"016f4007000f134b009b9e00" ID: 016f4007000f134b009b9e00	["_id", "message_id", "sensor", "time", "observation"]
"016f4007000f134b009c9e00" ID: 016f4007000f134b009c9e00	["_id", "message_id", "sensor", "time", "observation"]
"016f4007000f134b009d9e00" ID: 016f4007000f134b009d9e00	["_id", "message_id", "sensor", "time", "observation"]
"016f4007000f134b009e9e00" ID: 016f4007000f134b009e9e00	["_id", "message_id", "sensor", "time", "observation"]
"016f4007000f134b009f9e00" ID: 016f4007000f134b009f9e00	["_id", "message_id", "sensor", "time", "observation"]
"016f4007000f134b00a09e00" ID: 016f4007000f134b00a09e00	["_id", "message_id", "sensor", "time", "observation"]
"016f4007000f134b019a9e00" ID: 016f4007000f134b019a9e00	["_id", "message_id", "sensor", "time", "observation"]
"016f4007000f134b019b9e00" ID: 016f4007000f134b019b9e00	["_id", "message_id", "sensor", "time", "observation"]
"016f4007000f134b019c9e00" ID: 016f4007000f134b019c9e00	["_id", "message_id", "sensor", "time", "observation"]

Showing 1-10 of 2419200 rows

Figure A.6: Viewing a partial list of data using the Futon4CouchDB/MongoDB

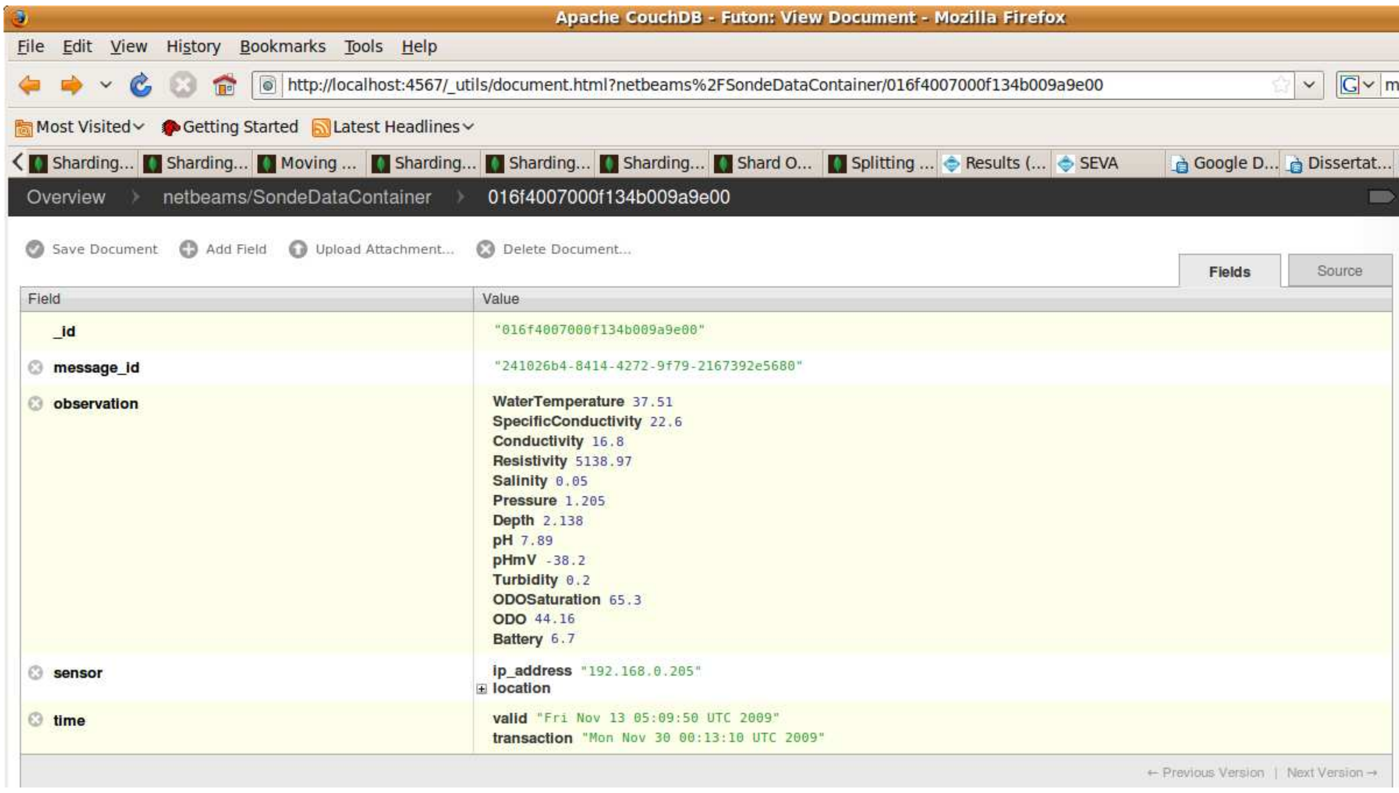
A.3.7 Exporting Data to OPeNDAP Format

During the experiments conducted for this work, the implementation of the export utility to OPeNDAP was written in Python, as shown in Listing A.25. In order to execute the command, the python programming language [pyt09] must be installed in the system. In this way, some limited documents are exported. The following command is an example of the execution of the export utility A.9:

Listing A.9: Query Element with specific projection limiting the result set size

```
marcello@netbeams-dev:~/workspaces/netbeams/versions/v2/persistence$ python data-export-python.py
    192.168.1.2:20000
```

Complete source-code listing can be reviewed in the next sections.



Field	Value
_id	"016f4007000f134b009a9e00"
message_id	"241026b4-8414-4272-9f79-2167392e5680"
observation	WaterTemperature 37.51 SpecificConductivity 22.6 Conductivity 16.8 Resistivity 5138.97 Salinity 0.05 Pressure 1.205 Depth 2.138 pH 7.89 pHmV -38.2 Turbidity 0.2 ODOSaturation 65.3 ODO 44.16 Battery 6.7
sensor	ip_address "192.168.0.205" location
time	valid "Fri Nov 13 05:09:50 UTC 2009" transaction "Mon Nov 30 00:13:10 UTC 2009"

Figure A.7: Viewing an instance of collected data using the Futon4CouchDB/MongoDB

A.4 Source-Code and Implemented Artifacts

Listing A.10: Build system using Apache ANT

```
<?xml version="1.0"?>
<project name="dsp-data-persistence" default="dsp-data-persistence.all">

    <import file="../../../../init-targets.xml"/>

    <target name="dsp-data-persistence.all" depends="init,dsp-data-persistence.compile,dsp-data-
        persistence.jar" />

    <target name="dsp-data-persistence.compile" depends="dsp-data-persistence.clean">
        <property name="dspmanagement.base.dir" value="${DSP.BUNDLES}/DSPManagement"/>
        <path id="dsp-data-persistence.classpath.id">
            <pathelement location="${THIRDPARTY}/mongodb/drivers/mongo-1.0.jar"/>
            <pathelement location="${THIRDPARTY}/stopwatch/stopwatch-0.4-with-deps.jar"/>
            <pathelement location="${XML.JAXB}/dsproperty/dist/dsproperty.jar"/>
            <pathelement location="${XML.JAXB}/message/dist/message.jar"/>
            <pathelement location="${XML.JAXB}/mouseactions/dist/mouseactions.jar"/>
            <pathelement location="${XML.JAXB}/sonde/dist/sonde.jar"/>
            <path refid="project.class.path"/>
        </path>
        <mkdir dir="${DSP.PERSISTENCE}/classes" />
        <javac destdir="${DSP.PERSISTENCE}/classes" debug="on" srcdir="${DSP.PERSISTENCE}/src">
            <classpath refid="dsp-data-persistence.classpath.id"/>
        </javac>
    </target>

    <target name="dsp-data-persistence.jar">
        <property name="MONGO_JAVA_VERSION" value="1.0" />

        <mkdir dir="${DSP.PERSISTENCE}/classes/libs"/>
        <copy todir="${DSP.PERSISTENCE}/classes/libs">
            <fileset dir="${THIRDPARTY}/mongodb/drivers/" includes="mongo-${MONGO_JAVA_VERSION}.jar"/>
            <fileset dir="${THIRDPARTY}/stopwatch/" includes="stopwatch-0.4-with-deps.jar"/>
        </copy>
        <jar basedir="${DSP.PERSISTENCE}/classes"
             jarfile="${DSP.DEPLOYMENT}/DSPDataPersistence-${DSP.VERSION}.jar"
             compress="true"
            includes="**/*"
             manifest="${DSP.PERSISTENCE}/META-INF/MANIFEST.MF"/>
    </target>

    <!-- - - - - - - - - - - - - - - - - - - -
          target: setup-mongodb
         - - - - - - - - - - - - - - - - - - - -->
    <target name="download-version" depends="init">
        <property name="MONGO_JAVA_VERSION" value="1.0" />
        <property name="MONGO_APP_VERSION" value="1.1.3" />
        <mkdir dir="${THIRDPARTY}/mongodb/drivers"/>
        <get src="http://cloud.github.com/downloads/mongodb/mongo-java-driver/mongo-${MONGO_JAVA_VERSION}.
            jar"
             dest="${THIRDPARTY}/mongodb/drivers/mongo-${MONGO_JAVA_VERSION}.jar"/>
        <get src="http://downloads.mongodb.org/linux/mongodb-linux-i686-${MONGO_APP_VERSION}.tgz"
             dest="${THIRDPARTY}/mongodb/mongodb-linux-i686-${MONGO_APP_VERSION}.tgz"/>
    </target>

    <target name="setup-mongodb" depends="init">
        <chmod perm="777" dir="${DSP.PERSISTENCE.DB}"/>
        <delete dir="${DSP.PERSISTENCE.DB}/mongodb/**/*"/>
        <delete dir="${DSP.PERSISTENCE.DB}/data/*.*"/>
        <mkdir dir="${DSP.PERSISTENCE.DB}/data"/>
        <chmod perm="777" dir="${DSP.PERSISTENCE.DB}"/>
        <unzip src="${THIRDPARTY}/mongodb/mongodb-linux-i686-1.1.3.zip" dest="${DSP.PERSISTENCE.DB}/" />
        <move file="${DSP.PERSISTENCE.DB}/mongodb-linux-i686-1.1.3" tofile="${DSP.PERSISTENCE.DB}/mongodb"
            />
        <chmod perm="+x" type="file">
         <fileset dir="${DSP.PERSISTENCE.DB}/mongodb">
           <include name="*"/>
         </fileset>
       </chmod>
    </target>

    <target name="run-futon4mongo" depends="init">
        <exec command="${THIRDPARTY}/mongodb/futon4mongo-0.2/serve" />
    </target>

    <target name="run-mongodb" depends="init">
        <exec command="${DSP.PERSISTENCE.DB}/mongodb/bin/mongod">
            <arg value="--dbpath ${DSP.PERSISTENCE.DB}/data"/>
        </exec>
    </target>

    <target name="run-experiment" depends="init">
        <exec command="${DSP.PERSISTENCE.DB}/run-persistence-experiment">
            <arg value="50"/>
        </exec>
    </target>

    <target name="dsp-data-persistence.clean">
        <delete dir="${DSP.PERSISTENCE}/classes" />
        <delete dir="${DSP.PERSISTENCE}/build" />
    </target>
</project>
```

Listing A.11: OSGi Manifest Descriptor

```
Manifest-Version: 1.0
Bundle-Activator: org.netbeams.dsp.persistence.osgi.DataPersistenceActivator
Bundle-Category: DSP
Bundle-ClassPath: .,
 libs/mongo-1.0.jar,
 libs/stopwatch-0.4-with-deps.jar
Bundle-Description: The DSP Data Persistence service
Bundle-ManifestVersion: 2
Bundle-Name: DSP Data Persistence
Bundle-SymbolicName: DSPDataPersistence
Bundle-Vendor: NetBEAMS.org
Bundle-Version: 0.1
Import-Package: org.netbeams.dsp,
 org.netbeams.dsp.message,
 org.netbeams.dsp.util,
 org.netbeams.dsp.platform.osgi,
 org.netbeams.dsp.data.property,
 org.netbeams.dsp.platform,
 org.netbeams.dsp.ysi,
 org.netbeams.dsp.demo.mouseactions,
 org.osgi.framework,
 org.apache.log4j
```

Listing A.12: DSP Deployment Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<components>

  <component>
    <name>DSPWireTransportClient</name>
    <bundle>DSPWireTransportClient-0.1.jar
    </bundle>
    <priority>71</priority>
    <required />
  </component>

  <component>
    <name>DSPWireTransportServer</name>
    <bundle>DSPWireTransportServer-0.1.jar
    </bundle>
    <priority>72</priority>
    <required />
  </component>

  <component>
```

```
    <name>DSPJSwingMouseClient</name>
    <bundle>DSPJSwingMouseClient-0.1.jar
    </bundle>
    <priority>80</priority>
    <required />
  </component>

  <component>
    <name>DSPDataPersistence</name>
    <bundle>DSPDataPersistence-0.1.jar
    </bundle>
    <priority>74</priority>
    <required />
  </component>

  <component>
    <name>DSPManager</name>
    <bundle>DSPManagement-0.1.jar</bundle>
    <priority>77</priority>
    <required />
  </component>

</components>
```

Listing A.13: DSP Matching Rules

```
<?xml version="1.0" encoding="UTF-8"?>
<config>

  <!--
    * ANY Existing information
    * LOCAL Any address representing the local Network Interface:
          LOCAL, LOCALHOST, 127.0.0.0 and current IP
    * KEEP Keep current information
    * NONE NO information should exist
  -->

  <matchRule>
    <ruleid>LocalRule</ruleid>

    <matchCriteria>
      <producerType>ANY</producerType>
      <producerAddress>ANY</producerAddress>
      <consumerType>ANY</consumerType>
      <consumerAddress>LOCAL</consumerAddress>
    </matchCriteria>

    <matchTarget>
      <consumerType>KEEP</consumerType>
      <consumerAddress>KEEP</consumerAddress>
    </matchTarget>
  </matchRule>

  <matchRule>
    <ruleid>all_messages_no_consumer_sent_to_componentmanager
    </ruleid>
    <matchCriteria>
      <producerType>ANY</producerType>
      <producerAddress>LOCAL</producerAddress>
      <consumerType>NONE</consumerType>
      <consumerAddress>NONE</consumerAddress>
    </matchCriteria>

    <matchTarget>
      <consumerType>org.netbeams.dsp.management
      </consumerType>
      <consumerAddress>LOCAL</consumerAddress>
    </matchTarget>
  </matchRule>

  <matchRule>
    <ruleid>all_messages_with_consumer_sent_to_componentmanager
    </ruleid>
    <matchCriteria>
      <producerType>ANY</producerType>
      <producerAddress>ANY</producerAddress>
      <consumerType>ANY</consumerType>
      <consumerAddress>LOCAL</consumerAddress>
    </matchCriteria>

    <matchTarget>
      <consumerType>org.netbeams.dsp.management
      </consumerType>
      <consumerAddress>LOCAL</consumerAddress>
    </matchTarget>
  </matchRule>

  <matchRule>
    <ruleid>all_messages_with_consumer_sent_server_persist
    </ruleid>
    <matchCriteria>
      <producerType>ANY</producerType>
      <producerAddress>ANY</producerAddress>
      <consumerType>ANY</consumerType>
      <consumerAddress>LOCAL</consumerAddress>
    </matchCriteria>

    <matchTarget>
      <consumerType>org.netbeams.dsp.persistence</consumerType>
      <consumerAddress>LOCAL</consumerAddress>
    </matchTarget>
  </matchRule>

  <matchRule>
    <ruleid>all_messages_no_consumer_send_to_data_persistence
    </ruleid>
    <matchCriteria>
      <producerType>ANY</producerType>
      <producerAddress>ANY</producerAddress>
      <consumerType>NONE</consumerType>
      <consumerAddress>NONE</consumerAddress>
    </matchCriteria>

    <matchTarget>
      <consumerType>org.netbeams.dsp.persistence</consumerType>
      <consumerAddress>LOCAL</consumerAddress>
    </matchTarget>
  </matchRule>


</config>
```

Listing A.14: DSP Matching Rules for Gumstix

```xml
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <matchRule>
    <ruleid>all_messages_sent_to_dsp_manager</ruleid>

    <matchCriteria>
              <nodeAddress>ANY</nodeAddress>
              <componentType>ANY</componentType>
          </matchCriteria>

          <matchTarget>
              <nodeAddress>ANY</nodeAddress>
              <componentType>org.netbeams.dsp.management</componentType>
          </matchTarget>
  </matchRule>

  <matchRule>
          <ruleid>all_messages_sent_to_dsp_manager_other</ruleid>

    <matchCriteria>
                <nodeAddress>ANY</nodeAddress>
                <componentType>ANY</componentType>
          </matchCriteria>

          <matchTarget>
              <nodeAddress>192.168.0.7</nodeAddress>
              <componentType>org.netbeams.dsp.management</componentType>
              <gatewayComponentType>org.netbeams.dsp.wiretransport.client</gatewayComponentType>
          </matchTarget>
    </matchRule>

</config>
```

Listing A.15: DSP Message used to bootstrap the DSP Data Persistence Component

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MessagesContainer uudi="7dccc68e-0998-44d1-83b6-50fb61f65967"
    creationTime="2009-03-06T15:17:18-0800" destinationHost="localhost">
    <UpdateMessage ContentType="org.netbeams.dsp.data.property.DSProperties"
        messageID="c6e565fa-99aa-4209-b358-fefcc502eb41">
        <Header>
            <CreationTime>1236381438480</CreationTime>
            <Producer>
                <ComponentType>
                    org.netbeams.dsp.platform.management.component.ComponentManager
                </ComponentType>
                <ComponentLocator>
                    <ComponentNodeId>4567</ComponentNodeId>
                    <NodeAddress>LOCAL</NodeAddress>
                </ComponentLocator>
            </Producer>
            <Consumer>
                <ComponentType>org.netbeams.dsp.persistence
                </ComponentType>
                <ComponentLocator>
                    <NodeAddress>LOCAL</NodeAddress>
                </ComponentLocator>
            </Consumer>
        </Header>
        <Body>
            <DSProperties>
                <Property name="TRANSIENT_DATA_FLUSHER_DELAY">
                    <Value>10</Value>
                </Property>
                <Property name="DATABASE_SERVER_IP_ADDRESS">
                    <Value>192.168.1.2</Value>
                </Property>
                <Property name="DATABASE_SERVER_PORT_NUMBER">
                    <Value>20000</Value>
                </Property>
                <Property name="SENSOR_1_IP_ADDRESS">
                    <Value>50</Value>
                </Property>
                <Property name="SENSOR_1_LATITUDE">
                    <Value>37.89155</Value>
                </Property>
                <Property name="SENSOR_1_LONGITUDE">
                    <Value>-122.4464</Value>
                </Property>
                <Property name="YSI_DESIRED_PROPERTIES">
                    <Value>watertemperature, conductivity, pressure</Value>
                </Property>
            </DSProperties>
        </Body>
    </UpdateMessage>
</MessagesContainer>
```

Listing A.16: OSGi Activator

```java
/*
 * Created on Sat Oct 18 21:26:01 PDT 2008
 */

package org.netbeams.dsp.persistence.osgi;

import org.apache.log4j.Logger;
import org.netbeams.dsp.persistence.controller.DSPDataPersistence;
import org.netbeams.dsp.platform.osgi.ActivatorHelper;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;

/**
 * The DataPersistenceActivator is responsible for the activator for the
 * Persistence.
 * @author Marcello de Sales (March 29, 2009)
 */
public class DataPersistenceActivator implements BundleActivator {

    private static final Logger log = Logger.getLogger(DataPersistenceActivator.class);

    /**
     * Bundle context
     */
    private BundleContext bundleContext;
    /**
     * The service registration instance
     */
    private ServiceRegistration serviceRegistration;
    /**
```

```
     * The reference to the Wire Transport client
     */
    private DSPDataPersistence persistence;

    /*
     * (non-Javadoc)
     *
     * @see org.osgi.framework.BundleActivator#start(org.osgi.framework.BundleContext)
     */
    public void start(BundleContext bc) throws Exception {
        log.info("DataPersistence.Activator.start()");
        this.bundleContext = bc;
        this.persistence = new DSPDataPersistence();
        this.serviceRegistration = ActivatorHelper.registerOSGIService(bundleContext, this.persistence);
    }

    /*
     * (non-Javadoc)
     *
     * @see org.osgi.framework.BundleActivator#stop(org.osgi.framework.BundleContext)
     */
    public void stop(BundleContext bc) throws Exception {
        log.info("DataPersistence.Activator.stop()");
        this.persistence.stopComponent();
        ActivatorHelper.unregisterOSGIService(this.bundleContext, this.serviceRegistration);
    }
}
```

Listing A.17: OSGi Service - Component

```
package org.netbeams.dsp.persistence.controller;

import java.net.UnknownHostException;
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

import org.apache.log4j.Logger;
import org.netbeams.dsp.ComponentDescriptor;
import org.netbeams.dsp.DSPComponent;
import org.netbeams.dsp.DSPContext;
import org.netbeams.dsp.DSPException;
import org.netbeams.dsp.data.property.DSProperties;
import org.netbeams.dsp.data.property.DSProperty;
import org.netbeams.dsp.message.ComponentIdentifier;
import org.netbeams.dsp.message.DSPMessagesFactory;
import org.netbeams.dsp.message.Header;
import org.netbeams.dsp.message.MeasureMessage;
import org.netbeams.dsp.message.Message;
import org.netbeams.dsp.message.MessageContent;
import org.netbeams.dsp.message.QueryMessage;
import org.netbeams.dsp.message.UpdateMessage;
import org.netbeams.dsp.persistence.model.TransientPersistenceLayer;
import org.netbeams.dsp.persistence.model.component.data.PersistentMessageUnit;
import org.netbeams.dsp.persistence.model.location.SensorLocation;
import org.netbeams.dsp.util.NetworkUtil;

import com.mongodb.MongoException;
```

```java

/**
 * The DSP Data Persistence is the main DSP Component responsible for the persistence of DSP Measure
     Messages into the
 * Database. When the component is activated, it starts the worker thread DSP Data Flusher with the flush
     rate (it
 * should be a Listener instead) given through the DSP Update Message during the activation.
 *
 * The first version of this component saves data into a mongoDB server.
 *
 * @author marcello
 *
 */
public class DSPDataPersistence implements DSPComponent {

    /**
     * Default logger
     */
    private static final Logger log = Logger.getLogger(DSPDataPersistence.class);
    /**
     * Executor responsible for the execution of the thread with a fixed delay to send the values.
     */
    public ScheduledExecutorService scheduler;
    /**
     * Main component type descriptor
     */
    private static final String COMPONENT_TYPE = "org.netbeams.dsp.persistence";
    /**
     * Default component descriptor
     */
    private static final ComponentDescriptor COMPONENT_DESCRIPTOR = null;
    /**
     * The DSPContext defines the external context
     */
    private DSPContext dspContext;
    /**
     * The DSP Node ID
     */
    private String componentNodeId;

    /**
     * Constructs a new DSP Data Persistence with an internal scheduler that flushes the current data to
         the database on
     * a given scheduled amount of time.
     */
    public DSPDataPersistence() {
        log.info("Starting DSP Data Persistence...");
        this.scheduler = Executors.newSingleThreadScheduledExecutor();
    }

    /**
     * The DSPDataFlusher updates the persistence layer with the current messages set in the queue.
     *
     * @author marcello de sales <marcello.sales@gmail.com>
     */
    private class DspDataFlusher extends Thread {

        private final Logger thLog = Logger.getLogger(DspDataFlusher.class);

        /**
         * Initializes the data flusher.
         */
        public DspDataFlusher() {
            thLog.info("Starting the DSP Data Flusher to transfer messages to database...");
        }

        public void run() {

            try {
                TransientPersistenceLayer transientLayer = TransientPersistenceLayer.INSTANCE;
                thLog.debug("Retrieving all transient messages to be flushed...");
                Set<PersistentMessageUnit> tranMsgs = Collections.synchronizedSet(transientLayer
                        .retrieveTransientMessagesUnitSet());

                DSPMongoCRUDService.INSTANCE.insertPersistentUnitMessageContents(tranMsgs);
                if (tranMsgs.size() > 0) {
                    synchronized (tranMsgs) {
                        for (PersistentMessageUnit pmu : tranMsgs) {
                            transientLayer.setMessageToFlushed(pmu);
                        }
                    }
                    thLog.debug("Preparing to transfer messages " + tranMsgs.size() + " to database...");

                } else {
                    thLog.debug("There are no transient messages in the transient persistent layer...");
                }

            } catch (Exception e) {
                thLog.error(e.getMessage(), e);
            }
        }
    }

    /**
     * Shuts down all the threads started by the scheduler.
     */
    private void shutdownFlushWorkers() {
        if (!this.scheduler.isShutdown()) {
            log.debug("Shutting down flush workers...");
            this.scheduler.shutdown();
        }
    }

    /**
     * Updates the internal Data Persistence properties.
     *
     * @param updateMessage is an update message to configure the component.
     * @throws DSPException if any problem happens during the update.
     */
    private void updateComponentProperties(UpdateMessage updateMessage) throws DSPException {

        MessageContent propertiesNode = updateMessage.getBody().getAny();

        DSProperties properties = (DSProperties) propertiesNode;
        boolean delayHasChanged = false;
        boolean databaseAddressHasChanged = false;
        for (DSProperty property : properties.getProperty()) {
            if (property.getName().equals("TRANSIENT_DATA_FLUSHER_DELAY")
                    && (System.getProperty(property.getName()) == null || !System.getProperty(property.
                        getName())
                            .equals(property.getValue()))) {
                delayHasChanged = true;
            }
            if (property.getName().equals("DATABASE_SERVER_IP_ADDRESS")
                    && (System.getProperty(property.getName()) == null || !System.getProperty(property.
                        getName())
```

```
                              .equals(property.getValue()))) {
                    databaseAddressHasChanged = true;
                }
                System.setProperty(property.getName(), property.getValue());
                log.debug("Update Property: " + property.getName() + "=" + property.getValue());
            }

            if (databaseAddressHasChanged) {
                SensorLocation.Builder builder = new SensorLocation.Builder();
                builder.setIpAddress(System.getProperty("SENSOR_1_IP_ADDRESS"));
                builder.setLatitude(Double.valueOf(System.getProperty("SENSOR_1_LATITUDE")));
                builder.setLongitude(Double.valueOf(System.getProperty("SENSOR_1_LONGITUDE")));
                Set<SensorLocation> sensors = new HashSet<SensorLocation>(1);
                sensors.add(builder.build());

                try {
                    String ipAddress = System.getProperty("DATABASE_SERVER_IP_ADDRESS");
                    int portNumber = Integer.parseInt(System.getProperty("DATABASE_SERVER_PORT_NUMBER"));
                    String propertiesList = System.getProperty("YSI_DESIRED_PROPERTIES");
                    DSPMongoCRUDService.INSTANCE.initialize(ipAddress, portNumber, sensors, propertiesList);
                    log.info("Starting the database service targeting server at IP " + ipAddress);
                    log.info("Sensors Registered: " + sensors);

                } catch (UnknownHostException uhe) {
                    log.error("Check if the IP for the database service is valid", uhe);
                    throw new DSPException(uhe);
                } catch (MongoException me) {
                    log.error(me.getMessage(), me);
                    throw new DSPException(me);
                }
            }

            long delay = this.getDelayForFlushingIntoDatabase();
            if (!this.scheduler.isShutdown()) {
                log.info("Starting scheduling the transient data flusher to wake up at every " + delay + "
                     seconds...");
                this.scheduler.scheduleWithFixedDelay(new DspDataFlusher(), delay, delay, TimeUnit.SECONDS);
            } else if (delayHasChanged) {
                this.shutdownFlushWorkers();
                log.info("Rescheduling the transient data flusher to wake up at every " + delay + " seconds...
                    ");
                this.scheduler = Executors.newSingleThreadScheduledExecutor();
                this.scheduler.scheduleWithFixedDelay(new DspDataFlusher(), delay, delay, TimeUnit.SECONDS);
            }

            // Send the acknowledgment IFF the producer was the management.
            if (updateMessage.getHeader().getProducer().getComponentType().equals("org.netbeams.dsp.management
                ")) {
                this.sendBackAcknowledge(updateMessage);
            }
        }

        private void sendBackAcknowledge(UpdateMessage message) throws DSPException {
            DSProperties props = new DSProperties();

            // Obtain original producer
            ComponentIdentifier origProducer = message.getHeader().getProducer();
            String originalMessageId = message.getMessageID();
            // Create reply message

            String localIPAddress = NetworkUtil.getCurrentEnvironmentNetworkIp();
            ComponentIdentifier producer = DSPMessagesFactory.INSTANCE.makeDSPComponentIdentifier(
                 getComponentNodeId(),
                    localIPAddress, getComponentType());

            Header header = DSPMessagesFactory.INSTANCE.makeDSPMessageHeader(null, producer, origProducer);
            header.setCorrelationID(originalMessageId);

            Message replyMsg = DSPMessagesFactory.INSTANCE.makeDSPAcknowledgementMessage(header, props);
            this.deliver(replyMsg);
        }

        /**
         * @return the value for the transport delay set from the system properties.
         */
        private long getDelayForFlushingIntoDatabase() {
            long delay = 600;
            try {
                delay = Long.valueOf(System.getProperty("TRANSIENT_DATA_FLUSHER_DELAY")).longValue();
            } catch (NumberFormatException nfe) {
                log.error(nfe.getMessage(), nfe);
                log.error("The property TRANSIENT_DATA_FLUSHER_DELAY must be set with an int value.");
                log.error("Using default value of 600 seconds for TRANSIENT_DATA_FLUSHER_DELAY");
            }
            return delay;
        }

        public void deliver(Message message) throws DSPException {
            log.debug("Delivering message, ID " + message.getMessageID());

            // Processing start-up or configuration messages
            if (message instanceof UpdateMessage) {

                log.debug("Update messages delivered: configuring DSP Data Persistence with Properties");
                this.updateComponentProperties((UpdateMessage) message);

            } else if (message instanceof QueryMessage) {

                log.debug("Query message delivered...");
                this.queryComponentProperties((QueryMessage) message);

            } else if (message instanceof MeasureMessage) {

                log.debug("Adding the DSP message to the Transient Persistence layer...");
                // add the message to the queues
                TransientPersistenceLayer.INSTANCE.addMessageToPersistenceLayer(message);
            }
        }

        /**
         * Queries the DSP Data Persistence component's current properties values.
         *
         * @param queryMessage the DSP Query Message.
         * @throws DSPException if any problem with the DSP platform occurs
         */
        private void queryComponentProperties(QueryMessage queryMessage) throws DSPException {
            MessageContent content = queryMessage.getBody().getAny();
            log.debug("Content class " + content.getClass().getName());

            if (content instanceof DSProperties) {
                log.debug("Got query configuration");

                DSProperties props = new DSProperties();

                DSProperty prp = new DSProperty();
                prp.setName("TRANSIENT_DATA_FLUSHER_DELAY");
                prp.setValue(System.getProperty("TRANSIENT_DATA_FLUSHER_DELAY"));
                props.getProperty().add(prp);

                // Obtain original producer
                ComponentIdentifier origProducer = queryMessage.getHeader().getProducer();
                String originalMessageId = queryMessage.getMessageID();
                // Create reply message

                String localIPAddress = NetworkUtil.getCurrentEnvironmentNetworkIp();
                ComponentIdentifier producer = DSPMessagesFactory.INSTANCE.makeDSPComponentIdentifier(
                    getComponentNodeId(),
                        localIPAddress, getComponentType());

                Header header = DSPMessagesFactory.INSTANCE.makeDSPMessageHeader(null, producer, origProducer)
                    ;
                header.setCorrelationID(originalMessageId);

                Message replyMsg = DSPMessagesFactory.INSTANCE.makeDSPMeasureMessage(header, props);
                this.deliver(replyMsg);
            } else {
                log.debug("Query message dropped because it did not contain DSProperties: " + content.getClass
                    ().getName());
            }
        }

        public Message deliverWithReply(Message message) throws DSPException {
            log.error("Delivering message with reply not implemented.");
            return null;
        }

        public Message deliverWithReply(Message message, long waitTime) throws DSPException {
            log.error("Delivering message with reply with delay not implemented.");
            return null;
        }

        public ComponentDescriptor getComponentDescriptor() {
            return COMPONENT_DESCRIPTOR;
        }

        public void startComponent() throws DSPException {
            log.info("Starting component");
        }

        public void stopComponent() throws DSPException {
            log.info("Stopping component");
            this.shutdownFlushWorkers();
        }

        public String getComponentNodeId() {
            return this.componentNodeId;
        }

        public String getComponentType() {
            return COMPONENT_TYPE;
        }

        public void initComponent(String componentNodeId, DSPContext context) throws DSPException {
            this.dspContext = context;
            this.componentNodeId = componentNodeId;
            log.info("Initializing component ID " + componentNodeId + " with context " + this.dspContext.
                toString());
        }
    }
```

Listing A.18: DSP Data to Mongo DB Service

```java
package org.netbeams.dsp.persistence.controller;

import java.net.UnknownHostException;
import java.text.DecimalFormat;
import java.util.Calendar;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.StringTokenizer;

import org.apache.log4j.Logger;
import org.netbeams.dsp.demo.mouseactions.MouseAction;
import org.netbeams.dsp.demo.mouseactions.MouseActionsContainer;
import org.netbeams.dsp.message.MessageContent;
import org.netbeams.dsp.persistence.model.component.data.PersistentMessageUnit;
import org.netbeams.dsp.persistence.model.location.SensorLocation;
import org.netbeams.dsp.ysi.SondeDataContainer;
import org.netbeams.dsp.ysi.SondeDataType;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.Mongo;
import com.mongodb.MongoException;

/**
 * This class is responsible for the CRUD services with the mongoDB. Although a regular CRUD service
     includes the Update
 * (U) of the CRUD, it will NOT be implemented as data in sensor networks DOES NOT CHANGE.
 *
 * C - Create: Any document created will be created <li>sensor_ip_address - the ip of the sensor <li>
     message_id - the
 * identification of the DSP message that transferred the message <li>transaction_time - the time when the
      data was
 * received by the server. <li>fact_time - the time when the data was transferred from the sensor. <li>
     data - the actual
 * sample data
 *
 * R - Retrieve: Retrieve any created documents from the mongoDB given the message type U - Update: NOT
     Implemented as
 * data can't be updated. D - Delete: Deletes an item from a collection.
 *
 * @author Marcello de Sales (marcello.sales@gmail.com)
 *
 */
public enum DSPMongoCRUDService {

    INSTANCE();

    /**
     * Default logger
     */
    private static final Logger log = Logger.getLogger(DSPMongoCRUDService.class);

    private static final DecimalFormat ONE_DECIMAL_FORMATTER = new DecimalFormat("###0.0");
    private static final DecimalFormat TWO_DECIMALS_FORMATTER = new DecimalFormat("###0.00");
    private static final DecimalFormat THREE_DECIMALS_FORMATTER = new DecimalFormat("###0.000");

    private static final boolean SETUP_INDEXES = true;
    /**
     * The default mongoDB database name for the project.
     */
    private static final String NETBEAMS_DATASTORE_NAME = "netbeams";
    /**
     * The local cache for the collection names based on the content type
     */
    private static final Map<String, DBCollection> netBeamscollectionsCache = new HashMap<String,
        DBCollection>();
    /**
     * The IP address of the mongo server. Either the single server of the cluster head when in sharded
         environment
     */
    private String serverIpAddress;
    private int serverPortNumber;
    /**
     * Tracks if the service was initialized before being used. After getting the INSTANCE instance, the
         method
     * initialize must be called.
     */
    private static boolean initialized;
    /**
     * The list of properties to be included. temperature = true salinity = true
     */
    private Map<String, Boolean> includeProperty = new HashMap<String, Boolean>(15);

    private static boolean finishedInit;

    /**
     * Initializes the mongo CRUD service, with the given IP address from the mongo server, the current
         list of known
     * sensors and the list of desired properties to be persisted for the YSI sonde data.
     *
     * @param serverIpAddress is the IP address of the mongo server. Must be numerical IP address or
         machine name.
     *            (127.0.0.1, localhost).
     * @param sensorsLocation is the list of known sensor locations
     * @param propertiesList is the list of attributes to be persisted in the database, as a string
         delimited by a comma
     *            ",". For instance "salinity,temperature,turbinity".
     * @throws UnknownHostException
     * @throws MongoException
     */
    public void initialize(String serverIpAddress, int serverPortNumber, Set<SensorLocation>
        sensorsLocation, String propertiesList)
            throws UnknownHostException, MongoException {
        initialized = true;
        this.serverIpAddress = serverIpAddress;
        this.serverPortNumber = serverPortNumber;
        this.setupSelectedProperties(propertiesList);
        finishedInit = true;
        log.info("Persistence Service Initialized...");
    }

    /**
     * @return if the service has been initialized.
     */
    public static boolean hasFinished() {
        return finishedInit;
    }

    /**
     * Sets the selected properties to be selected when making data persistent.
     *
     * @param propertiesList is the list of properties delimited by comma ",".
     */
    private void setupSelectedProperties(String propertiesList) {
        if (propertiesList == null || propertiesList.equals("") && !propertiesList.contains(",")) {
            // all properties will be selected.
            return;
        }
        propertiesList = propertiesList.trim().replace(" ", "").toLowerCase();
        StringTokenizer tokenizer = new StringTokenizer(propertiesList, ",");
        while (tokenizer.hasMoreTokens()) {
            String token = tokenizer.nextToken();
            this.includeProperty.put(token, true);
        }
        log.info("Setup Properties List: " + propertiesList);
    }

    /**
     * @param persistentMessage is the DSP Message that was persisted after transmission.
     * @return the DBCollection instance based on the Message Content Type from the DSP Message that
           transmitted from
     *         the PersistentMessageUnit instance.
     * @throws UnknownHostException
     * @throws MongoException
     */
    public DBCollection getPersistenceStorage(PersistentMessageUnit persistentMessage) throws
         UnknownHostException,
            MongoException {
        DBCollection dbCollection = netBeamscollectionsCache.get(persistentMessage.getMessageContentType()
             );
        if (dbCollection == null) {
            DB netbeamsDb = this.getNetbeamMongoDb();
            dbCollection = netbeamsDb.getCollection(persistentMessage.getMessageContentType());
            netBeamscollectionsCache.put(persistentMessage.getMessageContentType(), dbCollection);
            if (SETUP_INDEXES) {
                this.setupMetadataIndexes(dbCollection);
                this.setupDataIndexes(dbCollection);
            }
        }
        return dbCollection;
    }

    /**
     * @return the default Mongo instance based on the default NETBEAMS_DATASTORE_NAME.
     * @param serverIpAddress is the IP address of the server where the mongo DB is running. It is the
           main DB server,
     *             on a single db environment, or the main head of the mongo DB cluster in a sharded
           environment.
     * @throws UnknownHostException
     * @throws MongoException
     */
    public DB getNetbeamMongoDb() throws UnknownHostException, MongoException {
        Mongo dbInstance = new Mongo(this.serverIpAddress, this.serverPortNumber);
        return dbInstance.getDB(NETBEAMS_DATASTORE_NAME);
    }

    /**
     * Adds the needed indexes for the metadata used for any collected data.
     *
     * @param dataCollection
     */
    private void setupMetadataIndexes(DBCollection dataCollection) {
        dataCollection.ensureIndex(new BasicDBObject("message_id", "1"), true);
        dataCollection.ensureIndex(new BasicDBObject("sensor.ip_address", "1"), true);
        dataCollection.ensureIndex(new BasicDBObject("sensor.location.latitude", "1"), true);
        dataCollection.ensureIndex(new BasicDBObject("sensor.location.longitude", "1"), true);
        dataCollection.ensureIndex(new BasicDBObject("time.valid", "1"), true);
        dataCollection.ensureIndex(new BasicDBObject("time.transaction", "1"), true);
    }

    /**
     * Adds the needed indexes for the sonde data collection
     *
     * @param dbCollection
     */
    private void setupDataIndexes(DBCollection sondeDataCollection) {

        final String DATA_PREFIX = "observation.";
        if (this.includeProperty.size() == 0 || this.includeProperty.get("watertemperature") == Boolean.
            TRUE) {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "WaterTemperature", "1"), true
                );
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("specificconductivity") ==
            Boolean.TRUE) {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "SpecificConductivity", "1"),
                true);
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("conductivity") == Boolean.TRUE)
            {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "Conductivity", "1"), true);
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("resistivity") == Boolean.TRUE) {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "Resistivity", "1"), true);
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("salinity") == Boolean.TRUE) {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "Salinity", "1"), true);
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("pressure") == Boolean.TRUE) {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "Pressure", "1"), true);
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("depth") == Boolean.TRUE) {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "Depth", "1"), true);
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("ph") == Boolean.TRUE) {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "pH", "1"), true);
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("phmv") == Boolean.TRUE) {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "pHmV", "1"), true);
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("turbidity") == Boolean.TRUE) {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "Turbidity", "1"), true);
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("odosaturation") == Boolean.TRUE)
            {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "ODOSaturation", "1"), true);
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("odo") == Boolean.TRUE) {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "ODO", "1"), true);
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("battery") == Boolean.TRUE) {
            sondeDataCollection.ensureIndex(new BasicDBObject(DATA_PREFIX + "Battery", "1"), true);
        }
    }

    /**
     * Inserts the DSP Message Content into the mongoDB as it is extracted and converted from the given
     * PersistentMessageUnit.
     *
     * @param tranMsg is the PersistentMessageUnit containing information about the sensor location and
        the message.
     * @param serverIpAddress is the IP address of the server to insert this collection. It must be a
        single server or
     *            the master node of a mongo cluster in a sharded environment.
     * @throws UnknownHostException
     * @throws MongoException
     */
    public synchronized void insertPersistentUnitMessageContents(Set<PersistentMessageUnit> tranMsgs)
            throws UnknownHostException, MongoException {
        if (!initialized) {
            throw new IllegalStateException("Initialize the service before inserting");
        }
        log.info("Inserting transient DSP Messages: " + tranMsgs.size() + " messages");
        // Start the mongoDB Transaction mechanism
        this.getNetbeamMongoDb().requestStart();
        for (PersistentMessageUnit tranMsg : tranMsgs) {
            DBCollection netbeamsDbCollection = this.getPersistenceStorage(tranMsg);
            MessageContent messageContent = tranMsg.getDspMessage().getBody().getAny();

            BasicDBObject docValue = null;
            long factTime = 0;
            if (messageContent instanceof SondeDataContainer) {
                SondeDataContainer sondeContainer = (SondeDataContainer) messageContent;
                for (SondeDataType sondeData : sondeContainer.getSondeData()) {
                    // build the document value
                    docValue = buildValueSegment(sondeData);
                    // extract the fact time from the message, adding to the key
                    factTime = sondeData.getDateTime().getTimeInMillis();
                }
            } else if (messageContent instanceof MouseActionsContainer) {
                MouseActionsContainer mouseActionsContainer = (MouseActionsContainer) messageContent;
                for (MouseAction data : mouseActionsContainer.getMouseAction()) {
                    // build the document value
                    docValue = buildValueSegment(data);
                    // extract the fact time from the message, adding to the key
                    factTime = data.getCollectionTime();
                }
            }
            // build the document key
            BasicDBObject docKey = buildKeySegment(tranMsg, factTime, docValue);
            // insert the final collection
            netbeamsDbCollection.insert(docKey);
            // flush the message from transient
        }
        // Terminate the mongoDB Transaction
        this.getNetbeamMongoDb().requestDone();
    }

    /**
     * @param tranMsg is the persistent message unit holding an instance of a transmitted DSP message.
     * @param validTime is the time in which the data was collected.
     * @param docValue is the value of the document that will be added into the "data" attribute.
     * @return the Key Segment for the given Persistent Message Unit's DSP Message instance. It extracts
        the sensor's IP
     *          address, the Message ID and the Transaction time from the Unit, which constitutes the Key.
     * @throws MongoException
     * @throws UnknownHostException
     */
    public BasicDBObject buildKeySegment(PersistentMessageUnit tranMsg, long validTime, BasicDBObject
        docValue)
            throws UnknownHostException, MongoException {
```

```
        BasicDBObject doc = new BasicDBObject();
        doc.put("message_id", tranMsg.getDspMessage().getMessageID());
        // doc.put("sensor", new DBRef(this.getNetbeamMongoDb(), "sensors", )); VERY SLOW

        BasicDBObject sensorDoc = new BasicDBObject();
        sensorDoc.put("ip_address", tranMsg.getSensorLocation().getIpAddress());
        BasicDBObject locDoc = new BasicDBObject();
        locDoc.put("latitude", tranMsg.getSensorLocation().getLatitude());
        locDoc.put("longitude", tranMsg.getSensorLocation().getLongitude());
        sensorDoc.put("location", locDoc);
        doc.put("sensor", sensorDoc);

        BasicDBObject dimensionDoc = new BasicDBObject();
        Calendar validTimeCal = Calendar.getInstance();
        validTimeCal.setTimeInMillis(validTime);
        dimensionDoc.put("valid", validTimeCal.getTime());
        Calendar transTime = Calendar.getInstance();
        transTime.setTimeInMillis(tranMsg.getCollectionTimeMilliseconds());
        dimensionDoc.put("transaction", transTime.getTime());
        doc.put("time", dimensionDoc);

        doc.put("observation", docValue);
        return doc;
    }

    /**
     * @param sondeData is the instance of the sonde data.
     * @return the basic DB Object representation of the sonde data type with all the properties.
     */
    public BasicDBObject buildValueSegment(SondeDataType sondeData) {
        BasicDBObject docValue = new BasicDBObject();
        if (this.includeProperty.size() == 0 || this.includeProperty.get("watertemperature") == Boolean.
            TRUE) {
            docValue.put("WaterTemperature", Double.valueOf(TWO_DECIMALS_FORMATTER.format(sondeData.
                getTemp())));
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("specificconductivity") ==
            Boolean.TRUE) {
            docValue.put("SpecificConductivity", Double.valueOf(ONE_DECIMAL_FORMATTER.format(sondeData.
                getSpCond())));
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("conductivity") == Boolean.TRUE)
            {
            docValue.put("Conductivity", Double.valueOf(ONE_DECIMAL_FORMATTER.format(sondeData.getCond()))
                );
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("resistivity") == Boolean.TRUE) {
            docValue.put("Resistivity", Double.valueOf(TWO_DECIMALS_FORMATTER.format(sondeData.getResist()
                )));
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("salinity") == Boolean.TRUE) {
            docValue.put("Salinity", Double.valueOf(TWO_DECIMALS_FORMATTER.format(sondeData.getSal())));
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("pressure") == Boolean.TRUE) {
            docValue.put("Pressure", Double.valueOf(THREE_DECIMALS_FORMATTER.format(sondeData.getPress()))
                );
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("depth") == Boolean.TRUE) {
            docValue.put("Depth", Double.valueOf(THREE_DECIMALS_FORMATTER.format(sondeData.getDepth())));
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("ph") == Boolean.TRUE) {
            docValue.put("pH", Double.valueOf(TWO_DECIMALS_FORMATTER.format(sondeData.getPH())));
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("phmv") == Boolean.TRUE) {
            docValue.put("pHmV", Double.valueOf(ONE_DECIMAL_FORMATTER.format(sondeData.getPhmV())));
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("turbidity") == Boolean.TRUE) {
            docValue.put("Turbidity", Double.valueOf(ONE_DECIMAL_FORMATTER.format(sondeData.getTurbid())))
                ;
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("odosaturation") == Boolean.TRUE)
             {
            docValue.put("ODOSaturation", Double.valueOf(ONE_DECIMAL_FORMATTER.format(sondeData.getODOSat
                ())));
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("odo") == Boolean.TRUE) {
            docValue.put("ODO", Double.valueOf(TWO_DECIMALS_FORMATTER.format(sondeData.getODOConc())));
        }
        if (this.includeProperty.size() == 0 || this.includeProperty.get("battery") == Boolean.TRUE) {
            docValue.put("Battery", Double.valueOf(ONE_DECIMAL_FORMATTER.format(sondeData.getBattery())));
        }
        return docValue;
    }

    /**
     * @param sondeData is the instance of the sonde data.
     * @return the basic DB Object representation of the sonde data type with all the properties.
     */
    public BasicDBObject buildValueSegment(MouseAction mouseData) {
        BasicDBObject docValue = new BasicDBObject();
        docValue.put("x", mouseData.getX());
        docValue.put("y", mouseData.getY());
        docValue.put("button", mouseData.getButton().toString());
        docValue.put("event", mouseData.getEvent().toString());
        return docValue;
    }
}
```

Listing A.19: DSP Message with a YSI sonde data payload

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MessagesContainer uudi="24929c29-60ee-4d17-af08-64d9446277ef"
        creationTime="2009-03-06T15:17:18-0800" destinationHost="192.168.0.106">
   <MeasureMessage ContentType="org.netbeams.dsp.ysi"
         messageID="435a61f6-370f-458d-aeb7-6e92270a79cb">
      <Header>
         <CreationTime>1236381438480</CreationTime>

         <Producer>
            <ComponentType>
                 org.netbeams.dsp.platform.management.component.ComponentManager
            </ComponentType>

            <ComponentLocator>
               <ComponentNodeId>1234</ComponentNodeId>
               <NodeAddress>192.168.0.103</NodeAddress>
            </ComponentLocator>
         </Producer>

         <Consumer>
            <ComponentType>
                 org.netbeams.dsp.wiretransport.client
            </ComponentType>

            <ComponentLocator>
                <NodeAddress>LOCAL</NodeAddress>
            </ComponentLocator>
        </Consumer>
    </Header>
    <Body>
        <SondeDataContainer>
            <soundeData date="15:17:18" time="03-06-2009">
                <Temp>21.20</Temp>
                <SpCond>193</SpCond>
                <Cond>179</Cond>
                <Resist>5588.40</Resist>
                <Sal>0.09</Sal>
                <Press>0.084</Press>
                <Depth>0.059</Depth>
                <pH>7.98</pH>
                <phmV>-79.6</phmV>
                <ODOSat>99.5</ODOSat>
                <ODOConc>8.83</ODOConc>
                <Turbid>0.4</Turbid>
                <Battery>8.7</Battery>
            </soundeData>
        </SondeDataContainer>
    </Body>
  </MeasureMessage>
</MessagesContainer>
```

A.5 Experiment Implementation

Listing A.20: Remote Shards Setup Execution Log

```
marcello-de-saless-macbook-pro:persistence marcello$ ifconfig
en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
  inet6 fe80::226:8ff:fee3:5538%en1 prefixlen 64 scopeid 0x5
  inet 192.168.1.2 netmask 0xffffff00 broadcast 192.168.1.255
  ether 00:26:08:e3:55:38
  media: autoselect status: active
  supported media: autoselect

marcello-de-saless-macbook-pro:persistence marcello$ mongod --dbpath /mongo-data/shard-1/ --port 20001 |
     tee logs/shard1.log &
[4] 4365
marcello-de-saless-macbook-pro:persistence marcello$ Sat Dec 12 16:38:42 Mongo DB : starting : pid = 4364
     port = 20001 dbpath = /mongo-data/shard-1/ master = 0 slave = 0  64-bit
Sat Dec 12 16:38:42 db version v1.2.0, pdfile version 4.5
Sat Dec 12 16:38:42 git version: 2add01f37ddf5a262dbc66f81353e88f38723dc6
Sat Dec 12 16:38:42 sys info: Darwin erh2.10gen.cc 9.6.0 Darwin Kernel Version 9.6.0: Mon Nov 24 17:37:00
     PST 2008; root:xnu-1228.9.59~1/RELEASE_I386 i386 BOOST_LIB_VERSION=1_37
Sat Dec 12 16:38:42 waiting for connections on port 20001

marcello-de-saless-macbook-pro:persistence marcello$ mongod --dbpath /mongo-data/shard-2/ --port 20002 |
     tee logs/shard2.log &
[3] 4341
Sat Dec 12 16:38:27 db version v1.2.0, pdfile version 4.5
Sat Dec 12 16:38:27 git version: 2add01f37ddf5a262dbc66f81353e88f38723dc6
Sat Dec 12 16:38:27 sys info: Darwin erh2.10gen.cc 9.6.0 Darwin Kernel Version 9.6.0: Mon Nov 24 17:37:00
     PST 2008; root:xnu-1228.9.59~1/RELEASE_I386 i386 BOOST_LIB_VERSION=1_37
```

```
Sat Dec 12 16:38:27 waiting for connections on port 20002

marcello-de-saless-macbook-pro:persistence marcello$ ps aux | grep mongo
marcello  1461   4.3 18.6  4359300 1556880   ??  S     2:54PM   8:29.60 /Applications/VirtualBox.app/
     Contents/MacOS/../Resources/VirtualBoxVM.app/Contents/MacOS/VirtualBoxVM --comment NetBEAMS +

marcello-de-saless-macbook-pro:persistence marcello$ ps aux | grep shard
marcello  5100   0.0  0.0  2435468    320 s000  R+    4:50PM   0:00.00 grep shard
marcello  4365   0.0  0.0  2434756    432 s000  S     4:38PM   0:00.00 tee logs/shard1.log
marcello  4364   0.0  0.0  2452168   2416 s000  S     4:38PM   0:00.02 mongod --dbpath /mongo-data/shard
     -1/ --port 20001
marcello  4341   0.0  0.0  2434756    432 s000  S     4:38PM   0:00.00 tee logs/shard2.log
marcello  4340   0.0  0.0  2452168   2464 s000  S     4:38PM   0:00.03 mongod --dbpath /mongo-data/shard
     -2/ --port 20002

marcello-de-saless-macbook-pro:persistence marcello$ ls -lah /mongo-data/shard-1/
total 8
drwxr-xr-x  3 marcello  staff   102B Dec 12 01:49 .
drwxr-xr-x  4 marcello  admin   136B Dec 12 15:00 ..
-rwxr-xr-x  1 marcello  staff     5B Dec 12 15:01 mongod.lock

marcello-de-saless-macbook-pro:persistence marcello$ ls -lah /mongo-data/shard-2
total 20865032
drwxr-xr-x  13 marcello  staff   442B Dec 12 14:53 .
drwxr-xr-x   4 marcello  admin   136B Dec 12 15:00 ..
-rwxr-xr-x   1 marcello  staff     5B Dec 12 15:02 mongod.lock
-rw-------   1 marcello  staff    64M Dec 12 15:31 netbeams.0
-rw-------   1 marcello  staff   128M Dec 12 15:31 netbeams.1
-rw-------   1 marcello  staff   256M Dec 12 15:31 netbeams.2
-rw-------   1 marcello  staff   512M Dec 12 15:31 netbeams.3
-rw-------   1 marcello  staff   1.0G Dec 12 15:31 netbeams.4
-rw-------   1 marcello  staff   2.0G Dec 12 15:31 netbeams.5
-rw-------   1 marcello  staff   2.0G Dec 12 15:31 netbeams.6
-rw-------   1 marcello  staff   2.0G Dec 12 15:31 netbeams.7
-rw-------   1 marcello  staff   2.0G Dec 12 04:10 netbeams.8
-rw-------   1 marcello  staff    16M Dec 12 15:31 netbeams.ns
```

Listing A.21: Remote Shards Cluster Head and Metadata Setup Bash Script

```
#!/bin/bash
# Using "seq"

echo "############## Setting up shards for experiment..."
if [ -z "$1" ]; then
    echo "Usage: $0 x, where x is the number of shards to be created"
    exit
fi

NUMBER_SHARDS=$1
SHARDS_DIR=shards
SHARD_DIR_PREFIX=shard

rm -rf $SHARDS_DIR
mkdir $SHARDS_DIR

mkdir data/$SHARDS_DIR
for shard_number in $(seq $NUMBER_SHARDS)
do
  mkdir data/$SHARDS_DIR/$SHARD_DIR_PREFIX-$shard_number
  mongod --dbpath data/$SHARDS_DIR/$SHARD_DIR_PREFIX-$shard_number/ --port 2000$shard_number | tee logs/
      shard-$shard_number.log &
```

```
done

mkdir data/$SHARDS_DIR/config
mongod --dbpath data/$SHARDS_DIR/config --port 10000 | tee logs/shard-config.log &

ps aux | grep mongod

mongos -vvv --configdb localhost:10000 | tee logs/mongos-cluster-head.log &

sleep 1

mongo < setup-mongo-shards.js | tee logs/setup-mongo-shards-for-netbeams.log &
```

Listing A.22: Shards Setup in Javascript

```
admin = db.getSisterDB("admin")
config = db.getSisterDB("config")

admin.runCommand( { addshard: "192.168.1.2:20001" } )
admin.runCommand( { addshard: "192.168.1.2:20002" } )
admin.runCommand( { addshard: "localhost:20001", allowLocal: true } )

admin.runCommand( { listshards:1 } )

admin.runCommand( { enablesharding: "netbeams" } )
admin.runCommand( { shardcollection: "netbeams.SondeDataContainer", key: { "observation.pH" : 1} } )

netbeams = db.getSisterDB("netbeams")
netbeams.SondeDataContainer.ensureIndex( { "message_id":1 } )
netbeams.SondeDataContainer.ensureIndex( { "sensor.ip_address":1 } )
netbeams.SondeDataContainer.ensureIndex( { "sensor.location.latitude":1 } )
netbeams.SondeDataContainer.ensureIndex( { "sensor.location.longitude":1 } )
netbeams.SondeDataContainer.ensureIndex( { "time.valid":1 } )
netbeams.SondeDataContainer.ensureIndex( { "time.transaction":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.WaterTemperature":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.SpecificConductivity":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.Conductivity":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.Resistivity":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.Salinity":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.Pressure":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.Depth":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.pH":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.pHmV":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.Turbidity":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.ODOSaturation":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.ODO":1 } )
netbeams.SondeDataContainer.ensureIndex( { "observation.Battery":1 } )

config.chunks.find()
```

Listing A.23: Main Client Experiment Execution Bash Script

```
#!/bin/sh

#Make sure to run the steps 1 and 2 if running the experiment with sharded environment.
echo "####### NetBEAMS Experiments - Persistence on MongoDB ########\n"

if [ -z "$1" ]; then
```

```
    echo "Usage: $0 X, where X is the size of the workload to be inserted from NetBEAMS YSI Data Handler to
        mongoDB"
    exit
fi

DATA_DIR=data
LOG_SUFIX_DATE=$(date +%Y%m%d-%H%M%S)
NUMBER_SAMPLES=$1
DATABASE_NAME=netbeams
COLLECTION_NAME=SondeDataContainer

JAVA_MEM_MIN=-Xms200m
JAVA_MEM_MAX=-Xmx1800m
MONGO_JAVA=../thirdparty/mongodb/drivers/mongo-1.0.jar
MONGO_SERVER_IP=192.168.1.10:27017
PROPERTIES_LIST=$3
JAVA_EXPERIMENT_CLASSPATH=../runtime/deployment/DSPDataPersistence-0.1.jar:../runtime/deployment/
    DSPSondeProducer-0.1.jar:../apps/xml/jaxb/message/dist/message.jar:../apps/xml/jaxb/sonde/dist/sonde.
    jar:$MONGO_JAVA:../thirdparty/stopwatch/stopwatch-0.4-with-deps.jar
JAVA_EXPERIMENT_CLASS=org.netbeams.dsp.persistence.controller.DSPMessageToMongoDBExperiment

MONGO_SERVER_LOG=logs/experiment-$NUMBER_SAMPLES-mongodb-server-status-$LOG_SUFIX_DATE.log
MONGO_CLIENT_LOG=logs/experiment-$NUMBER_SAMPLES-mongodb-client-status-$LOG_SUFIX_DATE.log
NETBEAMS_2_MONGO_LOG=logs/experiment-$NUMBER_SAMPLES-netbeams-to-mongodb-$LOG_SUFIX_DATE.log
MONGO_DATA_EXPORT_CSV=logs/experiment-$NUMBER_SAMPLES-data-exported-$LOG_SUFIX_DATE.csv

echo "########### Netbeams to MongoDB Experiment $LOG_SUFIX ############# "
echo "\n# 1. Cleaning any existing MongoDB data at '$DATA_DIR'\n"
#rm -rf $DATA_DIR/*
#ls -lah $DATA_DIR/shard-a
#ls -lah $DATA_DIR/shard-b

echo "\n# 2. Starting MongoDB Server with 2 local shards (1 remote)... NetBEAMS data will be saved at dir
    '$DATA_DIR'\n"

############ Single Server Environment #################
#mongod --dbpath $DATA_DIR | tee $MONGO_SERVER_LOG &

############ Data-Centric Cluster - Sharded Environment ##################
###############  Shards ##################
#mongod --dbpath data/shard-a/ --port 10000 | tee logs/shard-a.log &
#mongod --dbpath data/shard-b/ --port 10001 | tee logs/shard-b.log &
#mongod --dbpath data/shards-config/ --port 20000 | tee logs/shards-config.log &
####################
############# Main Cluster Head ########################
#mongos --configdb localhost:20000 | tee $MONGO_SERVER_LOG &
###################################################

echo "\n# 3. Ready to run Java experiment with $NUMBER_SAMPLES samples\n"

java $JAVA_MEM_MIN $JAVA_MEM_MAX -classpath $JAVA_EXPERIMENT_CLASSPATH $JAVA_EXPERIMENT_CLASS
    $MONGO_SERVER_IP $NUMBER_SAMPLES $PROPERTIES_LIST | tee $NETBEAMS_2_MONGO_LOG

echo "\n* 4. Exporting the generated mongoDB data in CSV file format\n"
echo "Executing the export command \nmongoexport -h $MONGO_SERVER_IP -d $DATABASE_NAME -c $COLLECTION_NAME
    --query \"{\"time.valid\" : { $gte:new Date(2009,11,1) , $lt:new Date(2009,11,8) }\" --csv -f \"_id,
    sensor.ip_address,time.valid,observation.WaterTemperature,observation.SpecificConductivity,observation
    .Conductivity,observation.Resistivity,observation.Salinity,observation.Pressure,observation.Depth,
    observation.pH,observation.pHmV,observation.Turbidity,observation.ODOSaturation,observation.ODO,
    observation.Battery\" -o $MONGO_DATA_EXPORT_CSV\n"

#mongoexport -h $MONGO_SERVER_IP -d $DATABASE_NAME -c $COLLECTION_NAME --query "{\"time.valid\" : { \"$gte
    \":{\"$date\":1257829200000}, $lt:{\"$date\" : 1257829200100}}}" --csv -f "_id,sensor.ip_address,time.
    valid , observation . WaterTemperature , observation . SpecificConductivity , observation . Conductivity ,
    observation . Resistivity , observation . Salinity , observation . Pressure , observation . Depth , observation . pH ,
    observation . pHmV, observation . Turbidity , observation . ODOSaturation , observation . ODO, observation . Battery"
    -o $MONGO_DATA_EXPORT_CSV

echo "\n* 5. Experiments Results on the following logs:"
echo "- mongoDB Server log output: $MONGO_SERVER_LOG"
echo "- mongoDB Client log output: $MONGO_CLIENT_LOG"
echo "- NetBEAMS to mongoDB data transfer log output: $NETBEAMS_2_MONGO_LOG (after running the client)"
echo "- mongoDB data exported in CSV format: $MONGO_DATA_EXPORT_CSV"

echo "\n* 6. mongoDB data dir '$DATA_DIR' size after experiments...\n"
#du -h $DATA_DIR/shard-a/
#ls -lah $DATA_DIR/shard-b/

echo "\n* 7. Accessing the persisted sensor data after the experiments...\n"
echo " - Through the mongoDB Client using via command line \"mongo $DATABASE_NAME\""
echo " - Through the mongoDB REST Web Services client using HTTP GET Requests http://$MONGO_SERVER_IP
    :28017/$DATABASE_NAME/$COLLECTION_NAME/?limit=-5"
echo " - Through Futon2Mongo, the check the web client at http://127.0.0.1:4567/_utils/index.html"

echo "\n* 8. Running the mongDB client after the experiments...\n"
echo " - The database name is '$DATABASE_NAME'. The collection name is '$COLLECTION_NAME'"
echo " - Type 'show collections' to show all the collections in the current database"
echo " - Type 'use $DATABASE_NAME' to change to that database."
echo " - Type 'db.$COLLECTION_NAME.*' to issue a command to the collection '$COLLECTION_NAME'"
echo " - Ex: 'db.$COLLECTION_NAME.count()' = Global Shard Query, returns the number of docs on the
    collection '$COLLECTION_NAME'"
echo " -     'db.$COLLECTION_NAME.findOne()' = Target Shard Query, returns the first element of the
    collection '$COLLECTION_NAME'"
echo " -     'db.$COLLECTION_NAME.find().limit(3)' = Target Shard Query, returns the first 3 elements of
    the collection '$COLLECTION_NAME'"
echo " -     'db.$COLLECTION_NAME.find( {sensor_ip_address:\"192.168.0.79\"} ).length()' = returns the
    number of elements of the collection '$COLLECTION_NAME' with the given sensor's ip address."
echo " -     'db.$COLLECTION_NAME.find( {\"data.ph\":1.45} ).limit(4)' = returns 4 docs that has the
    property 'data.ph' equals to '1.45'"
echo " -      db.$COLLECTION_NAME.dataSize() = returns the size of the collection in bytes"
echo " -   More details on how to use the mongo client shell at http://www.mongodb.org/display/DOCS/
    dbshell+Reference"

#mongo -h $MONGO_SERVER_IP $DATABASE_NAME | tee $MONGO_SERVER_IP-$MONGO_CLIENT_LOG

echo "\n* 9. Execute the scenarios defined"
mongo MONGO_SERVER_IP < experiment-client-scenarios.js | tee scenarios-execution.log
```

Listing A.24: Experiment Scenarios Implementation in Javascript

```
//Select the database NetBEAMS to be used
use netbeams

//Each collected data from the sersor devices are
//stored on separate collections. Show all of them
print("Show all the collections of collected data (sensor devices' data)")
show collections

print("Show the statistics of the collection for the collected data for the YSI Sonde device")
db.SondeDataContainer.stats()

print("Verify some of the values using command abstraction shortcuts")
db.SondeDataContainer.count()
db.SondeDataContainer.dataSize()
```

```

print("An example of a random Show one random sample from the database using the findOne method
db.SondeDataContainer.findOne()

//db.SondeDataContainern.find({'$where' => 'this.time.valid.getTime() != this.time.transaction.getTime()
    '},{:limit => 50}).sort(:time.valid, Mongo::DESCENDING)

//the month variable of date class in javascript is defined as [0-11]. Therefore, 10 is November.

print("Execute Scenario R1) search all the documents in a given date range. Count and Limit retrieval
    result")
db.SondeDataContainer.find( {"time.valid" : { $gte:new Date(2009,11,8) , $lt:new Date(2009,11,15) }} ).
    count()
db.SondeDataContainer.find( {"time.valid" : { $gte:new Date(2009,11,8) , $lt:new Date(2009,11,15) }} ).
    limit(3)
print ("Explain the search over the shards for R1)")
db.SondeDataContainer.find( {"time.valid" : { $gte:new Date(2009,11,8) , $lt:new Date(2009,11,15) }} ).
    explain()

print("Execute Scenario R2) search all the documents for a given sensor device by IP address. Count and
    Limit retrieval result")
db.SondeDataContainer.find( {"sensor.ip_address":"192.168.0.102"} ).count()
db.SondeDataContainer.find( {"sensor.ip_address":"192.168.0.102"} ).limit(3)
print("Explain the search over the shards for R2")
db.SondeDataContainer.find( {"sensor.ip_address":"192.168.0.102"} ).explain()

print("Execute Scenario R3) search all the documents that has some observation values. Salinity = .01,
    Water Temperature = 46.47")
db.SondeDataContainer.find( {"observation.Salinity":0.01, "observation.WaterTemperature":46.47} ).count()
db.SondeDataContainer.find( {"observation.Salinity":0.01, "observation.WaterTemperature":46.47} ).limit(3)
print ("Explain the search over the shards for R3")
db.SondeDataContainer.find( {"observation.Salinity":0.01, "observation.WaterTemperature":46.47} ).explain
    ()

print("Execute Scenario U1) update all the documents for a given date December 12, by annotating them
    using tags")
db.SondeDataContainer.ensureIndex( { "tag":1 } )
print ("Number of Observations from December 12")
db.SondeDataContainer.find( {"time.valid" : { "$gte":new Date(2009,11,12) , "$lt":new Date(2009,11,13) }})
    .count()
print ("How many with the tag 'oil spill'")
db.SondeDataContainer.find( { "tag": "oil spill" } ).count()
db.SondeDataContainer.update( {"time.valid" : { "$gte":new Date(2009,11,12) , "$lt":new Date(2009,11,13)
    }} , {"$set": {"tag": "oil spill"}} , false, true)
print ("After annotating, how many documents have the tag?")
db.SondeDataContainer.find( { "tag": "oil spill" } ).count()
print ("Show an annotated document with the tag 'oil spil'")
db.SondeDataContainer.find( { "tag": "oil spill" } ).limit(1)

print("Execute Scenario D1) deleting all documents with for a given day (say Dec 13)")
db.SondeDataContainer.find( {"time.valid" : { $gte:new Date(2009,11,13) , $lt:new Date(2009,11,14) }} ).
    count()
db.SondeDataContainer.remove( {"time.valid" : { $gte:new Date(2009,11,13) , $lt:new Date(2009,11,14) }} )
print("How many documents are there after deleting?")
db.SondeDataContainer.find( {"time.valid" : { $gte:new Date(2009,11,13) , $lt:new Date(2009,11,14) }} ).
    count()
```

Listing A.25: Data Export to from mongoDB to OPeNDAP Format Python Script

```
# Copyright 2009 Marcello de Sales (marcello.sales@gmail.com).
# Converts the collected sensor data from the mongoDB to the OPEnDAP format
```

```

import pymongo
import sys

EXPORT_FILE_NAME="exported-by-python"

# Prints the header similar to the OPEnDAP format in the given file f,
# with the first line identifying the dataset, and the second one the
# columns of the measurements.
def printHeader(f):
  f.write('Dataset: ntb_YSI_PUF.dsp\n')
  f.write('YSI_REALTIME_CSV.Month,YSI_REALTIME_CSV.Day,YSI_REALTIME_CSV.Year,YSI_REALTIME_CSV.Hour,
      YSI_REALTIME_CSV.Min,YSI_REALTIME_CSV.Sec,YSI_REALTIME_CSV.WaterTemperature,YSI_REALTIME_CSV.
      SpecificConductivity,YSI_REALTIME_CSV.Conductivity,YSI_REALTIME_CSV.Resistivity,YSI_REALTIME_CSV.
      Salinity,YSI_REALTIME_CSV.Pressure,YSI_REALTIME_CSV.Depth,YSI_REALTIME_CSV.pH,YSI_REALTIME_CSV.pHmV,
      YSI_REALTIME_CSV.Turbidity,YSI_REALTIME_CSV.ODOSaturation,YSI_REALTIME_CSV.ODO,YSI_REALTIME_CSV.
      Battery\n')

# Prints given ysiData returned from mongoDB, in a dict format, in to the file
# f, following the format of comma-separated values.
def printRows(f, ysiData):
  f.write(ysiData["time"]["valid"].strftime("%m,%d,%Y,%H,%M,%S,"))
  f.write(str(ysiData["observation"]["WaterTemperature"]))
  f.write(",")
  f.write(str(ysiData["observation"]["SpecificConductivity"]))
  f.write(",")
  f.write(str(ysiData["observation"]["Conductivity"]))
  f.write(",")
  f.write(str(ysiData["observation"]["Resistivity"]))
  f.write(",")
  f.write(str(ysiData["observation"]["Salinity"]))
  f.write(",")
  f.write(str(ysiData["observation"]["Pressure"]))
  f.write(",")
  f.write(str(ysiData["observation"]["Depth"]))
  f.write(",")
  f.write(str(ysiData["observation"]["pH"]))
  f.write(",")
  f.write(str(ysiData["observation"]["pHmV"]))
  f.write(",")
  f.write(str(ysiData["observation"]["Turbidity"]))
  f.write(",")
  f.write(str(ysiData["observation"]["ODOSaturation"]))
  f.write(",")
  f.write(str(ysiData["observation"]["ODO"]))
  f.write(",")
  f.write(str(ysiData["observation"]["Battery"]))
  f.write("\n")

# processes the export request using the given ipAddress on the numerical
# portNumber, for the optional query supplied.
def processExport(ipAddress, portNumber, query):

  con = pymongo.Connection(ipAddress, portNumber)
  db = con.netbeams
  if (not query):
    collectedData = db.SondeDataContainer.find().limit(100)
  else:
    collectedData = db.SondeDataContainer.find(query).limit(100)
  if pymongo.cursor.Cursor.count(collectedData) > 0:
    f = open("/tmp/export-mongodb-to-sfbeams.csv", 'w')
    printHeader(f)
    for data in collectedData:
      printRows(f, data)
    f.close
  else:
    print "No records returned from NetBEAMS-mongoDB server"

# Main export method forwarded from the command-line execution with the
# given argument list. The first position must be the ip address of the
# NetBEAMS server, including the port number (ip:port). The second
# argument is an optional query in the JSON format for mongoDB to
# query the collection SondeDataContainer.
def exportOPEnDAP(argv):
  if len(argv) < 1:
    print "Please specify the server location (ip:port) and an optional query to export to OPEnDAP format"
  else:
    ipAddressPort = argv[0].split(':')
    query = None if len(argv) == 1 else argv[1]
    queryString = "" if not query else "Query supplied: ", query
    if len(ipAddressPort) == 2:
      print "Connecting to NetBEAMS database server at " , ipAddressPort[0] , " on port ", ipAddressPort
          [1] ,"... Query " , queryString
      processExport(ipAddressPort[0], int(ipAddressPort[1]), query)
    else:
      print "Please specify the server with the port number. E.g.: 10.1.25.13:20009, where 20009 is the
          port number"

if __name__ == "__main__":
    exportOPEnDAP(sys.argv[1:])
```

Listing A.26: The DSP YSI Sonde Data Java Handler

```
//
// This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference Implementation,
     vhudson-jaxb-ri-2.1-661
// See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation of the source schema.
// Generated on: 2009.03.04 at 06:20:22 PM PST
//


package org.netbeams.dsp.ysi;

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;

/**
 * <p>Java class for sondeDataType complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this class.
 *
 * <pre>
 * &lt;complexType name="sondeDataType">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="Date" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *         &lt;element name="Time" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *         &lt;element name="Temp" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *         &lt;element name="SpCond" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *         &lt;element name="Cond" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *          &lt;element name="Resist" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *          &lt;element name="Sal" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *          &lt;element name="Press" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *          &lt;element name="Depth" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *          &lt;element name="pH" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *          &lt;element name="phmV" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *          &lt;element name="ODOSat" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *          &lt;element name="ODOConc" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *          &lt;element name="Turbid" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *          &lt;element name="Battery" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *        &lt;/sequence>
 *      &lt;/restriction>
 *    &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
public class SondeDataType {

    public static final DateFormat dateTimeFormat = new SimpleDateFormat ("yyyy/MM/dd HH:mm:ss");

//    @XmlElement(name = "Date", required = true)
    protected Calendar samplingTimeStamp;
//    @XmlElement(name = "Temp", required = true)
    protected Float temp;
//    @XmlElement(name = "SpCond", required = true)
    protected Float spCond;
//    @XmlElement(name = "Cond", required = true)
    protected Float cond;
//    @XmlElement(name = "Resist", required = true)
    protected Float resist;
//    @XmlElement(name = "Sal", required = true)
    protected Float sal;
//    @XmlElement(name = "Press", required = true)
    protected Float press;
//    @XmlElement(name = "Depth", required = true)
    protected Float depth;
//    @XmlElement(name = "pH", required = true)
    protected Float ph;
//    @XmlElement(required = true)
    protected Float phmV;
//    @XmlElement(name = "ODOSat", required = true)
    protected Float odoSat;
//    @XmlElement(name = "ODOConc", required = true)
    protected Float odoConc;
//    @XmlElement(name = "Turbid", required = true)
    protected Float turbid;
//    @XmlElement(name = "Battery", required = true)
    protected Float battery;

    /**
     * Gets the value of the date property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Calendar getDateTime() {
        return this.samplingTimeStamp;
    }

    /**
     * @param format is an instance of a SimpleDateFormat. If you want to obtain
     * the information about time or date in separate, just build an instance with
     * the parameters you want. The instance used is the dateTime captured.
     * @return the string representation of the dateTime instance based on the formatter.
     */
    public String getDateAsString(SimpleDateFormat format) {
        return format.format(this.samplingTimeStamp.getTime());
    }

    public String getDateString() {
        return new SimpleDateFormat("yyyy/MM/dd").format(this.samplingTimeStamp.getTime());
    }

    public String getTimeString() {
        return new SimpleDateFormat("HH:mm:ss").format(this.samplingTimeStamp.getTime());
    }

    /**
     * Sets the value of the date property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setDateTime(String date, String time) {
        this.samplingTimeStamp = Calendar.getInstance();
        try {
            this.samplingTimeStamp.setTime(dateTimeFormat.parse(date + " " + time));
        } catch (ParseException e) {
        }
    }

    /**
     * Gets the value of the temp property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getTemp() {
        return temp;
    }

    /**
     * Sets the value of the temp property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setTemp(Float value) {
        this.temp = value;
    }

    /**
     * Gets the value of the spCond property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getSpCond() {
        return spCond;
    }

    /**
     * Sets the value of the spCond property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setSpCond(Float value) {
        this.spCond = value;
    }

    /**
     * Gets the value of the cond property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getCond() {
        return cond;
    }

    /**
     * Sets the value of the cond property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setCond(Float value) {
        this.cond = value;
    }

    /**
     * Gets the value of the resist property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getResist() {
        return resist;
    }

    /**
     * Sets the value of the resist property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setResist(Float value) {
        this.resist = value;
    }

    /**
     * Gets the value of the sal property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getSal() {
        return sal;
    }

    /**
     * Sets the value of the sal property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setSal(Float value) {
        this.sal = value;
    }

    /**
     * Gets the value of the press property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getPress() {
        return press;
    }

    /**
     * Sets the value of the press property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setPress(Float value) {
        this.press = value;
    }

    /**
     * Gets the value of the depth property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getDepth() {
        return depth;
    }

    /**
     * Sets the value of the depth property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setDepth(Float value) {
        this.depth = value;
    }

    /**
     * Gets the value of the ph property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getPH() {
        return ph;
    }

    /**
     * Sets the value of the ph property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setPH(Float value) {
        this.ph = value;
    }

    /**
     * Gets the value of the phmV property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getPhmV() {
        return phmV;
    }

    /**
     * Sets the value of the phmV property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setPhmV(Float value) {
        this.phmV = value;
    }

    /**
     * Gets the value of the odoSat property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getODOSat() {
        return odoSat;
    }

    /**
     * Sets the value of the odoSat property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setODOSat(Float value) {
        this.odoSat = value;
    }

    /**
     * Gets the value of the odoConc property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getODOConc() {
        return odoConc;
    }

    /**
     * Sets the value of the odoConc property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setODOConc(Float value) {
        this.odoConc = value;
    }

    /**
     * Gets the value of the turbid property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getTurbid() {
        return turbid;
    }

    /**
     * Sets the value of the turbid property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setTurbid(Float value) {
        this.turbid = value;
    }

    /**
     * Gets the value of the battery property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public Float getBattery() {
        return battery;
    }

    /**
     * Sets the value of the battery property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setBattery(Float value) {
        this.battery = value;
    }

    public String toXml() {
        StringBuilder builder = new StringBuilder();
        builder.append("<sondeData");
        if (this.samplingTimeStamp != null) {
            builder.append(" samplingTimeStamp=\"" + dateTimeFormat.format (this.samplingTimeStamp.getTime
                ()) + "\"");
        }
        builder.append(">");
        builder.append("<Temp>" + this.temp + "</Temp>");
        builder.append("<SpCond>" + this.spCond + "</SpCond>");
        builder.append("<Cond>" + this.cond + "</Cond>");
        builder.append("<Resist>" + this.resist + "</Resist>");
        builder.append("<Sal>" + this.sal + "</Sal>");
        builder.append("<Press>" + this.press + "</Press>");
        builder.append("<Depth>" + this.depth + "</Depth>");
        builder.append("<pH>" + this.ph + "</pH>");
        builder.append("<phmV>" + this.phmV + "</phmV>");
        builder.append("<ODOSat>" + this.odoSat + "</ODOSat>");
        builder.append("<ODOConc>" + this.odoConc + "</ODOConc>");
        builder.append("<Turbid>" + this.turbid + "</Turbid>");
        builder.append("<Battery>" + this.battery + "</Battery>");
        builder.append("</sondeData>");
        return builder.toString();
    }

}
```

Listing A.27: Random Test Data Java Generator

```
package org.netbeams.dsp.ysi;
```

```java

import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;

/**
 * Generates random test data. The number of data must be specified during the creation of an instance of
     this class.
 *
 * @author Teresa L. Johnson <gamma.particle@gmail.com> (Until revision 444)
 * @author Marcello de Sales <marcello.sales@gmail.com> (From revision 444)
 *
 */
public class SondeTestData {

    private static final SimpleDateFormat DATE_FORMATTER = new SimpleDateFormat("yyyy/MM/dd");
    private static final SimpleDateFormat TIME_FORMATTER = new SimpleDateFormat("HH:mm:ss");

    private static final DecimalFormat ONE_DECIMAL_FORMATTER = new DecimalFormat("###0.0");
    private static final DecimalFormat TWO_DECIMALS_FORMATTER = new DecimalFormat("###0.00");
    private static final DecimalFormat THREE_DECIMALS_FORMATTER = new DecimalFormat("###0.000");

    private static final int CALENDARS_CACHE_SIZE = 15;
    private static final ArrayList<Calendar> calendarsCache = new ArrayList<Calendar>(CALENDARS_CACHE_SIZE
        );

    static {
        for (int i = 0; i < CALENDARS_CACHE_SIZE; i++) {
            Calendar cdr = Calendar.getInstance();
            cdr.set(Calendar.DAY_OF_MONTH, (int) (Math.random() * (30 - 1)) + 1);
            cdr.set(Calendar.HOUR_OF_DAY, (int) (Math.random() * 23));
            cdr.set(Calendar.MINUTE, (int) (Math.random() * 59));
            cdr.set(Calendar.SECOND, (int) (Math.random() * 59));
            long val1 = cdr.getTimeInMillis();

            cdr.set(Calendar.HOUR_OF_DAY, (int) (Math.random() * 23));
            cdr.set(Calendar.MINUTE, (int) (Math.random() * 59));
            cdr.set(Calendar.SECOND, (int) (Math.random() * 59));
            long val2 = cdr.getTimeInMillis();

            long randomTS = (long) (Math.random() * (val2 - val1)) + val1;
            cdr.setTime(new Date(randomTS));
            calendarsCache.add(cdr);
        }
    }

    /**
     * Creates a new test data with the given number of sonde data to be in the container.
     *
     * @param numberOfSondeData is the number of data to be in the container.
     */
    private SondeTestData() {

    }

    private static Calendar getRandomCalendar() {
        int calendarIndex = (int) (Math.random() * (calendarsCache.size() - 1));
        return calendarsCache.get(calendarIndex);
    }

    /**
     * @param numberOfRandomSondeDataTypes is the number of Sonde Data to be in the container.
     * @return a SondeDataContainer with a random list of SondeDataTypes containing the given number of
         random Sonde
     *          Data Types.
     */
    public static SondeDataContainer generateRandomSondeDataContainer(int numberOfRandomSondeDataTypes) {
        SondeDataContainer container = new SondeDataContainer();
        container.sondeData = new ArrayList<SondeDataType>(numberOfRandomSondeDataTypes);
        container.sondeData.addAll(generateRandomSondeData(numberOfRandomSondeDataTypes));
        return container;
    }

    /**
     * @param numberOfRandomSondeData is the number of random sonde data types to be generated.
     * @return a random list of SondeDataType with the current date and time information.
     */
    public static List<SondeDataType> generateRandomSondeData(int numberOfRandomSondeData) {
        List<SondeDataType> randomSondeData = new ArrayList<SondeDataType>(numberOfRandomSondeData);
        for (int i = 0; i < numberOfRandomSondeData; i++) {
            randomSondeData.add(getRandomSondeData());
        }
        return randomSondeData;
    }

    /**
     * @return a random SondeDataType with the current date and time.
     */
    public static SondeDataType getRandomSondeData() {
        SondeDataType sdt = new SondeDataType();
        Calendar cal = getRandomCalendar();
        sdt.setDateTime(DATE_FORMATTER.format(cal.getTime()), TIME_FORMATTER.format(cal.getTime()));
        sdt.setTemp(Float.valueOf(TWO_DECIMALS_FORMATTER.format(100.69 * Math.random())));
        sdt.setSpCond(Float.valueOf(ONE_DECIMAL_FORMATTER.format(183.0 * Math.random())));
        sdt.setCond(Float.valueOf(ONE_DECIMAL_FORMATTER.format(175.0 * Math.random())));
        sdt.setResist(Float.valueOf(TWO_DECIMALS_FORMATTER.format(5704.66 * Math.random())));
        sdt.setSal(Float.valueOf(TWO_DECIMALS_FORMATTER.format(0.09 * Math.random())));
        sdt.setPress(Float.valueOf(THREE_DECIMALS_FORMATTER.format(1.999 * Math.random())));
        sdt.setDepth(Float.valueOf(THREE_DECIMALS_FORMATTER.format(2.999 * Math.random())));
        sdt.setPH(Float.valueOf(TWO_DECIMALS_FORMATTER.format(8.22 * Math.random())));
        sdt.setPhmV(Float.valueOf(ONE_DECIMAL_FORMATTER.format(-94.00 * Math.random())));
        sdt.setODOSat(Float.valueOf(ONE_DECIMAL_FORMATTER.format(111.70 * Math.random())));
        sdt.setODOConc(Float.valueOf(TWO_DECIMALS_FORMATTER.format(66.63 * Math.random())));
        sdt.setTurbid(Float.valueOf(ONE_DECIMAL_FORMATTER.format(0.30 * Math.random())));
        sdt.setBattery(Float.valueOf(ONE_DECIMAL_FORMATTER.format(10.10 * Math.random())));
        return sdt;
    }

    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        SondeDataContainer container = SondeTestData.generateRandomSondeDataContainer(10000);
        long end = System.currentTimeMillis();

        System.out.println("generated in " + (end - start) + " millis of size: " + container.getSondeData
            ().size());
        System.out.println();

        // Verify that the data can be retrieved (first 3).
        for (int i = 0; i < 3; i++) {
            System.out.print("Date: " + container.getSondeData().get(i).getDateString());
            System.out.print(" Time: " + container.getSondeData().get(i).getTimeString());
            System.out.print(" Temp: " + container.getSondeData().get(i).getTemp());
            System.out.print(" SpCond: " + container.getSondeData().get(i).getSpCond());
            System.out.print(" Cond: " + container.getSondeData().get(i).getCond());
            System.out.print(" Resist: " + container.getSondeData().get(i).getResist());
            System.out.print(" Sal: " + container.getSondeData().get(i).getSal());
            System.out.print(" Press: " + container.getSondeData().get(i).getPress());
            System.out.print(" Depth: " + container.getSondeData().get(i).getDepth());
            System.out.print(" pH: " + container.getSondeData().get(i).getPH());
            System.out.print(" pH mV: " + container.getSondeData().get(i).getPhmV());
            System.out.print(" ODO Sat: " + container.getSondeData().get(i).getODOSat());
            System.out.print(" ODO Cond: " + container.getSondeData().get(i).getODOConc());
            System.out.print(" Turbidity: " + container.getSondeData().get(i).getTurbid());
            System.out.println(" Battery: " + container.getSondeData().get(i).getBattery());
            System.out.println();
        }
        // print the last 3
        for (int i = container.getSondeData().size() - 3; i < container.getSondeData().size(); i++) {
            System.out.print("Date: " + container.getSondeData().get(i).getDateString());
            System.out.print(" Time: " + container.getSondeData().get(i).getTimeString());
            System.out.print(" Temp: " + container.getSondeData().get(i).getTemp());
            System.out.print(" SpCond: " + container.getSondeData().get(i).getSpCond());
            System.out.print(" Cond: " + container.getSondeData().get(i).getCond());
            System.out.print(" Resist: " + container.getSondeData().get(i).getResist());
            System.out.print(" Sal: " + container.getSondeData().get(i).getSal());
            System.out.print(" Press: " + container.getSondeData().get(i).getPress());
            System.out.print(" Depth: " + container.getSondeData().get(i).getDepth());
            System.out.print(" pH: " + container.getSondeData().get(i).getPH());
            System.out.print(" pH mV: " + container.getSondeData().get(i).getPhmV());
            System.out.print(" ODO Sat: " + container.getSondeData().get(i).getODOSat());
            System.out.print(" ODO Cond: " + container.getSondeData().get(i).getODOConc());
            System.out.print(" Turbidity: " + container.getSondeData().get(i).getTurbid());
            System.out.println(" Battery: " + container.getSondeData().get(i).getBattery());
            System.out.println();
        }
    }
}
```

Listing A.28: Inserting Random Data to MongoDB Java Experiment

```
package org.netbeans.dsp.persistence.controller;

import java.net.UnknownHostException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

import org.netbeans.dsp.message.ComponentIdentifier;
import org.netbeans.dsp.message.DSPMessagesFactory;
import org.netbeans.dsp.message.Header;
import org.netbeans.dsp.message.Message;
import org.netbeans.dsp.message.MessageContent;
import org.netbeans.dsp.persistence.model.component.data.PersistentMessageUnit;
import org.netbeans.dsp.persistence.model.location.SensorLocation;
import org.netbeans.dsp.ysi.SondeDataContainer;
import org.netbeans.dsp.ysi.SondeDataType;
import org.netbeans.dsp.ysi.SondeTestData;

import com.commsen.stopwatch.Report;
import com.commsen.stopwatch.Stopwatch;
import com.mongodb.BasicDBObject;
import com.mongodb.DBCollection;
```

```java
import com.mongodb.MongoException;

/**
 * The experiment to transfer DSP Messages to mongoDB. It uses random data of DSP Message Contents.
 *
 * @author Marcello de Sales (marcello.sales@gmail.com)
 *
 */
public class DSPMessageToMongoDBExperiment {

    private static final SimpleDateFormat DATE_FORMATTER = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss:SS");
    /**
     * Sleep interval for garbage collection
     */
    private static long fSLEEP_INTERVAL = 100;

    private static final Set<SensorLocation> randomLocationsCache = new HashSet<SensorLocation>();

    private static Set<PersistentMessageUnit> persistentUnits;

    private static boolean inWriteLock = true;

    private static int insertedDocuments = 0;

    public static int getNumberOfInsertedDocuments() {
        return insertedDocuments;
    }

    public static boolean isInWriteLock() {
        return inWriteLock;
    }

    /**
     * @param startMemoryUse is the number of bytes previously measured
     * @return the amount of memory in bytes between the start number measurement and the current.
     */
    public static long getMemoryUsedSinceGivenMemorySample(long startMemoryUse) {
        float endMemoryUse = getMemoryUse();
        return Math.round((endMemoryUse - startMemoryUse) / 1024);
    }

    /**
     * @param start time in milliseconds
     * @param end time in milliseconds
     * @return the difference between the end and start values formatted in string
     */
    public static String getTimeDifference(long start, long end) {
        long diff = end - start;
        return (float) (((float) diff / (float) 1000) / 60) + " minutes (" + diff + " milliseconds) at "
                + DATE_FORMATTER.format(new Date(end));
    }

    /**
     * @return the values of bytes used by the JVM
     */
    private static long getMemoryUse() {
        putOutTheGarbage();
        long totalMemory = Runtime.getRuntime().totalMemory();

        putOutTheGarbage();
        long freeMemory = Runtime.getRuntime().freeMemory();

        return (totalMemory - freeMemory);
    }

    /**
     * Try to force the garbage collection to clean up the memory
     */
    private static void putOutTheGarbage() {
        collectGarbage();
        collectGarbage();
    }

    /**
     * Try to call garbage collection
     */
    private static void collectGarbage() {
        try {
            System.gc();
            Thread.currentThread().sleep(fSLEEP_INTERVAL);
            System.runFinalization();
            Thread.currentThread().sleep(fSLEEP_INTERVAL);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }

    /**
     * Generates a given number of persistent message units
     *
     * @param numberItems is the number of persistent messages unit. Must be bigger than 0
     * @return the set of unique persistent message Unit items
     */
    private static Set<PersistentMessageUnit> generateRandomMessages(int numberItems) {

        System.out.println("Producing " + numberItems + " random PersistentUnits with one DSP Message with
             a random "
               + " sample frequence from the YSI Sonde");

        persistentUnits = new HashSet<PersistentMessageUnit>(numberItems);
        final Calendar TRANSACTION_TIME = Calendar.getInstance();
        for (int i = 0; i < numberItems; i++) {
            // build a random sensor location with fix GPS coordination from the RTC pier.
            SensorLocation.Builder locationBuider = new SensorLocation.Builder();
            locationBuider.setIpAddress("192.168.0." + ((int) (Math.random() * 253) + 2));
            locationBuider.setLatitude(Double.valueOf(37.89155));
            locationBuider.setLongitude(Double.valueOf(-122.4464));
            SensorLocation location = locationBuider.build();
            randomLocationsCache.add(location);

            // build the producer identifier
            ComponentIdentifier producerId = DSPMessagesFactory.INSTANCE.makeDSPComponentIdentifier("
                experiment_comp",
                    location.getIpAddress(), "sonde-data-type");

            // build the message header
            Header header = DSPMessagesFactory.INSTANCE.makeDSPMessageHeader(null, producerId, null);

            // build just one sample observation for the message
            SondeDataContainer container = SondeTestData.generateRandomSondeDataContainer(1);
            Message msg = DSPMessagesFactory.INSTANCE.makeDSPMeasureMessage(header, container);

            // persistence unit
            PersistentMessageUnit pmu = new PersistentMessageUnit(msg, locationBuider.build(),
                TRANSACTION_TIME);
            persistentUnits.add(pmu);
        }
        return persistentUnits;
    }

    /**
     * Inserts the DSP Message Content into the mongoDB as it is extracted and converted from the given
     * PersistentMessageUnit.
     *
     * @param tranMsg is the PersistentMessageUnit containing information about the sensor location and
         the message.
     * @throws UnknownHostException
     * @throws MongoException
     */
    public static void insertPersistentUnitMessageContents(Set<PersistentMessageUnit> tranMsgs)
            throws UnknownHostException, MongoException {

        DSPMongoCRUDService.INSTANCE.getNetbeamMongoDb().requestStart();
        for (PersistentMessageUnit tranMsg : tranMsgs) {
            DBCollection netbeamsDbCollection = DSPMongoCRUDService.INSTANCE.getPersistenceStorage(tranMsg
                );
            MessageContent messageContent = tranMsg.getDspMessage().getBody().getAny();
            SondeDataContainer sondeContainer = (SondeDataContainer) messageContent;
            SondeDataType sondeData = sondeContainer.getSondeData().get(0);
            // build the document value
            BasicDBObject docValue = DSPMongoCRUDService.INSTANCE.buildValueSegment(sondeData);
            // extract the fact time from the message, adding to the key
            long factTime = sondeData.getDateTime().getTimeInMillis();
            // build the document key
            BasicDBObject docKey = DSPMongoCRUDService.INSTANCE.buildKeySegment(tranMsg, factTime,
                docValue);
            // insert the final collection
            netbeamsDbCollection.insert(docKey);
            ++insertedDocuments;
        }
        inWriteLock = false;
        DSPMongoCRUDService.INSTANCE.getNetbeamMongoDb().requestDone();
    }

    public static void main(String[] args) throws UnknownHostException, MongoException,
         InterruptedException {
        if (args.length < 2) {
            throw new IllegalArgumentException(
                    "You need to provide the server IP address and number of netbeams samples to generate:
                          "
                            + "\nExample: DSPMongoCRUDService 127.0.0.1 5");
        }

        putOutTheGarbage();
        Stopwatch.setActive(true);
        long watchExperimentId = Stopwatch.start("Complete Experiene", "entire-experience");

        // the needed 3 arguments
        final String SERVER_IP_ADDRESS = args[0].split(":")[0];
        ;
        final int SERVER_PORT_NUMBER = Integer.parseInt(args[0].split(":")[1]);
        final int NUMBER_SAMPLES = Integer.valueOf(args[1]);
        String PROPERTIES_LIST = null;
        if (args.length == 3) {
            PROPERTIES_LIST = args[2];
        }

        long startMemoryUse = getMemoryUse();
        System.out.println("Experiment started at " + DATE_FORMATTER.format(new Date()));
        long start = System.currentTimeMillis();
        System.out.println("Starting to generate " + NUMBER_SAMPLES + " sonde samples at "
                + DATE_FORMATTER.format(new Date(start)) + "; MEM USED: " + startMemoryUse);

        Thread status = new Thread("Messages-Generator") {
            public void run() {
                int i = 0;
                int min = 0;
                System.out.println();
                while (persistentUnits == null || persistentUnits.size() < NUMBER_SAMPLES) {
                    try {
                        i++;
                        if (i < 60) {
                            System.out.print(".");
                        }
                        if (i % 20 == 0) {
                            System.out.println();
                            System.out.println("Generated Messages = " + persistentUnits.size());
                            System.out.println();
                        }
                        if (i == 60) {
                            i = 0;
                            ++min;
                            System.out.println();
                            System.out.println(" " + min + " min #= " + persistentUnits.size());
                            System.out.println();
                        }
                        this.sleep(1000);
                    } catch (InterruptedException e) {

                    }
                }
            }
        };
        status.start();
        long watchTransaction = Stopwatch.start("Time for creating random PersistentMessage Units", "all-
             messages");
        Set<PersistentMessageUnit> messageUnits = generateRandomMessages(NUMBER_SAMPLES);
        putOutTheGarbage();
        Stopwatch.stop(watchTransaction);
        System.out.println("Persistent Message Units: " + persistentUnits.size());

        status = new Thread("Status-Initializer") {
            public void run() {
                int i = 0;
                int min = 0;
                while (!DSPMongoCRUDService.hasFinished()) {
                    try {
                        i++;
                        if (i < 60) {
                            System.out.print(".");
                        }
                        if (i == 60) {
                            i = 0;
                            ++min;
                            System.out.println();
                            System.out.println(" " + min + " min on Initializing Persistence Service");
                            System.out.println();
                        }
                        this.sleep(1000);
                    } catch (InterruptedException e) {

                    }
                }
            }
        };
        status.start();
        watchTransaction = Stopwatch.start("Time for initializing the mongo service", "mongo-service");
        DSPMongoCRUDService.INSTANCE.initialize(SERVER_IP_ADDRESS, SERVER_PORT_NUMBER,
             randomLocationsCache,
                PROPERTIES_LIST);
        Stopwatch.stop(watchTransaction);

        long end = System.currentTimeMillis();
        float result = getMemoryUsedSinceGivenMemorySample(startMemoryUse);
        System.out.println("Finished Generating " + NUMBER_SAMPLES + " sonde samples on "
                + getTimeDifference(start, end) + " consuming ~" + result + "Kb");

        status = new Thread("Status-Insertion") {
            public void run() {
                int i = 0;
                int min = 0;
                while (isInWriteLock()) {
                    try {
                        i++;
                        if (i < 60) {
                            System.out.print(".");
                        }
                        if (i % 20 == 0) {
                            System.out.println();
                            System.out.println("Inserted Units = " + getNumberOfInsertedDocuments());
                            System.out.println();
                        }
                        if (i == 60) {
                            i = 0;
                            ++min;
                            System.out.println();
                            System.out.println(" " + min + " min on Inserting Data");
                            System.out.println();
                        }
                        this.sleep(1000);
                    } catch (InterruptedException e) {

                    }
                }
            }
        };
        status.start();
        try {
            watchTransaction = Stopwatch.start("Time for inserting all PersistentMessage Units", "insert-
                 all");
            insertPersistentUnitMessageContents(messageUnits);
            Stopwatch.stop(watchTransaction);
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (MongoException e) {
            e.printStackTrace();
        }

        Stopwatch.stop(watchExperimentId);
        end = System.currentTimeMillis();
        result = getMemoryUsedSinceGivenMemorySample(startMemoryUse);
        System.out.println("Experiment finished saving " + messageUnits.size() + " sonde samples on
             MongoDB on "
            + getTimeDifference(start, end) + " consuming ~" + result + " Kb");

        System.out.println("\nProcessing times for sections");
        for (Report report : Stopwatch.getAllReports()) {
            System.out.println(report);
        }
    }
}
```

A.5.1 Experiment Execution Output Logs

Listing A.29: Execution of Shards Setup Log

```
marcello@netbeams-mongo-dev02:~/development/workspaces/netbeams/persistence$ MongoDB shell version: 1.2.0
url: test
connecting to: test
Sat Dec 12 20:58:41 connection accepted from 127.0.0.1:51182 #1
type "help" for help
> admin = db.getSisterDB("admin")
admin
> config = db.getSisterDB("config")
config
>
> admin.runCommand( { addshard: "192.168.1.2:20001" } )
Sat Dec 12 20:58:41 Request::process ns: admin.$cmd msg id:1709579090 attempt: 0
Sat Dec 12 20:58:41 single query: admin.$cmd  { addshard: "192.168.1.2:20001" }  ntoreturn: -1
{ "msg" : "already exists", "ok" : 0, "errmsg" : "" }
> admin.runCommand( { addshard: "192.168.1.2:20002" } )
Sat Dec 12 20:58:41 Request::process ns: admin.$cmd msg id:1709579091 attempt: 0
Sat Dec 12 20:58:41 single query: admin.$cmd  { addshard: "192.168.1.2:20002" }  ntoreturn: -1
Sat Dec 12 20:58:41 creating new connection for pool to:192.168.1.2:20002
{ "added" : "192.168.1.2:20002", "ok" : 1 }
> admin.runCommand( { addshard: "localhost:20001", allowLocal: true } )
Sat Dec 12 20:58:42 Request::process ns: admin.$cmd msg id:1709579092 attempt: 0
Sat Dec 12 20:58:42 single query: admin.$cmd  { addshard: "localhost:20001", allowLocal: true }  ntoreturn
    : -1
Sat Dec 12 20:58:42 creating new connection for pool to:localhost:20001
Sat Dec 12 20:58:42 connection accepted from 127.0.0.1:52822 #1
{ "added" : "localhost:20001", "ok" : 1 }
>
> admin.runCommand( { listshards:1 } )
Sat Dec 12 20:58:42 Request::process ns: admin.$cmd msg id:1709579093 attempt: 0
Sat Dec 12 20:58:42 single query: admin.$cmd  { listshards: 1.0 }  ntoreturn: -1
{
  "shards" : [
    {
      "_id" : ObjectId("4b24740682b9fcc8b1713769"),
      "host" : "192.168.1.2:20001"
    },
    {
      "_id" : ObjectId("4b247481a9a1fb6a9d67dea8"),
      "host" : "192.168.1.2:20002"
    },
    {
      "_id" : ObjectId("4b247482a9a1fb6a9d67dea9"),
      "host" : "localhost:20001"
    }
  ],
  "ok" : 1
}
```

```
>
> admin.runCommand( { enablesharding: "netbeams" } )
Sat Dec 12 20:58:42 Request::process ns: admin.$cmd msg id:1709579094 attempt: 0
Sat Dec 12 20:58:42 single query: admin.$cmd  { enablesharding: "netbeams" }  ntoreturn: -1
Sat Dec 12 20:58:42 couldn't find database [netbeams] in config db
Sat Dec 12 20:58:42    put [netbeams] on: 192.168.1.2:20002
{ "ok" : 1 }
<collection: "netbeams.SondeDataContainer", key: { "observation.pH" : 1} } )
Sat Dec 12 20:58:42 Request::process ns: admin.$cmd msg id:1709579095 attempt: 0
Sat Dec 12 20:58:42 single query: admin.$cmd  { shardcollection: "netbeams.SondeDataContainer", key: { observation.pH: 1.0 } }  ntoreturn: -1
Sat Dec 12 20:58:42 no chunks for:netbeams.SondeDataContainer so creating first: shard  ns:netbeams.SondeDataContainer shard: 192.168.1.2:20002 min: { observation.pH: MinKey } max: { observation.pH: MaxKey }
Sat Dec 12 20:58:42 building new index on { _id: ObjId(000000000000000000000000) } for config.chunks...
Sat Dec 12 20:58:42 Buildindex config.chunks idxNo:0 { name: "_id_", ns: "config.chunks", key: { _id: ObjId(000000000000000000000000) } }
Sat Dec 12 20:58:42 done for 0 records 0.001secs
Sat Dec 12 20:58:42 before: { _id: ObjId(4b247482981e648053eebfed) }  shard  ns:netbeams.SondeDataContainer shard: 192.168.1.2:20002 min: { observation.pH: MinKey } max: { observation.pH: MaxKey }
Sat Dec 12 20:58:42 after : { _id: ObjId(4b247482981e648053eebfed) }  shard  ns:netbeams.SondeDataContainer shard: 192.168.1.2:20002 min: { observation.pH: MinKey } max: { observation.pH: MaxKey }
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579096 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579097 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
{ "collectionsharded" : "netbeams.SondeDataContainer", "ok" : 1 }
>
> netbeams = db.getSisterDB("netbeams")
netbeams
> netbeams.SondeDataContainer.ensureIndex( { "message_id":1 } )
Sat Dec 12 20:58:42 couldn't find database [test] in config db
Sat Dec 12 20:58:42    put [test] on: 192.168.1.2:20002
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579098 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
> netbeams.SondeDataContainer.ensureIndex( { "sensor.ip_address":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579099 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579100 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
> netbeams.SondeDataContainer.ensureIndex( { "sensor.location.latitude":1 Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579101 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
} )
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579102 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579103 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579104 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
> netbeams.SondeDataContainer.ensureIndex( { "sensor.locationSat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579105 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
<er.ensureIndex( { "sensor.location.longitude":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579106 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579107 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
> netbeams.SondeDataContainer.ensureIndex( { "time.valid":1 } )
> netbeams.SondeDataContainer.ensureIndex( { "time.transaSat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579108 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579109 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579110 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
ction":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579111 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579112 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579113 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
<er.ensureIndex( { "observation.WaterTemperature":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579114 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579115 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579116 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579117 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
<er.ensureIndex( { "observation.SpecificConductivity":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579118 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579119 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
> netbeams.SondeDataContainer.ensureIndex( { "observation.CoSat Dec 12 20:58:42 Request::process ns:
     netbeams.system.indexes msg id:1709579120 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
nductivity":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579121 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579122 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579123 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
> netbeams.SondeDataContainer.ensureIndex( { "observation.Resistivity":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579124 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579125 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
> netbeams.SondeDataContainer.ensureIndex( { "observation.Salinity":1 } )
> netbeams.SondeDataContainer.ensureIndex( { "observation.Pressure":1 Sat Dec 12 20:58:42 Request::process
      ns: netbeams.system.indexes msg id:1709579126 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579127 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579128 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579129 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579130 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579131 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579132 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
} )
> netbeams.SondeDataContainer.ensureIndex( { "observation.Depth":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579133 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579134 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579135 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579136 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579137 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
> netbeams.SondeDataContainer.ensureIndex( { "observation.pH":1 } )
> netbeams.SondeDataContainer.ensureIndex( { "observation.pHmV":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579138 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579139 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579140 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579141 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
> netbeams.SondeDataContainer.ensureIndex( { "observation.Turbidity":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579142 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579143 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579144 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
<er.ensureIndex( { "observation.ODOSaturation":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579145 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579146 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579147 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
> netbeams.SondeDataContainer.ensureIndex( { "observation.ODO":1 } )
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579148 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579149 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
Sat Dec 12 20:58:42 Request::process ns: netbeams.system.indexes msg id:1709579150 attempt: 0
Sat Dec 12 20:58:42  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 20:58:42 Request::process ns: netbeams.$cmd msg id:1709579151 attempt: 0
Sat Dec 12 20:58:42 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
> netbeams.SondeDataContainer.ensureIndex( { "observation.Battery":1 } )
Sat Dec 12 20:58:42 Request::process ns: test.$cmd msg id:1709579152 attempt: 0
Sat Dec 12 20:58:42 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: -1
>
> config.chunks.find()
Sat Dec 12 20:58:42 Request::process ns: config.chunks msg id:1709579153 attempt: 0
Sat Dec 12 20:58:42 single query: config.chunks  {}  ntoreturn: 0
creating WriteBackListener for: localhost:10000
Sat Dec 12 20:58:42 creating new connection for pool to:localhost:10000
Sat Dec 12 20:58:42 connection accepted from 127.0.0.1:60571 #2
Sat Dec 12 20:58:42 end connection 127.0.0.1:51182
{ "lastmod" : { "t" : 1260680322000, "i" : 1 }, "ns" : "netbeams.SondeDataContainer", "min" : { "
    observation" : { "pH" : { $minKey : 1 } } }, "minDotted" : { "observation.pH" : { $minKey : 1 } }, "
    max" : { "observation" : { "pH" : { $maxKey : 1 } } }, "maxDotted" : { "observation.pH" : { $maxKey :
    1 } }, "shard" : "192.168.1.2:20002", "_id" : ObjectId("4b247482981e648053eebfed") }
>
> bye
```

Listing A.30: Main Client Experiment Execution Log

```
####### NetBEAMS Experiments − Persistence on MongoDB ########

```

```
########### Netbeams to MongoDB Experiment  #############

# 1. Cleaning any existing MongoDB data at 'data'


# 2. Starting MongoDB Server with 2 local shards (1 remote)... NetBEAMS data will be saved at dir 'data'


# 3. Ready to run Java experiment with 483840 samples

Experiment started at 12/12/2009 02:04:26:777
Starting to generate 483840 sonde samples at 12/12/2009 02:04:26:782; MEM USED: 1777272

.Producing 483840 random PersistentUnits with one DSP Message with a random  sample frequence from the YSI
     Sonde
...................
Generated Messages = 173176

....................
Generated Messages = 373506

..........Persistent Message Units: 483840
.Finished Generating 483840 sonde samples on 0.9838167 minutes (59029 milliseconds) at 12/12/2009
    02:05:25:811 consuming ~553210.0Kb
....................
Inserted Units = 15886

....................
Inserted Units = 30447

...................
Inserted Units = 49684


 1 min on Inserting Data

....................
Inserted Units = 66326

....................
Inserted Units = 77562

...................
Inserted Units = 96924


 2 min on Inserting Data

....................
Inserted Units = 114967

....................
Inserted Units = 133254

...................
Inserted Units = 136228


 3 min on Inserting Data

....................
Inserted Units = 140756

```

```
....................
Inserted Units = 153353

...................
Inserted Units = 163442


 4 min on Inserting Data

....................
Inserted Units = 174926

....................
Inserted Units = 187694

...................
Inserted Units = 204208


 5 min on Inserting Data

....................
Inserted Units = 220089

....................
Inserted Units = 234139

...................
Inserted Units = 248243


 6 min on Inserting Data

....................
Inserted Units = 269302

....................
Inserted Units = 289526

...................
Inserted Units = 306940


 7 min on Inserting Data

....................
Inserted Units = 317020

....................
Inserted Units = 332196

...................
Inserted Units = 349706


 8 min on Inserting Data

....................
Inserted Units = 371025

....................
Inserted Units = 384756

```

```
...................
Inserted Units = 396657


 9 min on Inserting Data

....................
Inserted Units = 415808

....................
Inserted Units = 435336

...................
Inserted Units = 454749


 10 min on Inserting Data

....................
Inserted Units = 464837

....................
Inserted Units = 477627

.......Experiment finished saving 483840 sonde samples on MongoDB on 11.955417 minutes (717325
    milliseconds) at 12/12/2009 02:16:24:107 consuming ~572816.0 Kb

Processing times for sections
Group="Complete Experiene"  Label="entire-experience"  Count=1  Min=718255ms  Avg=718255ms  Max=718255ms
    Total=718255ms
Group="Time for inserting all PersistentMessage Units"  Label="insert-all"  Count=1  Min=648462ms  Avg
    =648462ms  Max=648462ms  Total=648462ms
Group="Time for creating random PersistentMessage Units"  Label="all-messages"  Count=1  Min=59009ms  Avg
    =59009ms  Max=59009ms  Total=59009ms
Group="Time for initializing the mongo service"  Label="mongo-service"  Count=1  Min=0ms  Avg=0ms  Max=0ms
      Total=0ms

* 4. Exporting the generated mongoDB data in CSV file format

Executing the export command
mongoexport -h 192.168.1.10:27017 -d netbeams -c SondeDataContainer --query "{"time.valid" : { :new Date
    (2009,11,1) , :new Date(2009,11,8) }" --csv -f "_id,sensor.ip_address,time.valid,observation.
    WaterTemperature,observation.SpecificConductivity,observation.Conductivity,observation.Resistivity,
    observation.Salinity,observation.Pressure,observation.Depth,observation.pH,observation.pHmV,
    observation.Turbidity,observation.ODOSaturation,observation.ODO,observation.Battery" -o logs/
    experiment-483840-data-exported-20091212-020424.csv


* 5. Experiments Results on the following logs:
- mongoDB Server log output: logs/experiment-483840-mongodb-server-status-20091212-020424.log
- mongoDB Client log output: logs/experiment-483840-mongodb-client-status-20091212-020424.log
- NetBEAMS to mongoDB data transfer log output: logs/experiment-483840-netbeams-to-mongodb
    -20091212-020424.log (after running the client)
- mongoDB data exported in CSV format: logs/experiment-483840-data-exported-20091212-020424.csv

* 6. mongoDB data dir 'data' size after experiments...


* 7. Accessing the persisted sensor data after the experiments...

 - Through the mongoDB Client using via command line "mongo netbeams"
 - Through the mongoDB REST Web Services client using HTTP GET Requests http://192.168.1.10:27017:28017/
     netbeams/SondeDataContainer/?limit=-5
 - Through Futon2Mongo, the check the web client at http://127.0.0.1:4567/_utils/index.html

* 7. Running the mongDB client after the experiments...

 - The database name is 'netbeams'. The collection name is 'SondeDataContainer'
 - Type 'show collections' to show all the collections in the current database
 - Type 'use netbeams' to change to that database.
 - Type 'db.SondeDataContainer.*' to issue a command to the collection 'SondeDataContainer'
 - Ex: 'db.SondeDataContainer.count()' = Global Shard Query, returns the number of docs on the collection
      'SondeDataContainer'
 -     'db.SondeDataContainer.findOne()' = Target Shard Query, returns the first element of the collection
       'SondeDataContainer'
 -     'db.SondeDataContainer.find().limit(3)' = Target Shard Query, returns the first 3 elements of the
      collection 'SondeDataContainer'
 -     'db.SondeDataContainer.find( {sensor_ip_address:"192.168.0.79"} ).length()' = returns the number of
       elements of the collection 'SondeDataContainer' with the given sensor's ip address.
 -     'db.SondeDataContainer.find( {"data.ph":1.45} ).limit(4)' = returns 4 docs that has the property '
      data.ph' equals to '1.45'
 -      db.SondeDataContainer.dataSize() = returns the size of the collection in bytes
 -    More details on how to use the mongo client shell at http://www.mongodb.org/display/DOCS/dbshell+
      Reference
```

Listing A.31: RUD Scenarios Execution Log

```
marcello@netbeams-dev:~/workspaces/netbeams/versions/v2/persistence$ mongo 192.168.1.10 < experiment-
    client-scenarios.js | tee scenarios-execution.log
MongoDB shell version: 1.1.4
url: 192.168.1.10
connecting to: 192.168.1.10/test
type "help" for help
> //Select the database NetBEAMS to be used
> use netbeams
switched to db netbeams
>
> //Each collected data from the sersor devices are
> //stored on separate collections. Show all of them

Show all the collections of collected data (sensor devices' data)
> show collections
SondeDataContainer
system.indexes

Show the statistics of the collection for the collected data for the YSI Sonde device
> db.SondeDataContainer.stats()
{
   "ns" : "netbeams.SondeDataContainer",
   "count" : 2422043,
   "size" : 1175737304,
   "storageSize" : 1416246240,
   "nindexes" : 39,
   "ok" : 1
}

Verify some of the values using command abstraction shortcuts
> db.SondeDataContainer.count()
2422043
> db.SondeDataContainer.dataSize()
1175737304
>

Show one random sample from the database using the findOne method
```

```
... db.SondeDataContainer.findOne()

{
  "_id" : ObjectId("e26f40072f68234b6af3d600"),
  "message_id" : "b405e634-fd4b-450c-9466-82dc0555ea06",
  "sensor" : {
    "ip_address" : "192.168.0.178",
    "location" : {
      "latitude" : 37.89155,
      "longitude" : -122.4464
    }
  },
  "time" : {
    "valid" : "Sun Dec 06 2009 10:18:22 GMT-0800 (PST)",
    "transaction" : "Sat Dec 12 2009 01:52:42 GMT-0800 (PST)"
  },
  "observation" : {
    "WaterTemperature" : 23.45,
    "SpecificConductivity" : 35.4,
    "Conductivity" : 139.6,
    "Resistivity" : 899.07,
    "Salinity" : 0.02,
    "Pressure" : 0.693,
    "Depth" : 2.224,
    "pH" : 6.25,
    "pHmV" : -76,
    "Turbidity" : 0.2,
    "ODOSaturation" : 31.3,
    "ODO" : 54.83,
    "Battery" : 1.1
  }
}


...
< date class in javascript is defined as [0-11]. Therefore, 10 is November.

<new Date(2009,11,8) , $lt:new Date(2009,11,15) }} ).count()
519112

<new Date(2009,11,8) , $lt:new Date(2009,11,15) }} ).limit(3)
{ "_id" : ObjectId("a96f40070178244beb640b00"), "message_id" : "10358fa1-53e4-4187-aef5-034b732189cf", "
    sensor" : { "ip_address" : "192.168.0.87", "location" : { "latitude" : 37.89155, "longitude" :
    -122.4464 } }, "time" : { "valid" : "Tue Dec 08 2009 04:51:29 GMT-0800 (PST)", "transaction" : "Sat
    Dec 12 2009 21:12:26 GMT-0800 (PST)" }, "observation" : { "WaterTemperature" : 48.03, "
    SpecificConductivity" : 127.4, "Conductivity" : 133.1, "Resistivity" : 2925.15, "Salinity" : 0.05, "
    Pressure" : 0.419, "Depth" : 2.218, "pH" : 2.85, "pHmV" : -90.8, "Turbidity" : 0, "ODOSaturation" :
    110.9, "ODO" : 42.96, "Battery" : 8.3 } }
{ "_id" : ObjectId("a96f40070178244bfa640b00"), "message_id" : "9933d4fc-350c-44b8-9e72-b344fa7520c6", "
    sensor" : { "ip_address" : "192.168.0.149", "location" : { "latitude" : 37.89155, "longitude" :
    -122.4464 } }, "time" : { "valid" : "Tue Dec 08 2009 04:51:29 GMT-0800 (PST)", "transaction" : "Sat
    Dec 12 2009 21:12:26 GMT-0800 (PST)" }, "observation" : { "WaterTemperature" : 41.73, "
    SpecificConductivity" : 126.8, "Conductivity" : 93.1, "Resistivity" : 3305.75, "Salinity" : 0.02, "
    Pressure" : 1.674, "Depth" : 1.004, "pH" : 3.48, "pHmV" : -54, "Turbidity" : 0.3, "ODOSaturation" :
    83.9, "ODO" : 36.61, "Battery" : 0.5 } }
{ "_id" : ObjectId("a96f40070178244b00650b00"), "message_id" : "8de2e398-3ffe-4ef8-81c1-a2cb942398ed", "
    sensor" : { "ip_address" : "192.168.0.137", "location" : { "latitude" : 37.89155, "longitude" :
    -122.4464 } }, "time" : { "valid" : "Tue Dec 08 2009 04:51:29 GMT-0800 (PST)", "transaction" : "Sat
    Dec 12 2009 21:12:26 GMT-0800 (PST)" }, "observation" : { "WaterTemperature" : 5.05, "
    SpecificConductivity" : 12.9, "Conductivity" : 114.2, "Resistivity" : 4991.7, "Salinity" : 0.05, "
    Pressure" : 0.167, "Depth" : 1.827, "pH" : 4.36, "pHmV" : -31.4, "Turbidity" : 0.3, "ODOSaturation" :
    76.4, "ODO" : 45.91, "Battery" : 3.2 } }

```

```
Explain the search over the shards for R1)
<new Date(2009,11,8) , $lt:new Date(2009,11,15) }} ).explain()
{
  "cursor" : "BtreeCursor time.valid_1",
  "startKey" : {
    "time.valid" : "Tue Dec 08 2009 00:00:00 GMT-0800 (PST)"
  },
  "endKey" : {
    "time.valid" : "Tue Dec 15 2009 00:00:00 GMT-0800 (PST)"
  },
  "nscanned" : 519112,
  "n" : 519112,
  "millis" : 1262,
  "oldPlan" : {
    "cursor" : "BtreeCursor time.valid_1",
    "startKey" : {
      "time.valid" : "Tue Dec 08 2009 00:00:00 GMT-0800 (PST)"
    },
    "endKey" : {
      "time.valid" : "Tue Dec 15 2009 00:00:00 GMT-0800 (PST)"
    }
  },
  "allPlans" : [
    {
      "cursor" : "BtreeCursor time.valid_1",
      "startKey" : {
        "time.valid" : "Tue Dec 08 2009 00:00:00 GMT-0800 (PST)"
      },
      "endKey" : {
        "time.valid" : "Tue Dec 15 2009 00:00:00 GMT-0800 (PST)"
      }
    }
  ]
}
>

Execute Scenario R2) search all the documents for a given sensor device by IP address. Count and Limit
     retrieval result
> db.SondeDataContainer.find( {"sensor.ip_address":"192.168.0.102"} ).count()
9707
<d( {"sensor.ip_address":"192.168.0.102"} ).limit(3)
{ "_id" : ObjectId("e26f40072f68234b11f4d600"), "message_id" : "b7712746-9a87-4ad3-abce-9a89efc1ef44", "
     sensor" : { "ip_address" : "192.168.0.102", "location" : { "latitude" : 37.89155, "longitude" :
     -122.4464 } }, "time" : { "valid" : "Mon Dec 07 2009 07:42:56 GMT-0800 (PST)", "transaction" : "Sat
     Dec 12 2009 01:52:42 GMT-0800 (PST)" }, "observation" : { "WaterTemperature" : 33.78, "
     SpecificConductivity" : 146.5, "Conductivity" : 69.1, "Resistivity" : 3113.28, "Salinity" : 0.02, "
     Pressure" : 0.487, "Depth" : 0.722, "pH" : 1.77, "pHmV" : -79.1, "Turbidity" : 0.1, "ODOSaturation" :
     2.2, "ODO" : 46.95, "Battery" : 5.4 } }
{ "_id" : ObjectId("e26f40072f68234b51f4d600"), "message_id" : "b0e67b21-9721-4441-81e6-543d472334f7", "
     sensor" : { "ip_address" : "192.168.0.102", "location" : { "latitude" : 37.89155, "longitude" :
     -122.4464 } }, "time" : { "valid" : "Fri Dec 18 2009 07:33:55 GMT-0800 (PST)", "transaction" : "Sat
     Dec 12 2009 01:52:42 GMT-0800 (PST)" }, "observation" : { "WaterTemperature" : 93.6, "
     SpecificConductivity" : 23.5, "Conductivity" : 163.1, "Resistivity" : 4705.03, "Salinity" : 0.04, "
     Pressure" : 0.287, "Depth" : 1.878, "pH" : 5.79, "pHmV" : -37.8, "Turbidity" : 0.1, "ODOSaturation" :
     46.2, "ODO" : 35.76, "Battery" : 2.5 } }
{ "_id" : ObjectId("e26f40072f68234b4bf5d600"), "message_id" : "f373b2db-9f66-4af6-8a7f-1c873776fecf", "
     sensor" : { "ip_address" : "192.168.0.102", "location" : { "latitude" : 37.89155, "longitude" :
     -122.4464 } }, "time" : { "valid" : "Sun Dec 06 2009 08:25:52 GMT-0800 (PST)", "transaction" : "Sat
     Dec 12 2009 01:52:42 GMT-0800 (PST)" }, "observation" : { "WaterTemperature" : 0.3, "
     SpecificConductivity" : 101.6, "Conductivity" : 41.9, "Resistivity" : 2874.45, "Salinity" : 0.05, "
     Pressure" : 0.354, "Depth" : 2.305, "pH" : 7.13, "pHmV" : -57.1, "Turbidity" : 0, "ODOSaturation" :
     92.8, "ODO" : 26.24, "Battery" : 5.4 } }
```

```

Explain the search over the shards for R2
<d( {"sensor.ip_address":"192.168.0.102"} ).explain()
{
  "cursor" : "BtreeCursor sensor.ip_address_1",
  "startKey" : {
    "sensor.ip_address" : "192.168.0.102"
  },
  "endKey" : {
    "sensor.ip_address" : "192.168.0.102"
  },
  "nscanned" : 9707,
  "n" : 9707,
  "millis" : 26,
  "oldPlan" : {
    "cursor" : "BtreeCursor sensor.ip_address_1",
    "startKey" : {
      "sensor.ip_address" : "192.168.0.102"
    },
    "endKey" : {
      "sensor.ip_address" : "192.168.0.102"
    }
  },
  "allPlans" : [
    {
      "cursor" : "BtreeCursor sensor.ip_address_1",
      "startKey" : {
        "sensor.ip_address" : "192.168.0.102"
      },
      "endKey" : {
        "sensor.ip_address" : "192.168.0.102"
      }
    }
  ]
}
>
<n values. Salinity = .01, Water Temperature = 46.47")

Execute Scenario R3) search all the documents that has some observation values. Salinity = .01, Water Temperature = 46.47
 <:0.01, "observation.WaterTemperature":46.47} ).count()
36
 <:0.01, "observation.WaterTemperature":46.47} ).limit(3)
{ "_id" : ObjectId("e26f40073368234b5e0ad600"), "message_id" : "6b9ef47e-756c-4fdd-8984-cd73f56b6ab8", "sensor" : { "ip_address" : "192.168.0.155", "location" : { "latitude" : 37.89155, "longitude" : -122.4464 } }, "time" : { "valid" : "Mon Dec 07 2009 07:42:56 GMT-0800 (PST)", "transaction" : "Sat Dec 12 2009 01:52:42 GMT-0800 (PST)" }, "observation" : { "WaterTemperature" : 46.47, "SpecificConductivity" : 78, "Conductivity" : 108.1, "Resistivity" : 2056.15, "Salinity" : 0.01, "Pressure" : 1.132, "Depth" : 0.665, "pH" : 0.83, "pHmV" : -25.6, "Turbidity" : 0, "ODOSaturation" : 35.6, "ODO" : 23.06, "Battery" : 6.1 } }
{ "_id" : ObjectId("e26f4007ef68234b2d9eda00"), "message_id" : "70d5bfde-4c80-42ba-b864-4e163e6b5593", "sensor" : { "ip_address" : "192.168.0.142", "location" : { "latitude" : 37.89155, "longitude" : -122.4464 } }, "time" : { "valid" : "Sun Dec 27 2009 09:18:17 GMT-0800 (PST)", "transaction" : "Sat Dec 12 2009 01:52:42 GMT-0800 (PST)" }, "observation" : { "WaterTemperature" : 46.47, "SpecificConductivity" : 71.8, "Conductivity" : 58, "Resistivity" : 2567.23, "Salinity" : 0.01, "Pressure" : 1.591, "Depth" : 2.236, "pH" : 7.19, "pHmV" : -23.7, "Turbidity" : 0.1, "ODOSaturation" : 107.1, "ODO" : 13.94, "Battery" : 2.2 } }
{ "_id" : ObjectId("e26f40075c69234b0031db00"), "message_id" : "5f242c0f-e051-4d76-b437-9e8100674c53", "sensor" : { "ip_address" : "192.168.0.59", "location" : { "latitude" : 37.89155, "longitude" : -122.4464 } }, "time" : { "valid" : "Sun Dec 13 2009 01:34:00 GMT-0800 (PST)", "transaction" : "Sat Dec 12 2009 01:52:42 GMT-0800 (PST)" }, "observation" : { "WaterTemperature" : 46.47, "SpecificConductivity" : 42, "Conductivity" : 10.2, "Resistivity" : 5162.14, "Salinity" : 0.01, "Pressure" : 1.223, "Depth" : 2.667, "pH" : 5.41, "pHmV" : -21.5, "Turbidity" : 0.2, "ODOSaturation" :
    24.1, "ODO" : 41.59, "Battery" : 5.1 } }


Explain the search over the shards for R3
<:0.01, "observation.WaterTemperature":46.47} ).explain()
{
  "cursor" : "BtreeCursor observation.WaterTemperature_1",
  "startKey" : {
    "observation.WaterTemperature" : 46.47
  },
  "endKey" : {
    "observation.WaterTemperature" : 46.47
  },
  "nscanned" : 226,
  "n" : 36,
  "millis" : 2,
  "oldPlan" : {
    "cursor" : "BtreeCursor observation.WaterTemperature_1",
    "startKey" : {
      "observation.WaterTemperature" : 46.47
    },
    "endKey" : {
      "observation.WaterTemperature" : 46.47
    }
  },
  "allPlans" : [
    {
      "cursor" : "BtreeCursor observation.WaterTemperature_1",
      "startKey" : {
        "observation.WaterTemperature" : 46.47
      },
      "endKey" : {
        "observation.WaterTemperature" : 46.47
      }
    },
    {
      "cursor" : "BtreeCursor observation.Salinity_1",
      "startKey" : {
        "observation.Salinity" : 0.01
      },
      "endKey" : {
        "observation.Salinity" : 0.01
      }
    },
    {
      "cursor" : "BtreeCursor observation.WaterTemperature_",
      "startKey" : {
        "observation.WaterTemperature" : 46.47
      },
      "endKey" : {
        "observation.WaterTemperature" : 46.47
      }
    },
    {
      "cursor" : "BtreeCursor observation.Salinity_",
      "startKey" : {
        "observation.Salinity" : 0.01
      },
      "endKey" : {
        "observation.Salinity" : 0.01
      }
    },
    {
        "cursor" : "BasicCursor",
        "startKey" : {

        },
        "endKey" : {

        }
      }
    ]
}
>

Execute Scenario U1) update all the documents for a given date December 12, by annotating them using tags
> db.SondeDataContainer.ensureIndex( { "tag":1 } )

Number of Observations from December 12
<":new Date(2009,11,12) , "$lt":new Date(2009,11,13) }}).count()
9085
> print ("How many with the tag 'oil spill'")
How many with the tag 'oil spill'
> db.SondeDataContainer.find( { "tag": "oil spill" }  ).count()
9085
<) }} ,  {"$set": {"tag": "oil spill"}} , false , true)
<notated with the tag 'oil spil' are as follows...")
Some documents annotated with the tag 'oil spil' are as follows...
<:new Date(2009,11,12) , $lt:new Date(2009,11,13) }} ).limit(1)
{ "_id" : ObjectId("cf6f40072cbc194b9ac1fd00"), "message_id" : "d7a63936-482c-449d-b870-a28e4786beb1", "
    sensor" : { "ip_address" : "192.168.0.209", "location" : { "latitude" : 37.89155, "longitude" :
    -122.4464 } }, "time" : { "valid" : "Sat Dec 05 2009 04:45:27 GMT-0800 (PST)", "transaction" : "Fri
    Dec 04 2009 17:48:13 GMT-0800 (PST)" }, "observation" : { "WaterTemperature" : 93.74, "
    SpecificConductivity" : 144.7, "Conductivity" : 79.6, "Resistivity" : 1501.75, "Salinity" : 0.08, "
    Pressure" : 1.825, "Depth" : 2.85, "pH" : 0.53, "pHmV" : -69.9, "Turbidity" : 0.1, "ODOSaturation" :
    56, "ODO" : 56.68, "Battery" : 8.8 }, "tag" : "oil spill" }

Execute Scenario D1) deleting all documents with for a given day (say Dec 13)
<new Date(2009,11,13) , $lt:new Date(2009,11,14) }} ).count()
207823
<e:new Date(2009,11,13) , $lt:new Date(2009,11,14) }} )
0
```

Listing A.32: Remote Shards Cluster Head Execution Log

```
Sat Dec 12 04:48:32 Mongo DB : starting : pid = 23708 port = 10000 dbpath = data/shards/config master = 0
    slave = 0  64-bit
Sat Dec 12 04:48:32 db version v1.2.0, pdfile version 4.5
Sat Dec 12 04:48:32 git version: 2add01f37ddf5a262dbc66f81353e88f38723dc6
Sat Dec 12 04:48:32 sys info: Linux ofc-n1.10gen.com 2.6.23.17-88.fc7 #1 SMP Thu May 15 00:02:29 EDT 2008
    x86_64 BOOST_LIB_VERSION=1_33_1
Sat Dec 12 04:48:32 waiting for connections on port 10000
Sat Dec 12 04:48:32 connection accepted from 127.0.0.1:46735 #1
Sat Dec 12 04:48:32 allocating new datafile data/shards/config/config.ns, filling with zeroes...
Sat Dec 12 04:48:32 done allocating datafile data/shards/config/config.ns, size: 16MB, took 0.049 secs
Sat Dec 12 04:48:32 allocating new datafile data/shards/config/config.0, filling with zeroes...
Sat Dec 12 04:48:32 done allocating datafile data/shards/config/config.0, size: 64MB, took 0.075 secs
Sat Dec 12 04:48:32 building new index on { _id: ObjId(000000000000000000000000) } for config.version...
Sat Dec 12 04:48:32 Buildindex config.version idxNo:0 { name: "_id_", ns: "config.version", key: { _id:
    ObjId(000000000000000000000000) } }
Sat Dec 12 04:48:32 done for 0 records 0secs
Sat Dec 12 04:48:32 insert config.version 141ms
Sat Dec 12 04:48:33 building new index on { _id: ObjId(000000000000000000000000) } for config.databases...
Sat Dec 12 04:48:33 Buildindex config.databases idxNo:0 { name: "_id_", ns: "config.databases", key: { _id
    : ObjId(000000000000000000000000) } }
```

```
Sat Dec 12 04:48:33 done for 0 records 0secs
Sat Dec 12 04:48:33 building new index on { _id: ObjId(000000000000000000000000) } for config.shards...
Sat Dec 12 04:48:33 Buildindex config.shards idxNo:0 { name: "_id_", ns: "config.shards", key: { _id:
        ObjId(000000000000000000000000) } }
Sat Dec 12 04:48:33 done for 0 records 0secs
Sat Dec 12 04:48:33 building new index on { _id: ObjId(000000000000000000000000) } for config.chunks...
Sat Dec 12 04:48:33 Buildindex config.chunks idxNo:0 { name: "_id_", ns: "config.chunks", key: { _id:
        ObjId(000000000000000000000000) } }
Sat Dec 12 04:48:33 done for 0 records 0.001secs
Sat Dec 12 04:48:35 connection accepted from 127.0.0.1:46740 #2
Sat Dec 12 17:51:51 query config.shards ntoreturn:0 reslen:184 nscanned:3 {}  nreturned:3 321ms
```

Listing A.33: Remote Shards Cluster Head Execution

```
marcello@netbeams-mongo-dev02:~/development/workspaces/netbeams/persistence$ more logs/mongos-cluster-head
        .log
Sat Dec 12 04:48:32 shardObjTest passed
Sat Dec 12 04:48:32 mongos v0.2.6 (alpha 2f) starting (--help for usage)
Sat Dec 12 04:48:32 git version: 2add01f37ddf5a262dbc66f81353e88f38723dc6
Sat Dec 12 04:48:32 sys info: Linux ofc-n1.10gen.com 2.6.23.17-88.fc7 #1 SMP Thu May 15 00:02:29 EDT 2008
        x86_
64 BOOST_LIB_VERSION=1_33_1
Sat Dec 12 04:48:32 creating new connection for pool to:localhost:10000
Sat Dec 12 04:48:32 waiting for connections on port 27017

Sat Dec 12 04:48:33 connection accepted from 127.0.0.1:54814 #1

Sat Dec 12 04:48:33 couldn't find database [admin] in config db
Sat Dec 12 04:48:33    put [admin] on: localhost:10000
Sat Dec 12 04:48:33 Request::process ns: admin.$cmd msg id:1807425400 attempt: 0
Sat Dec 12 04:48:33 single query: admin.$cmd  { addshard: "192.168.1.2:20001" }  ntoreturn: -1
Sat Dec 12 04:48:33 creating new connection for pool to:192.168.1.2:20001
Sat Dec 12 04:48:33 Request::process ns: admin.$cmd msg id:1807425401 attempt: 0
Sat Dec 12 04:48:33 single query: admin.$cmd  { addshard: "192.168.1.2:20002" }  ntoreturn: -1
Sat Dec 12 04:48:33 creating new connection for pool to:192.168.1.2:20002
Sat Dec 12 04:48:33 Request::process ns: admin.$cmd msg id:1807425402 attempt: 0
Sat Dec 12 04:48:33 single query: admin.$cmd  { addshard: "localhost:20001", allowLocal: true }  ntoreturn
        : -1
Sat Dec 12 04:48:33 creating new connection for pool to:localhost:20001
Sat Dec 12 04:48:33 Request::process ns: admin.$cmd msg id:1807425403 attempt: 0
Sat Dec 12 04:48:33 single query: admin.$cmd  { listshards: 1.0 }  ntoreturn: -1
Sat Dec 12 04:48:33 Request::process ns: admin.$cmd msg id:1807425404 attempt: 0
Sat Dec 12 04:48:33 single query: admin.$cmd  { enablesharding: "netbeams" }  ntoreturn: -1

Sat Dec 12 04:48:33 couldn't find database [netbeams] in config db
Sat Dec 12 04:48:33    put [netbeams] on: 192.168.1.2:20002
Sat Dec 12 04:48:33 Request::process ns: admin.$cmd msg id:1807425405 attempt: 0
Sat Dec 12 04:48:33 single query: admin.$cmd  { shardcollection: "netbeams.SondeDataContainer", key: {
        observa
tion.pH: 1.0 } }  ntoreturn: -1
Sat Dec 12 04:48:33 no chunks for:netbeams.SondeDataContainer so creating first: shard  ns:netbeams.
        SondeDataC
ontainer shard: 192.168.1.2:20002 min: { observation.pH: MinKey } max: { observation.pH: MaxKey }
Sat Dec 12 04:48:33 before: { _id: ObjId(4b2366f19aa4e6dfcf750197) }  shard  ns:netbeams.
        SondeDataContainer
shard: 192.168.1.2:20002 min: { observation.pH: MinKey } max: { observation.pH: MaxKey }
Sat Dec 12 04:48:33 after : { _id: ObjId(4b2366f19aa4e6dfcf750197) }  shard  ns:netbeams.
        SondeDataContainer
shard: 192.168.1.2:20002 min: { observation.pH: MinKey } max: { observation.pH: MaxKey }
Sat Dec 12 04:48:33 Request::process ns: netbeams.system.indexes msg id:1807425406 attempt: 0
Sat Dec 12 04:48:33  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 04:48:33 Request::process ns: netbeams.$cmd msg id:1807425407 attempt: 0
Sat Dec 12 04:48:33 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: −1
Sat Dec 12 04:48:34 couldn't find database [test] in config db
Sat Dec 12 04:48:34    put [test] on: 192.168.1.2:20002

Sat Dec 12 04:48:34 Request::process ns: test.$cmd msg id:1807425408 attempt: 0
Sat Dec 12 04:48:34 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: −1
Sat Dec 12 04:48:34 Request::process ns: netbeams.system.indexes msg id:1807425409 attempt: 0
Sat Dec 12 04:48:34  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 04:48:34 Request::process ns: netbeams.$cmd msg id:1807425410 attempt: 0
Sat Dec 12 04:48:34 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: −1
Sat Dec 12 04:48:34 Request::process ns: test.$cmd msg id:1807425411 attempt: 0
Sat Dec 12 04:48:34 single query: test.$cmd  { getlasterror: 1.0 }  ntoreturn: −1
Sat Dec 12 04:48:34 Request::process ns: netbeams.system.indexes msg id:1807425412 attempt: 0
Sat Dec 12 04:48:34  .system.indexes write for: netbeams.system.indexes
Sat Dec 12 04:48:34 Request::process ns: netbeams.$cmd msg id:1807425413 attempt: 0
Sat Dec 12 04:48:34 single query: netbeams.$cmd  { getlasterror: 1.0 }  ntoreturn: −1


Sat Dec 12 04:53:09  have to set shard version for conn: 0x1976f90 ns:netbeams.SondeDataContainer my last
    seq: 0  current: 3
Sat Dec 12 04:53:09     setShardVersion  192.168.1.2:20002  netbeams.SondeDataContainer  { setShardVersion
    : "netbeams.SondeDataContainer", configdb: "localhost:10000", version: Timestamp 1260611313000|1
, serverID: ObjId(4b2366f09aa4e6dfcf750194) } 0x1976f90
Sat Dec 12 04:53:09        setShardVersion failed!
{ need_authoritative: true, errmsg: "first setShardVersion", ok: 0.0 }
Sat Dec 12 04:53:09 reloading shard info for: netbeams.SondeDataContainer
Sat Dec 12 04:53:09  have to set shard version for conn: 0x1976f90 ns:netbeams.SondeDataContainer my last
    seq: 0  current: 4
Sat Dec 12 04:53:09     setShardVersion  192.168.1.2:20002  netbeams.SondeDataContainer  { setShardVersion
    : "netbeams.SondeDataContainer", configdb: "localhost:10000", version: Timestamp 1260611313000|1
, serverID: ObjId(4b2366f09aa4e6dfcf750194), authoritative: true } 0x1976f90
Sat Dec 12 04:53:09       setShardVersion success!

Sat Dec 12 19:47:15     setShardVersion  192.168.1.2:20002  netbeams.SondeDataContainer  { setShardVersion
    : "netbeams.SondeDataContainer", configdb: "localhost:10000", version: Timestamp 1260611313000|1,
    serverID: ObjId(4b2366f09aa4e6dfcf750194), authoritative: true } 0x1b1b8a0
Sat Dec 12 19:47:16       setShardVersion success!
Sat Dec 12 19:47:16 Request::process ns: netbeams.SondeDataContainer msg id:21 attempt: 0
Sat Dec 12 19:47:16 write: netbeams.SondeDataContainer
Sat Dec 12 19:47:16 Request::process ns: netbeams.SondeDataContainer msg id:22 attempt: 0
Sat Dec 12 19:47:16 write: netbeams.SondeDataContainer
Sat Dec 12 19:47:16 Request::process ns: netbeams.SondeDataContainer msg id:23 attempt: 0
Sat Dec 12 19:47:16 write: netbeams.SondeDataContainer
Sat Dec 12 19:47:16 Request::process ns: netbeams.SondeDataContainer msg id:24 attempt: 0
...
...
Sat Dec 12 04:25:03 Request::process ns: netbeams.SondeDataContainer msg id:483857 attempt: 0
Sat Dec 12 04:25:03 write: netbeams.SondeDataContainer
Sat Dec 12 04:25:03 Request::process ns: netbeams.SondeDataContainer msg id:483858 attempt: 0
Sat Dec 12 04:25:03 write: netbeams.SondeDataContainer
Sat Dec 12 04:25:03 Request::process ns: netbeams.SondeDataContainer msg id:483859 attempt: 0
Sat Dec 12 04:25:03 write: netbeams.SondeDataContainer
Sat Dec 12 04:25:14 end connection 192.168.1.13:60950
```

Listing A.34: Remote Shard Execution Log

```
Sat Dec 12 16:38:27 Mongo DB : starting : pid = 4340 port = 20002 dbpath = /mongo−data/shard−2/ master = 0
     slave = 0  64−bit
Sat Dec 12 16:38:27 db version v1.2.0, pdfile version 4.5
Sat Dec 12 16:38:27 git version: 2add01f37ddf5a262dbc66f81353e88f38723dc6
Sat Dec 12 16:38:27 sys info: Darwin erh2.10gen.cc 9.6.0 Darwin Kernel Version 9.6.0: Mon Nov 24 17:37:00 PST 2008; root:xnu-1228.9.59~1/RELEASE_I386 i386 BOOST_LIB_VERSION=1_37
Sat Dec 12 16:38:27 waiting for connections on port 20002
Sat Dec 12 16:58:17 connection accepted from 192.168.1.10:45202 #1
Sat Dec 12 16:58:17 allocating new datafile /mongo-data/shard-2/netbeams.ns, filling with zeroes...
Sat Dec 12 16:58:17 done allocating datafile /mongo-data/shard-2/netbeams.ns, size: 16MB, took 0.028 secs
Sat Dec 12 16:58:18 allocating new datafile /mongo-data/shard-2/netbeams.0, filling with zeroes...
Sat Dec 12 16:58:18 done allocating datafile /mongo-data/shard-2/netbeams.0, size: 64MB, took 0.099 secs
Sat Dec 12 16:58:19 building new index on { _id: ObjId(000000000000000000000000) } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:0 { name: "_id_", ns: "netbeams.SondeDataContainer", key: { _id: ObjId(000000000000000000000000) } }
Sat Dec 12 16:58:19 allocating new datafile /mongo-data/shard-2/netbeams.1, filling with zeroes...
Sat Dec 12 16:58:19 done for 0 records 0.051secs
Sat Dec 12 16:58:19 info: creating collection netbeams.SondeDataContainer on add index
Sat Dec 12 16:58:19 building new index on { sensor.location.latitude: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:1 { name: "sensor.location.latitude_", ns: "netbeams.SondeDataContainer", key: { sensor.location.latitude: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:17 insert netbeams.system.indexes 2235ms
Sat Dec 12 16:58:19 building new index on { sensor.location.longitude: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:2 { name: "sensor.location.longitude_", ns: "netbeams.SondeDataContainer", key: { sensor.location.longitude: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0.063secs
Sat Dec 12 16:58:19 building new index on { time.valid: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:3 { name: "time.valid_", ns: "netbeams.SondeDataContainer", key: { time.valid: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { time.transaction: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:4 { name: "time.transaction_", ns: "netbeams.SondeDataContainer", key: { time.transaction: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { observation.WaterTemperature: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:5 { name: "observation.WaterTemperature_", ns: "netbeams.SondeDataContainer", key: { observation.WaterTemperature: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { observation.SpecificConductivity: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:6 { name: "observation.SpecificConductivity_", ns: "netbeams.SondeDataContainer", key: { observation.SpecificConductivity: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { observation.Conductivity: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:7 { name: "observation.Conductivity_", ns: "netbeams.SondeDataContainer", key: { observation.Conductivity: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0.001secs
Sat Dec 12 16:58:19 building new index on { observation.Resistivity: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:8 { name: "observation.Resistivity_", ns: "netbeams.SondeDataContainer", key: { observation.Resistivity: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { observation.Salinity: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:9 { name: "observation.Salinity_", ns: "netbeams.SondeDataContainer", key: { observation.Salinity: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { observation.Pressure: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:10 { name: "observation.Pressure_", ns: "netbeams.SondeDataContainer", key: { observation.Pressure: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { observation.Depth: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:11 { name: "observation.Depth_", ns: "
      netbeams.SondeDataContainer", key: { observation.Depth: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { observation.pH: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:12 { name: "observation.pH_", ns: "
      netbeams.SondeDataContainer", key: { observation.pH: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { observation.pHmV: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:13 { name: "observation.pHmV_", ns: "
      netbeams.SondeDataContainer", key: { observation.pHmV: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { observation.Turbidity: "1" } for netbeams.SondeDataContainer
      ...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:14 { name: "observation.Turbidity_", ns:
      "netbeams.SondeDataContainer", key: { observation.Turbidity: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0.002secs
Sat Dec 12 16:58:19 building new index on { observation.ODOSaturation: "1" } for netbeams.
      SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:15 { name: "observation.ODOSaturation_",
      ns: "netbeams.SondeDataContainer", key: { observation.ODOSaturation: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { observation.ODO: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:16 { name: "observation.ODO_", ns: "
      netbeams.SondeDataContainer", key: { observation.ODO: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 building new index on { observation.Battery: "1" } for netbeams.SondeDataContainer...
Sat Dec 12 16:58:19 Buildindex netbeams.SondeDataContainer idxNo:17 { name: "observation.Battery_", ns: "
      netbeams.SondeDataContainer", key: { observation.Battery: "1" }, unique: false }
Sat Dec 12 16:58:19 done for 0 records 0secs
Sat Dec 12 16:58:19 done allocating datafile /mongo-data/shard-2/netbeams.1, size: 128MB, took 0.256 secs
```

Listing A.35: Data Export to from mongoDB to OPeNDAP format.

```
 Dataset: ntb_YSI_PUF.dat
YSI_REALTIME_CSV.Month,YSI_REALTIME_CSV.Day,YSI_REALTIME_CSV.Year,YSI_REALTIME_CSV.Hour,YSI_REALTIME_CSV.
      Min,YSI_REALTIME_CSV.Sec,YSI_REALTIME_CSV.WaterTemperature,YSI_REALTIME_CSV.SpecificConductivity,
      YSI_REALTIME_CSV.Conductivity,YSI_REALTIME_CSV.Resistivity,YSI_REALTIME_CSV.Salinity,YSI_REALTIME_CSV.
      Pressure,YSI_REALTIME_CSV.Depth,YSI_REALTIME_CSV.pH,YSI_REALTIME_CSV.pHmV,YSI_REALTIME_CSV.Turbidity,
      YSI_REALTIME_CSV.ODOSaturation,YSI_REALTIME_CSV.ODO,YSI_REALTIME_CSV.Battery
12,19,2009,13,24,27,18.84,51.6,85.4,5447.6,0.08,1.826,2.471,1.47,-8.6,0.2,101.8,15.84,6.7
12,16,2009,14,02,38,86.97,13.5,60.2,5229.99,0.05,0.401,2.369,7.25,-18.4,0.1,87.3,46.68,4.7
12,19,2009,13,24,27,51.29,110.7,104.5,1710.07,0.01,1.094,1.808,3.04,-46.4,0.2,59.0,10.66,7.1
12,25,2009,11,35,24,12.71,93.5,101.7,1183.54,0.09,1.252,2.166,5.52,-11.2,0.2,81.8,6.36,3.7
12,22,2009,16,05,20,47.75,93.9,52.9,1972.35,0.07,0.355,0.222,3.2,-66.0,0.0,2.5,14.51,5.1
```

Listing A.36: JSON representation of the data produced by a YSI sensor on mongoDB

```
{
    "_id" : ObjectId("5d6f40078ec8074bba980900"),
    "message_id" : "020d82e1-18b2-4fe2-800c-a0f68e22ea86",
    "sensor" : {
        "ip_address" : "192.168.0.91",
        "location" : {
            "latitude" : 37.89155,
            "longitude" : -122.4464
        }
    },
    "time" : {
        "valid" : "Wed Nov 04 2009 02:30:12 GMT−0800 (PST)",
        "transaction" : "Sat Nov 21 2009 03:01:33 GMT−0800 (PST)"
    },
    "observation" : {
        "WaterTemperature" : 88.58,
        "SpecificConductivity" : 180.5,
        "Conductivity" : 167.3,
        "Resistivity" : 491.89,
        "Salinity" : 0.06,
        "Pressure" : 1.109,
        "Depth" : 2.642,
        "pH" : 7.11,
        "pHmV" : −42.9,
        "Turbidity" : 0.2,
        "ODOSaturation" : 72.4,
        "ODO" : 10.85,
        "Battery" : 2.7
    }
}
```

Listing A.37: HTTP GET Request x Response to the OPeNDAP server at SF-BEAMS

```
http://sfbeams.sfsu.edu:8080/opendap/sfbeams/data_ctd/rtc_ctd5−ysimoor/real−time/sfb_CTD5_PUF.dat.ascii?
GET /opendap/sfbeams/data_ctd/rtc_ctd5−ysimoor/real−time/sfb_CTD5_PUF.dat.dods?&Month=11&Day=12 HTTP/1.1

HTTP/1.x 200 OK
Server: Apache−Coyote/1.1
Last−Modified: Thu, 12 Nov 2009 17:55:06 GMT
XDODS−Server: Server−Version−Unknown
XOPeNDAP−Server: bes/3.5.1 freeform_handler/3.7.5, netcdf_handler/3.7.6
XDAP: 3.2
Content−Description: dods_data
Content−Type: text/txt
Date: Sun, 15 Nov 2009 03:08:45 GMT

Dataset: sfb_CTD5_PUF.dat
YSI_REALTIME_CSV.Month, YSI_REALTIME_CSV.Day, YSI_REALTIME_CSV.Year, YSI_REALTIME_CSV.Hour,
    YSI_REALTIME_CSV.Min, YSI_REALTIME_CSV.Sec, YSI_REALTIME_CSV.WaterTemperature, YSI_REALTIME_CSV.
    SpecificConductivity, YSI_REALTIME_CSV.Conductivity, YSI_REALTIME_CSV.Resistivity, YSI_REALTIME_CSV.
    TDS, YSI_REALTIME_CSV.Salinity, YSI_REALTIME_CSV.Pressure, YSI_REALTIME_CSV.Depth, YSI_REALTIME_CSV.pH
    , YSI_REALTIME_CSV.pHmV, YSI_REALTIME_CSV.Turbidity, YSI_REALTIME_CSV.ODOSaturation, YSI_REALTIME_CSV.
    ODO, YSI_REALTIME_CSV.Chlor, YSI_REALTIME_CSV.Chlor_RFU, YSI_REALTIME_CSV.Battery, YSI_REALTIME_CSV.
    InstSN
11, 12, 2009, 16, 30, 51, 13.78, 4.3945, 3.4532, 0, 28.56, 28.35, 2.65, 2.651, 7.9, −54.7, 9, −99999,
    −99999, 3.7, 1, 9.4, −99999
11, 12, 2009, 16, 36, 50, 13.78, 4.4095, 3.4642, 0, 28.66, 28.46, 2.628, 2.63, 7.91, −54.9, 11.9, −99999,
    −99999, 4, 1.1, 9.4, −99999
11, 12, 2009, 16, 42, 51, 13.77, 4.3943, 3.4515, 0, 28.56, 28.35, 2.631, 2.632, 7.9, −54.9, 9.8, −99999,
    −99999, 3.6, 1, 9.4, −99999
11, 12, 2009, 16, 48, 50, 13.76, 4.3893, 3.447, 0, 28.53, 28.32, 2.628, 2.63, 7.9, −54.8, 10.6, −99999,
    −99999, 3.8, 1, 9.4, −99999
```

Listing A.38: Export Command to CSV format

```
mongoexport −d netbeams −c SondeDataContainer −−dbpath data −−csv −f "_id,sensor_ip_address,
    transaction_time,fact_time,data.temperature,data.sp_condition,data.condition,data.resistence,data.
    salinitude,data.pressure,data.depth,data.ph,data.pH_mv,data.odo_sat,data.odo_condition,data.turbidity,
    data.battery" −o logs/experiment−483840−data−exported−20091129−190207.csv

```

```
Sun Nov 29 19:07:04 query netbeams.SondeDataContainer ntoreturn:0 reslen:2258 nscanned:101 { query: {}, $
     snapshot: 1 }  nreturned:101 202ms
Sun Nov 29 19:07:04 getmore netbeams.SondeDataContainer cid:7749363894863150628 ntoreturn:0 query: {
     query: {}, $snapshot: 1 }  bytes:1048562 nreturned:47661 1767ms
...
...
Sun Nov 29 19:10:18 getmore netbeams.SondeDataContainer cid:7749363894863150628 ntoreturn:0 query: {
     query: {}, $snapshot: 1 }  bytes:793098 nreturned:36049 1975ms
exported 2419200 records
```

Bibliography

[AGH05] Ken Arnold, James Gosling, and David Holmes. *Java(TM) Programming Language, The (4th Edition).* Addison-Wesley Professional, 2005.

[ASSC02] I.F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102–114, Aug 2002.

[Bai09] Tony Bain. Is the relational database doomed? `http://www.readwriteweb.com/enterprise/2009/02/is-the-relational-database-doomed.php`, February 2009.

[bas01] Bourne shell. `http://en.wikipedia.org/wiki/Bourne_shell`, January 2001.

[BDD09] Lan S. Bai, Robert P. Dick, and Peter A. Dinda. Archetype-based design: Sensor network programming for application experts, not just programming experts. In *IPSN '09: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pages 85–96, Washington, DC, USA, 2009. IEEE Computer Society.

[BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language user guide.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1999.

[bso09] mongodb - binary json. `http://www.mongodb.org/display/DOCS/BSON`, August 2009.

[Buc08] Peter Buckingham. Random footnotes of engineering - map reduce. `http://blogs.sun.com/fifors/entry/map_reduce`, December 2008.

[BYV$^+$09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, 2009.

[Cam07] Neil Campbell. The evolution of flight data analysis. In *ASASI 2007: Proceedings of Australian Society of Air Safety Investigators conference*, Civic Square, Australia, 2007.

[CDG$^+$08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):1–26, 2008.

[CFL$^+$07] Whei-Jen Chen, Alain Fisher, Aman Lalla, Andrew McLauchlan, and Doug Agnew. Database partitioning, table partitioning, and mdc for db2 9. `http://www.redbooks.ibm.com/abstracts/SG247467.html`, August 2007.

[CHZ09] Xujin Chen, Xiaodong Hu, and Jianming Zhu. Data gathering schedule for minimal aggregation time in wireless sensor networks. *Int. J. Distrib. Sen. Netw.*, 5(4):321–337, 2009.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[cou09] The couchdb project. `http://couchdb.apache.org/`, August 2009.

[CSFP09] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control With Subversion - The Official Guide And Reference Manual*. CreateSpace, Paramount, CA, 2009.

[CWH00] Mary Campione, Kathy Walrath, and Alison Huml. *The Java Tutorial: A Short Course on the Basics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[CZR03] A. Chaudhri, Roberto Zicari, and Awais Rashid. *XML Data Management: Native XML and XML Enabled DataBase Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[dAFN+08] André L. L. de Aquino, Carlos M. S. Figueiredo, Eduardo F. Nakamura, Alejandro C. Frery, Antonio A. F. Loureiro, and Antônio Otávio Fernandes. Sensor stream reduction for clustered wireless sensor networks. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2052–2056, New York, NY, USA, 2008. ACM.

[dAPZJF+01] Ana Luiza de Almeida Pereira Zuquim, Claudionor JosÃ© Nunes Coelho Jr, AntÃ´nio OtÃ¡vio Fernandes, Marcos PÃªgo de Oliveira, and AndrÃ©a Iabrudi Tavares. An embedded converter from rs232 to universal serial bus. *Integrated Circuit Design and System Design, Symposium on*, 0:0091, 2001.

[db209] Ibm db2 software. `http://www-01.ibm.com/software/data/db2/xml/`, July 2009.

[dec09] Decimal degrees format. `http://en.wikipedia.org/wiki/Decimal_degrees`, October 2009.

[DG04] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications ACM*, 51(1):107–113, 2008.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 205–220, New York, NY, USA, 2007. ACM.

[DML03] Enrique J. Duarte-Melo and Mingyan Liu. Data-gathering wireless sensor networks: organization and capacity. *Comput. Netw.*, 43(4):519–537, 2003.

[dS09] Marcello de Sales. The data sensor platform architecture. `http://code.google.com/p/netbeams/wiki/DataSensorPlatform`, December 2009.

[Duk09] R. V. Dukkipati. *Analysis & Design of Control Systems using MATLAB*. New Age International (P) Limited, New Delhi, India, India, 2009.

[EB82] M. Earle and K. Bush. Strapped-down accelerometer effects on ndbo wave measurements. In *OCEANS 1984*, volume 14, pages 838–848, Sep 1982.

[Els08] Amr Elssamadisy. *Agile Adoption Patterns: A Roadmap to Organizational Success.* Addison-Wesley Professional, 2008.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* Information and computer science, University of California, Irvine, 2000.

[FM08] David Flanagan and Yukihiro Matsumoto. *The ruby programming language.* O'Reilly, 2008.

[Gos06] Darren Gosbell. Key value pairs in database design. `http://geekswithblogs.net/darrengosbell/articles/KVPsInDatabaseDesign.aspx`, March 2006.

[gum08] Gumstix. `http://www.ibm.com/developerworks/rational/library/3101.html`, February 2008.

[HK02] Jan Hannemann and Gregor Kiczales. Design pattern implementation in java and aspectj. In *OOPSLA '02: Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 161–173, New York, NY, USA, 2002. ACM.

[HOE$^+$09] Elyes Ben Hamida, Hideya Ochiai, Hiroshi Esaki, Pierre Borgnat, Patrice Abry, and Eric Fleury. Measurement analysis of the live e! sensor network: Spatial-temporal correlations and data aggregation. In *SAINT '09: Proceedings of the 2009 Ninth Annual International Symposium on Applications and the Internet*, pages 263–266, Washington, DC, USA, 2009. IEEE Computer Society.

[HRGL08] N. Hoeller, C. Reinke, S. Groppe, and V. Linnemann. Xobe sensor networks: Integrating xml in sensor network programming. In *INSS 2008. 5th International Conference on Networked Sensing Systems*, pages 229–229. IEEE Computer Society, June 2008.

[HRN$^+$08] Nils Hoeller, Christoph Reinke, Jana Neumann, Sven Groppe, Daniel Boeckmann, and Volker Linnemann. Efficient xml usage within wireless sensor networks. In *WICON '08: Proceedings of the 4th Annual International Conference on Wireless Internet*, pages 1–10, ICST, Brussels, Belgium, Belgium,

2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[jav98] The javascript scripting language. `http://en.wikipedia.org/wiki/JavaScript`, November 1998.

[KB09] Kyoung-Don Kang and Can Basaran. Adaptive data replication for load sharing in a sensor data center. In *ICDCSW '09: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems Workshops*, pages 20–25, Washington, DC, USA, 2009. IEEE Computer Society.

[KBK+05] Rick Kazman, Len Bass, Mark Klein, Tony Lattanze, and Linda Northrop. A basis for analyzing software architecture analysis methods. *Software Quality Control*, 13(4):329–355, 2005.

[Ken83] William Kent. A simple guide to five normal forms in relational database theory. *Communununications ACM*, 26(2):120–125, 1983.

[KGH07] Majid I. Khan, Wilfried N. Gansterer, and Gunter Haring. In-network storage model for data persistence under congestion in wireless sensor network. In *CISIS '07: Proceedings of the First International Conference on Complex, Intelligent and Software Intensive Systems*, pages 221–228, Washington, DC, USA, 2007. IEEE Computer Society.

[kno08] Knopflerfish. `http://www.knopflerfish.org/`, October 2008.

[Lar04] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[LCS09] Xiaotao Liu, Mark Corner, and Prashant Shenoy. Seva: Sensor-enhanced video annotation. *ACM Trans. Multimedia Comput. Commun. Appl.*, 5(3):1–26, 2009.

[Li09] Peng Li. Exploring virtual environments in a decentralized lab. *SIGITE Newsl.*, 6(1):4–10, 2009.

[LKK07] Seungjae Lee, Changhwa Kim, and Sangkyung Kim. New database operators for sensor networks. In *SERA '07: Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management & Applications*, pages 689–696, Washington, DC, USA, 2007. IEEE Computer Society.

[LKN+09] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's inside the cloud? an architectural map of the cloud landscape. In *CLOUD '09: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23–31, Washington, DC, USA, 2009. IEEE Computer Society.

[LKNZ08] Liqian Luo, Aman Kansal, Suman Nath, and Feng Zhao. Sharing and exploring sensor streams over geocentric interfaces. In *GIS '08: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, pages 1–10, New York, NY, USA, 2008. ACM.

[LNH05] Jonathan Ledlie, Chaki Ng, and David A. Holland. Provenance-aware sensor data storage. In *ICDEW '05: Proceedings of the 21st International Conference on Data Engineering Workshops*, page 1189, Washington, DC, USA, 2005. IEEE Computer Society.

[LSPS07] Mika Luimula, Kirsti Sääskilahti, Timo Partala, and Ossi Saukko. Techniques for location selection on a mobile device. In *EATIS '07: Proceedings of the 2007 Euro American conference on Telematics and information systems*, pages 1–4, New York, NY, USA, 2007. ACM.

[Mel96] Jim Melton. Sql language summary. *ACM Comput. Surv.*, 28(1):141–143, 1996.

[Mer09a] Dwight Merriman. Dot notation (reaching into objects). `http://www.mongodb.org/display/DOCS/Dot+Notation+(Reaching+into+Objects)`, October 2009.

[Mer09b] Dwight Merriman. Sharding introduction - mongodb. `http://www.mongodb.org/display/DOCS/Sharding+Introduction`, June 2009.

[MFHH05] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[MHO04] Kirk Martinez, Jane K. Hart, and Royan Ong. Environmental sensor networks. *Computer*, 37(8):50–56, 2004.

[MMLM07] Simon Miles, Steve Munroe, Michael Luck, and Luc Moreau. Modelling the provenance of data in autonomous systems. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[mon09] mongodb: Combining the best features of document databases, key-value stores, and rdbmses. `http://www.mongodb.org/`, August 2009.

[Moo05] Matthew Moodie. *Pro Apache Ant (Pro)*. Apress, Berkely, CA, USA, 2005.

[mys99] Mysql database. `http://dev.mysql.com/`, April 1999.

[Nik05] Jacob Nikom. Real-time sensor data warehouse architecture using mysql. In *MySQL Users Conference*. O'Reilly Media, Inc., April 2005.

[NLF07] Eduardo F. Nakamura, Antonio A. F. Loureiro, and Alejandro C. Frery. Information fusion for wireless sensor networks: Methods, models, and classifications. *ACM Comput. Surv.*, 39(3):9, 2007.

[Oba09] Dare Obasanjo. Building scalable databases: Pros and cons of various database sharding schemes. `http://www.25hoursaday.com/weblog/2009/01/16/BuildingScalableDatabasesProsAndConsOfVariousDatabaseShardingSc` `aspx`, January 2009.

[OCR$^+$09] M.F. O'Connor, K. Conroy, M. Roantree, A.F. Smeaton, and N.M. Moyna. Querying xml data streams from wireless sensor networks: An evaluation of query engines. In *RCIS 2009: Third International Conference on Research Challenges in Information Science*, pages 23 – 30. IEEE Computer Society, April 2009.

[Oli06] Tal Olier. Database design using key-value tables. `http://www.devshed.com/c/a/MySQL/Database-Design-Using-KeyValue-Tables/`, February 2006.

[ope08] OPeNDAP. `http://opendap.org/`, October 2008.

[orm02] Object-relational mapping. `http://en.wikipedia.org/wiki/Object-relational_mapping`, January 2002.

[OS95] G. Ozsoyoglu and R.T. Snodgrass. Temporal and real-time databases: a survey. *Knowledge and Data Engineering, IEEE Transactions on*, 7(4):513–532, Aug 1995.

[osg08] OSGi. `http://www.osgi.org/Main/HomePage`, September 2008.

[PGVA08] Prasanna Padmanabhan, Le Gruenwald, Anita Vallur, and Mohammed Atiquzzaman. A survey of data replication techniques for mobile ad hoc network databases. *The VLDB Journal*, 17(5):1143–1164, 2008.

[PJS+09] Arno Puder, Teresa Johnson, Kleber Sales, Marcello de Sales, and Dale Davidson. A component-based sensor network for environmental monitoring. In *SNA-2009: 1st International Conference on Sensor Networks and Applications*, pages 54–60, San Francisco, CA, USA, 2009. The International Society for Computers and Their Applications - ISCA.

[Pri08] Dan Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, 2008.

[pyt09] The python language reference. `http://docs.python.org/reference/`, December 2009.

[RM04] K. Romer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.

[RRGP06] C. Raleigh, D. Robinson, N. Garfield, and D. Peterson. Sf-beams: an evolving estuarine ocean observing system. In *OS2006: 13th Ocean Sciences Meeting of ASLO, ERF, TOS and AGU*, Honolulu, HI, USA, February 2006. (AGU) - American Geophysical Union.

[Sch06] Robin Schumacher. Improving database performance with partitioning. `http://dev.mysql.com/tech-resources/articles/performance-partitioning.html`, March 2006.

[sfb08] SFBEAMS SYSTEM SPECIFICATIONS. `http://sfbeams.sfsu.edu/system.htm`, August 2008.

[sfb09] Oil spills into s.f. bay south of bay bridge. `http://www.sfgate.com/cgi-bin/article.cgi?f=/c/a/2009/10/30/BA9B1ACTST.DTL`, October 2009.

[sha09] An Unorthodox Approach to Database Design: The Coming of the Shard. `http://highscalability.com/unorthodox-approach-database-design-coming-shard`, August 2009.

[SLM06] Bo Sheng, Qun Li, and Weizhen Mao. Data storage placement in sensor networks. In *MobiHoc '06: Proceedings of the 7th ACM international symposium* *on Mobile ad hoc networking and computing*, pages 344–355, New York, NY, USA, 2006. ACM.

[SLWK07] Daby M. Sow, Lipyeow Lim, Min Wang, and Kyu Hyun Kim. Persisting and querying biometric event streams with hybrid relational-xml dbms. In *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 189–197, New York, NY, USA, 2007. ACM.

[sn-08] Swiss experiment: a new environmental monitoring platform in alpine environment. `http://lsir-swissex.epfl.ch/index.php/SwissEx:Publications`, December 2008.

[SPBV08] Régis Saint-Paul, Boualem Benatallah, and Julien Vayssière. Data services in your spreadsheet! In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 690–694, New York, NY, USA, 2008. ACM.

[SRK$^+$03] Scott Shenker, Sylvia Ratnasamy, Brad Karp, Ramesh Govindan, and Deborah Estrin. Data-centric storage in sensornets. *SIGCOMM Comput. Commun. Rev.*, 33(1):137–142, 2003.

[SSK$^+$08] Wen Zhan Song, B. Shirazi, S. Kedar, S. Chien, F. Webb, D. Tran, A. Davis, D. Pieri, R. LaHusen, J. Pallister, D. Dzurisin, S. Moran, and M. Lisowski. Optimized autonomous space in-situ sensor-web for volcano monitoring. In *Aerospace Conference, 2008 IEEE*, pages 1–10, March 2008.

[Ste08] Stoyan Stefanov. *Object-Oriented JavaScript.* Packt Publishing, 2008.

[tax09] Taxonomy. `http://en.wikipedia.org/wiki/Taxonomy`, September 2009.

[tin09] Tinydb: A declarative database for sensor networks. `http://telegraph.cs.berkeley.edu/tinydb/`, March 2009.

[TV08] Andre L.C. Tavares and Marco Tulio Valente. A gentle introduction to osgi. *SIGSOFT Softw. Eng. Notes*, 33(5):1–5, 2008.

[use07] Are use cases beneficial for developers using agile requirements? In *CERE '07: Proceedings of the 2007 Fifth International Workshop on Comparative Evaluation in Requirements Engineering*, pages 11–22, Washington, DC, USA, 2007. IEEE Computer Society.

[vKS07] Georg von Krogh and Sebastian Spaeth. The open source software phenomenon: Characteristics that promote research. *J. Strateg. Inf. Syst.*, 16(3):236–253, 2007.

[WX06] Lan Wang and Yang Xiao. A survey of energy-efficient scheduling mechanisms in sensor networks. *Mob. Netw. Appl.*, 11(5):723–740, 2006.

[xml04] Xml schema part 0: Primer second edition. `http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/`, October 2004.

[xml07] Xml path language (xpath) 2.0. `http://www.w3.org/TR/2007/REC-xpath20-20070123/`, January 2007.

[xml08] Extensible markup language (xml) 1.0 (fifth edition). `http://www.w3.org/TR/REC-xml/`, November 2008.

[YG08] Jie Yin and Mohamed Medhat Gaber. Clustering distributed time series in sensor networks. In *ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 678–687, Washington, DC, USA, 2008. IEEE Computer Society.

[YSI08] YSI Incorporated. *6-Series Multiparameter Water Quality Sondes - User Manual*, 2008. `https://www.ysi.com/DocumentServer/DocumentServer?docID=EMS_S_I`.

[ZMEM07] Jose-Luis Zechinelli-Martini and Ilian Elias-Morales. Modelling and querying sensor databases. In *ENC '07: Proceedings of the Eighth Mexican International Conference on Current Trends in Computer Science*, pages 138–148, Washington, DC, USA, 2007. IEEE Computer Society.