Toward Manageable Data Sources

Pekka SILLBERG [1]
Tampere University of Technology, Pori, Finland

Abstract. As information systems are producing vast amounts of data with ever increasing speed and diversity, the management of data is becoming an important part of gaining the information that we need. With this as the motivation, this paper proposes a Manageable Data Sources framework for the systematic management of data sources. The framework is derived from a new conceptual model of data processing: the Faucet-Sink-Drain model. The framework achieves two aims: the unification of data processing, and secondly, the componentization and decoupling of data processing related tasks. The framework is described and a reference architecture is laid out for the creation of a proof-of-concept implementation to solve the given use case.

Keywords. Framework, MDS, Conceptual model, Big Data, Data processing, Faucet-Sink-Drain model

1. Introduction

Information systems are producing data in ever increasing amounts, with greater and greater speed and diversity. At the same time, Internet of Things (IoT) devices are becoming smaller and more energy efficient, and require fewer resources to manufacture. These low power battery operated devices can stay connected to the network for up to ten years, or even longer [1]. This enables new kinds of applications, such as mounting sensor devices permanently into the structures of a building. As the quantity of devices grows larger, the more it increases the pressure on how the produced (big) data *can* and *should* be managed and utilized by the software. The ideal case is that it can be done in both a controllable and systematic way, while avoiding any error-prone ad-hoc implementations.

The use of patterns, frameworks, models, and re-usable code improves the quality of software. [2,3] By decomposing the system into smaller modules, the development of a piece of software becomes easier to modify and the re-usability of the code increases. For example, the Model-View-Controller (MVC) design pattern separates the application logic, data, and user interface into their own logical components. [4,5] To fully leverage the opportunities of big data in software engineering, we need to implement a decoupled and standardized component that is specialized in the handling of all data processing and searching-related requests.

Thus, the goal of this study is to identify a generic data processing model for creating a software framework to process data requests. The framework should make data

[1]Corresponding Author: Pekka Sillberg, Tampere University of Technology, Pori, P.O. Box 300, FI-28101 Pori, Finland; E-mail: pekka.sillberg@tut.fi.

processing easier and more organized, and improve the quality of the software on the data handling related operations. The framework must also be kept relatively simple. Simple and easily understandable concepts and terms may enable us to be more efficient when working with the big data challenges. Questions such as how to manage the ever increasing amount of data, how to utilize and visualize the data in a meaningful way, and how to programmatically extract knowledge and wisdom from the data are not necessarily answered in this study, but are nevertheless inspiring topics. The research question that this study attempts to solve can be formalized as follows:

> Does there exist a generic model or approach that enables the processing of all data in any given software application?

This paper proposes a new conceptual data processing model called Faucet-Sink-Drain model. The model is applied for a Manageable Data Sources (MDS) framework in order to create a systematic framework for the handling and processing of data. The framework covers the creation of data, includes the search, selection, and combination of real input data as well as any supplementary open data, and finishes with the outcome of the processed data (e.g., reports or visualizations). Furthermore, the framework gives the control of the data to the users by letting them choose and manage the desired data sources, data processors, and visualizations as best suits their need. The scope of this paper is to describe the model from the perspective of a single user and/or an instance, but there shall be no such limitation in the finalized framework.

The contents of this paper are as follows. Section 2 explains the background of the study; Section 3 concentrates on the specifications and definitions of the framework; finally, Section 4 concludes the paper and lists possible directions for future research.

2. Background

2.1. From Data to Wisdom

What is data and why do we need data, and where and how should we use it? Data is the raw building block of all information. This block, a symbol, represents the properties of objects and events [6]. Information is built upon data, knowledge is based on information, and wisdom requires knowledge [6,7]. Together they form a hierarchy, shown in Figure 1, oftentimes called the data-information-knowledge-wisdom hierarchy (DIKW). It is also known as the 'Knowledge Hierarchy,' 'Information Hierarchy,' or 'Knowledge Pyramid'. It is also referred to as the 'Wisdom Hierarchy' to promote the concept of wisdom over knowledge [7].

Literature has multiple definitions for each item in the DIKW hierarchy. In [7], the author reviewed 16 papers to find definitions and the essence of each of them. The author concluded that the difference between data and information is that plain data has no meaning as it is an unprocessed fact or observation. Information can be derived from the data by organizing it or by including a purpose or context, making it meaningful, useful, and relevant [7]. As a summary [7], knowledge can be seen as a combination of information, understanding, capability, experience, skills, and values. Knowledge can be differentiated into explicit and tacit knowledge, where the first is something that can be recorded programmatically but the latter is an embedded part of the human mind.

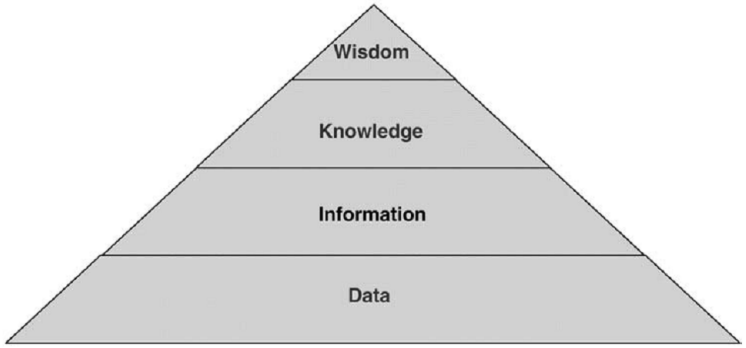

Figure 1. The hierarchy of data, information, knowledge, and wisdom (DIKW). [7]

2.2. Data Warehouse Design

The purpose of a Data Warehouse (DW) is to provide an architectural model to support the decision-making activities of an organization. It is achieved by extracting and storing variable data in a uniform format in a centralized database, which provides tools for querying, reporting, and information analysis. The definition by [8] says that "[a] data warehouse is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions." The data produced by different sources can be imported to the DW by utilizing Extraction-Transformation-Loading (ETL) processes. There are two main design styles for setting up a data warehouse: bottom-up and top-down [9]. In the bottom-up model, smaller logical sections called Data Marts (DM) are solved first, and then integrated into the DW. The data included in a DM is specific to a certain business problem, and knowing the Business Model (BM) helps to analyze what data should be included in it. [9,10] In the top-down model, the whole dataset is fitted into the DW, and then the smaller DMs are designed from data provided by the DW [9].

Designing a complete DW comprises several phases such as the design of conceptual and logical schemas or ETL processes. Each phase in turn may have several competing techniques and patterns, but a lack of formal design guidelines for a whole DW system has been identified [10,11]. The framework introduced in this study resembles the concept of DW in several ways. Incidentally, MDS was designed independently without prior knowledge of the similarities with DW. The key difference is that with MDS, the input data can be imported with more flexibility and with less design effort. Data marts in the DW are more static and unchanging representations of an identified business problem, but in MDS, any stream of data (or combination) can be analogous to a data mart.

2.3. Big Data

Big data is certainly an interesting field for harnessing by the MDS framework, as all kinds of appliances and IoT devices produce data. Big data has been identified by [12] as consisting of three characterizing Vs: volume, velocity, and variety. The original definitional qualities of big data have since been varied in several other ways, typically by addition of more Vs (e.g. [13,14,15]). The other Vs introduced in subsequent sources included terms such as veracity, variability, visualization and value. Originally, the three

Vs were "intended to define proportional dimensions and challenges specific to big data" [16] in the same way as width, height, and depth are magnitudes of measures in real life three-dimensional space. The additional Vs can be used to understand various important topics when working with big data, but they are not aspects unique to big data only.

If the big data is too much to handle there exist solutions to reduce it to smaller, more manageable data chunks by the creation of coresets, subsets of the original data. [17,18]. Other means of analytics, as well as views on next generation DW research are studied in [19]. From the viewpoint of the MDS framework, big data is simply a lot of data, and the framework should be able to import and to process it based on criteria defined by the user. The framework's key point is the ability to handle different kinds of data sources in a systematic and controlled manner, with the goal of generating a suitable information for creating reports and the most value to the user.

3. The Framework: Manageable Data Sources

This section describes the design ideas behind the MDS framework. It will also clarify the conceptual model of data processing (i.e., the Faucet-Sink-Drain model) used in this paper. The framework aims to help the processing, combination, and management of data sources, by making them controllable and systematic. With the systematical approach, the traceability of operations (i.e., what steps were taken to reach the result) should become more evident and reproducible.

3.1. Overview

Let us start by introducing the key concepts of the model shown in Figure 2. There are five core components, the first (topmost in the figure) being a *faucet*, which appears to be open, and running a couple of *streams*. These streams go into a *sink*, and eventually into a *sieve* (i.e., an filter on the bottom of the sink). The sieve is connected to the last component, a *drain*. In this figure, the faucet simply appears from somewhere whereas the drain disappears. They are not necessarily connected to each other, but they are most likely connected to different systems which are not in the scope of this figure. An extended version of this concept will be explained together with the meaning of the added components in Figure 3. Term "stream" in the context of this paper does not relate to term streaming of media.

How does the imaginary version of this rather mundane appliance compare to its real world counterpart? Firstly, the faucet can be seen as the source of the data. The running water is a collection of data streams, and the sink is used to store and display the data. The sieve is a filter of the data with the capability of selecting and processing any chunk of any given data stream. The drain is a piping system to remove the excess amount of data in order to prevent information overflow in the sink. The drain may also be used for looping the processed data to another faucet and sink for refining the data even more. Secondly, this appliance has far greater adaptability than the physical one as it is easy to add or remove components.

It is not enough merely to collect all kinds of data sources together and display the data produced by them. When the data is combined together and processed in a certain way, the result will provide more data, which in turn can be processed again

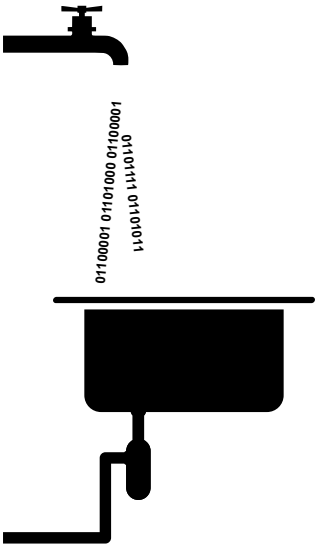

Figure 2. New conceptual model of data processing.

with some other data. If the data, selection criteria, and processing methods are chosen wisely, the data refinement cycle can eventually provide information that is valuable to its user. Therefore, the solution must be customizable and able to accommodate a feedback mechanism.

The framework strives to resolve this by being adaptable and able to grow seamlessly. Any number of data sources can be added, configurations for data visualizations and storage can be made based on stream criteria, and sieves can select and process any data stream. The extendability of the MDS framework is depicted in Figure 3, where an additional set of components (faucet, stream, sieve, and drain) have been "attached" to the sink to the sink (when compared to the starting point shown in Figure 2). From the software engineering and design point of view, the MDS framework also relies on the decoupling of data processing from the application logic in the same way as the MVC design pattern does with regard to model, view, and controller.

Let us examine Figure 3 more closely. The largest faucet located in the top left corner appears to be emitting two separate data streams. These two data streams[2] contain the following data pieces: `01100001 01101000 01100001` and `01101111 01101011`. In this case they are actually streams of binary data, usable by computers, but not so useful for humans. What could the user do to benefit from the generated data? The data can be refined in a series of loops, where each loop can improve the information value of the data. The feedback loop is the most essential part of this framework, because that is where all the data processing takes place in a systematic and controllable way.

A simple case of feedback processing is visualized on the right side of Figure 3. The added sieve is configured to select the "shorter" of the previous streams (`01101111 01101011`), process it, and forward the new processed data stream to a drain (piping system). If the drain is not connected to anything, the generated data will simply be dis-

[2]Streams actually have certain metadata attached to them, but they are omitted from the figure for the sake of simplicity.

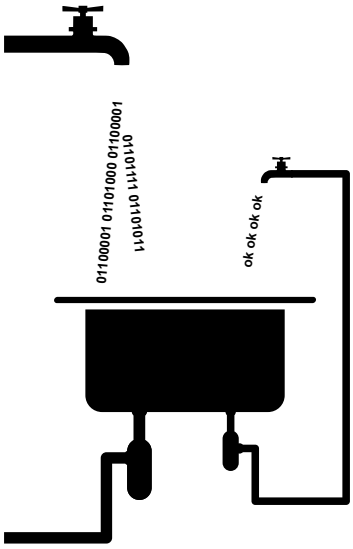

Figure 3. Extension of the conceptual model of data processing with a feedback mechanism.

carded. However, should we connect the drain to a new faucet, the data can then be directed to the data sink. The data stream that emerges from the faucet contains the following data: `ok ok ok ok`[3] which is an ASCII string representation of the previously selected binary data.

To recap, the MDS framework allows the addition of as many "feedback loops" as necessary. We can have any data stream (the old and the new) re-used, combined, and processed in additional sieves, which in turn create more data for additional faucets. If the chaining of the processed data is done wisely, the result will be useful information—and perhaps, knowledge.

3.2. Definitions, Glossary and Specifications

In this section, the framework's implementation-specific matters are dealt with. The key concepts and terms are defined as follows:

Data Component that forms the bulk of the content in the `DataSink`. It is basically of any type and format, and also includes the timestamp when the data was created. Each piece of data is associated to exactly one `DataStream` but the association is uni-directional, thus data is not aware of its owner.

DataStream Component which uniformly describes the type, format, and origin of the associated `Data`. Owns 0 to n `Data`, and is associated to exactly one `DataFaucet`. Not to be confused with with media streaming, or stream processing.

DataFaucet This component is the source of all content (i.e., data) and it also defines the physical characteristics of the source (e.g., location, update intervals and so on). The faucet has 0 to n `DataStreams`.

[3]The new stream in Figure 3 has four times more output compared to the input data; the reason for this is simply a more aesthetic look.

DataSieve Each `DataSieve` component has the ability to filter all data inside the sink, select the most interesting streams, and process and combine them into new streams. The processed output of the sieve—`DataStreams`—is transmitted to the attached drain. The sieve has exactly one `DataDrain`.

DataDrain The drain component is a transmission mechanism for the data. If left unconnected, the data will be discarded. Multiple faucets can be attached to a single drain, and all of these faucets will receive the same data. The drain may have 0 to n `DataFaucets`.

DataSink The central component where the data resides and can be observed. The sink will share the ownership of a `DataStream` once it has been emitted from the `DataFaucet`. Different kinds of `Visualizers` may be applied to any stream matching the defined `Criteria`. The sink may store and accumulate the data, or simply update the current data instance in the sink based on the `StreamOperator` defined on each stream. The `DataSink` knows 0 to n `DataFaucets`, and has 0 to n `DataStreams`. There must be at least one (1 to n) `DataSieve` to prevent "information overflow" in the sink.

DataProcessor A set of rules for processing `DataStreams` by the `DataSieve`. It defines any means necessary to create a new output data from the selected input data (e.g., addition, substraction, multiplication, logical AND & OR, set combination such as union & intersection, concatenations).

Criteria A set of rules for selecting the desired `DataStreams` that are available in the `DataSink`.

StreamOperator This controls how the `DataSink` needs to observe and handle the `Data` of a `DataStream` once the data has entered the sink. It says what needs to done to the data point to formulate a result, and how the result should be stored. For example, this could be a rule to accumulate the results for forming a time series of the data; to apply a differential and integral calculus to it; or to the continuous average and so on.

Type Definition of the (stream) type.

UUID Universally unique identifier.

Visualizer A visualizer may be used to create different views of the `DataStream` and the `Data` in the `DataSink`, such as documents, charts, maps, and reports. They are attachable and interchangeable, and do not modify the data.

Figure 4 depicts the reference architecture of the core components of the MDS framework. This high level view identifies the associations and quantitative relationships between these components. One important aspect is that this figure represents the case of a single user instance. This means that there may be other external components to which the components seen in Figure 4 are attached. For example, the `DataFaucet` may receive data from a drain created and defined by another user. Similarly, the `DataDrain` seen here can transmit its data to someone else's faucet.

Currently no implementations of the framework exist as the study is still in its early stages. Consequently, data formats regarding settings and description files have not been finalized, but the available techniques are constrained by the selected design criteria: compactness, splittability, exchangeability, and transferability. In practice, this means supporting at least the human-readable data serialization languages such as Extensible Markup Language (XML), JavaScript Object Notation (JSON), or YAML Ain't Markup

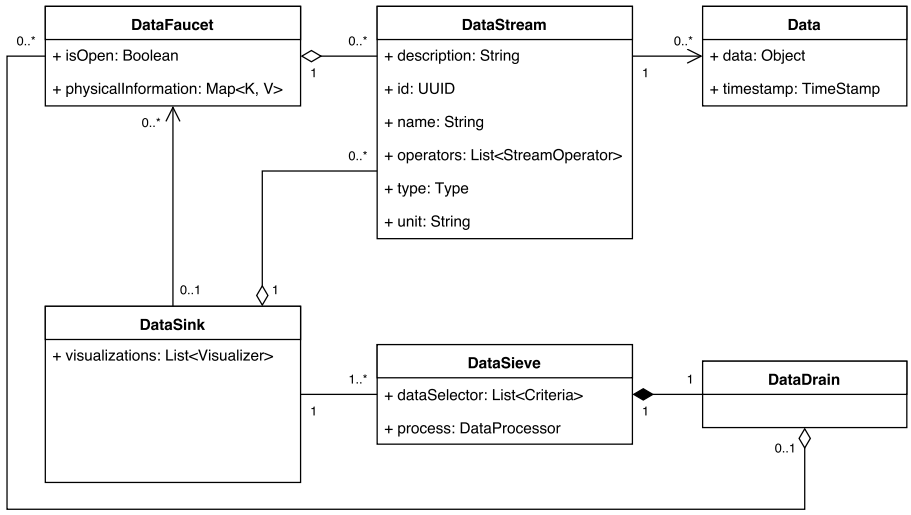

Figure 4. Reference architecture of Manageable Data Sources framework.

Language (YAML). Transferability, and easy sharing, would be important so that users could share their data sources with each other for more refined data, and information. This can be achieved by encoding the said files into barcodes or similar. If the files are splittable, it helps transferability, as individual file chunks can be fitted and shared with a storage medium such as barcodes or contactless integrated circuit cards (e.g., utilization of the NFC Data Exchange Format, NDEF). Being compact enough will allow the files to be fitted in a minimum amount of the previously mentioned media. The transmission of data does not have this kind of constraint, and it may use any means necessary to reach the system so that any internet connected device can be communicated with.

3.3. Discussion

The idea of the MDS framework has been introduced but there has not yet been any discussion of where it could be utilized. The use case scenarios described here are potential targets that might benefit from utilizing the MDS framework. Once the framework reaches the proof-of-concept maturity level, it should be benchmarked against at least one of the use cases.

Real-time Data The cars of today produce a lot of data and information with their electronic devices and sensors. A multitude of control units continuously monitor for the data in the Controller Area Network (CAN) bus. However, a fair share of that data cannot be accessed by the end user. Even if the on-board computer of the data can display information, it is typically limited to a pre-selected, non-customizable list of variables. The data is there, but the users have no means to access it without third party extensions and to make their own decision on what information to display. Ultimately, it is my car, and my data, and I want to have access to it.

So let us say that users could customize their access to their car's systems to select, process, and display any data produced by their car. OK, that would be nice, but where to next? There are multiple data sources in the car itself, as well as multiple other devices to

display data (e.g., dashboard, in-vehicle infotainment systems, laptops, mobile phones, and smart watches). In addition, there may be several users for the same car, but they may have different perspectives on the information that they need. An even more extended case is where the user has several cars—perhaps a fleet of cars—where they would like to have similar settings, preferences, and reporting features.

This is where the MDS framework comes in—by enabling the user to have their own systematically defined specification of what is important for them. The user's own pool of data, preferences, visualizations, reports and so on can follow them wherever they go, and allow them seamless integration with any available data source (i.e., by attaching their own `DataFaucet` to any openly available `DataDrain`). In an ideal case, the user would be able to (1) attach new data sources to their sink, (2) process and combine the incoming data, (3) refine the available data until the desired goal has been achieved, (4) control the results of the process (e.g., reports and visualizations), and (5) be able to release the results to other users.

Web Data Sources The data sources available around the Internet might not be providing a real-time data, but nevertheless, data processing and data source management could be a useful tool on monitoring of these sources. The following examples are chosen because I have had the opportunity to contribute in the development of the mentioned systems. In the first system [20], a third party Open Data source is being used to detect dangerously high heavy metal concentrations in water. This is a relatively simple use case to implement as the results of calculations are displayed in a tabulated format. Similarly, the second example [21] is quite a simple data collection and visualizer system, that has been used to gather the usage data of water, electricity and heat consumption in a public swimming pool in the City of Pori. In this case scatter and bar charts are used to visualize the consumption values. Both of the use cases could benefit from further data processing, such as the trend detection or implementation of any new analysis method that was not implemented in the original user interface. In order to achieve this, the data source as well all the chain of currently applied analysis methods should be transferable to another environment (for example, import from the application website to the user's personal computer). This transferable processing information, and the previously defined steps from 1 to 5, can be called a Faucet-Sink-Drain recipe.

Data Protection In the wake of General Data Protection Regulation (GDPR), the proposed framework could be even seen as a part of the underlying operating system where all data accessed by different applications require an authorization from the user first. The user may select which kind of data the application may read and process, and is that application allowed to make changes (e.g., add or remove) to the system-wide data or merely to the clone of the data that resides isolated in the application space. Basically, the operating system governs the biggest `DataSink` containing all the data, and each application have access to a "smaller" `DataSink` (i.e., the subset of the data). Applications can be seen as `DataFaucets` and/or `DataDrains` depending on their usage of the data. To be able to access the data on `DataSink`, the application must publish a list of the required and the optional data (i.e., `DataStreams` captured by `DataSieve`). The application developer would have to compose the list, and indicate if the application can not operate without the requested data. For example, a navigation application would probably be rather useless without location information, but the user might simply want to use that application because of the maps it can provide; the developer might accom-

modate to the user's wishes and offer a way to use the application without the navigation feature. Alternatively, the user might want to feed dummy location information to the application, but the application developer does not allow the use of application in this kind of situation.

4. Conclusion and Future Work

In this paper, I presented a new conceptual model of data processing and a framework called Manageable Data Sources for systematic management of different sources of data. The aim of the framework is to make data processing easier and more organized, and to improve the quality of the software on the data handling related operations.

The concept consists of five basic components: (1) faucets, (2) streams, (3) sink, (4) sieves, and (5) drains. The concept of MDS relies on the decoupling of the data processing component from the application logic in the same way as MVC design patterns with regard to model, view, and controller. Data processing in MDS is a continuous "feedback loop" of data, in which the data is refined bit by bit in order to provide more data and valuable information to the user. MDS is customizable to meet the requirements of different users. Finally, an example use case showing the potential for future research in the data source management field was introduced.

Future work will focus on the implementation of a proof-of-concept application which can be used to perform experiments and to collect evidence for evaluating the framework. The proof-of-concept implementation of the framework is to be evaluated by at least one of the designed methods, such as the Architecture Trade-off Analysis Method (ATAM) to find any possible defects and design flaws as early as possible [5,22]. Also, the applicability of the model in the example use cases requires more evidence and experiments.

References

[1] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low Power Wide Area Networks: An Overview," *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 855–873, 2017.

[2] T. T. Nguyen, *Improving software quality with programming patterns*. PhD thesis, Iowa State University, 2013.

[3] J. Greenfield and K. Short, "Software factories: Assembling applications with patterns, models, frameworks and tools," in *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '03, (New York, NY, USA), pp. 16–27, ACM, 2003.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture, A System of Patterns*. Wiley, 2013.

[5] K. Koskimies and T. Mikkonen, *Ohjelmistoarkkitehtuurit*. Talentum, 2005.

[6] R. L. Ackoff, "From Data to Wisdom," *Journal of Applied Systems Analysis*, vol. 16, no. 1, pp. 3–9, 1989.

[7] J. Rowley, "The wisdom hierarchy: representations of the DIKW hierarchy," *Journal of Information Science*, vol. 33, no. 2, pp. 163–180, 2007.

[8] W. H. Inmon, *Building the Data Warehouse*. New York, NY, USA: John Wiley & Sons, Inc., 1992.

[9] D. Bourgeois, *Information Systems for Business and Beyond*. The Saylor Foundation, 2014.

[10] S. Luján-Mora and J. Trujillo, "A Comprehensive Method for Data Warehouse Design," in *Proceedings of the 5th Intl. Workshop on Design and Management of Data Warehouses (DMDW'03)*, pp. 1–14, 2003.

[11] V. Peralta and R. Ruggia, “Using Design Guidelines to Improve Data Warehouse Logical Design,” in *DMDW*, 2003.

[12] D. Laney, “3-D Data Management: Controlling Data Volume, Velocity and Variety.” https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf, February 2001. Accessed January 18, 2018.

[13] M. Van Rijmenam, *Think Bigger: Developing a Successful Big Data Strategy for Your Business*. AMACOM, 2014.

[14] IBM, “Extracting Business Value from the 4 V’s of Big Data.” http://www.ibmbigdatahub.com/infographic/extracting-business-value-4-vs-big-data, January 2016. Accessed January 24, 2018.

[15] R. J. Self, “Impact of the "12 Vs" of Big Data on Questions of Ethics, Trust, Stewardship and Governance of Analytics.” http://computing.derby.ac.uk/c/wp-content/uploads/2014/11/Self_Richard_A2014.pdf, 2014. Accessed January 24, 2018.

[16] D. Laney, “Batman on Big Data.” https://blogs.gartner.com/doug-laney/batman-on-big-data, November 2013. Accessed January 18, 2018.

[17] D. Feldman, M. Schmidt, and C. Sohler, *Turning Big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering*, pp. 1434–1453. Society for Industrial and Applied Mathematics, 2013.

[18] D. Feldman, M. Volkov, and D. Rus, “Dimensionality Reduction of Massive Sparse Datasets Using Coresets,” in *Advances in Neural Information Processing Systems 29*, pp. 2766–2774, Curran Associates, Inc., 2016.

[19] A. Cuzzocrea, I.-Y. Song, and K. C. Davis, “Analytics over Large-scale Multidimensional Data: The Big Data Revolution!,” in *Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP*, DOLAP ’11, (New York, NY, USA), pp. 101–104, ACM, 2011.

[20] P. Sillberg, C. Veesommai, J. Soini, and H. Jaakkola, *Web-User-Interface System Utilizing rHMEI and Open Data for a Water Quality Analyzer*, pp. 420–428. Frontiers in Artificial Intelligence and Applications, Netherlands: IOS Press, 2018.

[21] J. Soini, P. Sillberg, and P. Rantanen, “Prototype system for improving manually collected data quality,” in *Proceedings of the 3rd Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, SQAMIA 2014, September 19-22, 2014, Lovran, Croatia*, pp. 99–106, 2014.

[22] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. SEI series in software engineering, Addison-Wesley, 2002.