ONTOLOGY-BASED DATA QUALITY FRAMEWORK FOR DATA STREAM APPLICATIONS

(Completed Academic Paper)

Sandra Geisler, Sven Weber, and Christoph Quix
Information Systems, RWTH Aachen University, Germany
{geisler,weber,quix}@dbis.rwth-aachen.de

Abstract: Data Stream Management Systems (DSMS) have been proposed to address the challenges of applications which produce continuous, rapid streams of data that have to be processed in real-time. Data quality (DQ) plays an important role in DSMS as there is usually a trade-off between accuracy and consistency on the one hand, and timeliness and completeness on the other hand. Previous work on data quality in DSMS has focused only on specific aspects of DQ. In this paper, we present a flexible, holistic ontology-based data quality framework for data stream applications. Our DQ model is based on a threefold notion of DQ. First, content-based evaluation of DQ uses semantic rules which can be user-defined in an extensible ontology. Second, query-based evaluation adds DQ information to the query results and updates it while queries are being processed. Third, the application-based evaluation can use any kind of function which computes an application-specific DQ value. The whole DQ process is driven by the metadata managed in an ontology which provides a semantically clear definition of the DQ features of the DSMS. The evaluation of our approach in two case studies in the domain of traffic information systems has shown that our framework provides the required flexibility, extensibility, and performance for DQ management in DSMS.

Key Words: Data Quality, Data Streams, Data Stream Management, Ontologies

INTRODUCTION

The amount of data that is available and has to be processed defines new challenges for data management solutions today. Moreover, many systems produce continuously new data, e.g., social networking applications such as Twitter, tickers with latest stock prices, sensor networks, or traffic management systems with recent data about the traffic state on various roads. Also, users are usually only interested in the most recent information and hence, the data has to be processed and analyzed in real-time. Data Stream Management Systems (DSMS) [1] offer full-fledged data querying and analysis capabilities and at the same time operate in real-time. Data streams are continuous, potentially unbounded and produce data at a high rate [1]. Queries have to be continuously evaluated while data is passing by (also called *human-passive machine-active approach* [2]). Therefore, query processing and data mining algorithms have to be adapted to fulfill the specific needs of streaming data.

Data quality (DQ) plays an important role in DSMS as there is usually a trade-off between accuracy and consistency on the one hand, and timeliness and completeness on the other hand. Especially in sensor networks in which a large number of sensors are deployed, the balancing between the costs of the sensors (including energy and communication costs) and the quality of the data is crucial. Sensors for measuring temperature, light, acceleration, or ground motion can be purchased for very low prices, but it happens very frequently that such sensors deliver incorrect measurements as they are influenced by environmental disturbances. In the worst case, sensors can just stop working (because of wear, damage, or empty battery). The data has therefore to be integrated from several sensors in the same area to get a coherent picture of the situation which might result in the desired DQ properties (e.g., accuracy, consistency, timeliness). Only if the data has the required quality, it should be used for further processing (e.g., notifying users of certain detected events). Thus, it is very important that DQ is a key concept in DSMS.

Previous works on data quality in data streams focused more on system aspects, i.e., Quality of Service

(QoS). QoS dimensions for DSMS are classified in time-based (e.g., throughput, output delay) and content-based (e.g., sampling rate, data mining quality) dimensions [3]. The Aurora System [2] defines in addition a value-based dimension, e.g., certain results are prioritized based on their values. DSMS which are aware of QoS dimensions (e.g., Aurora and QStream [4]) can adapt their operation to guarantee that the required QoS is achieved. A limitation in these approaches is that the QoS parameters are only provided for the output streams. Other approaches, such as Borealis [5] and the approach by Klein and Lehner [6], manage data quality at the operator level. The implementations of query operators are modified to maintain quality information throughout the query process. Quality information that is included in the input streams is propagated to the output by providing data quality functions which compute a new data quality value based on the input values. The main disadvantage of such an approach is the tight integration of the data quality management and the DSMS, which, e.g., hampers the extension of the data quality model with new dimensions. Furthermore, application-specific DQ functions cannot be easily introduced as it would require changing the underlying DSMS.

In this paper, we present a *flexible, holistic, ontology-based data quality framework for data stream applications*. The core of the DQ framework is an ontology in which we manage all DQ related metadata, such as the data sources, their DQ factors, DQ metrics, and so on. A key feature of our approach is the distinction between three types of DQ metrics: (i) *content-based metrics*, (ii) *query-based metrics*, and (iii) *application-based metrics*. Content-based metrics use semantic rules and functions to measure data quality, e.g., consistency is low if one sensor states that a road is icy while data from another sensor indicates a temperature of 20°C. Query-based metrics are methods to compute the DQ for certain query operators, e.g., the accuracy of an aggregation operator is the average accuracy of the input tuples. We use a query rewriting approach to offer more flexibility than the operator-based approaches mentioned above. Finally, application-based metrics measure DQ by any kind of application-specific functions, e.g., confidence values of a data mining classifier are provided by the data mining system. These different types of metrics allow a very broad range of DQ metrics. In particular, the methods for computing data quality are not fixed. As all DQ-related metadata is stored in an ontology, we follow an ontology-driven (or metadata-driven) approach for DQ management.

We evaluated our approach in the domain of traffic information systems. Intelligent traffic services (ITS) rely on data acquired by stationary or mobile sensors. Since more and more sensors are integrated in vehicles and in the traffic infrastructure, the amount of data is also rising. Cheap, ubiquitous communication facilities with high bandwidths, such as 3G/4G cellular networks, allow transmitting high amounts of data at high rates from vehicles to other vehicles or the traffic infrastructure, subsumed under the term Car-2-X communication, and also to central instances, such as traffic management centers. It is apparent that this huge amount of data is useful in many traffic applications, e.g., safety relevant applications such as intersection assistance and queue-end detection [7], but also for less critical applications such as traffic state estimation. Measuring the DQ in traffic applications is important especially for user acceptance of such methods. If a queue-end detection mechanism delivers too many false positives, users tend to ignore such “noisy” systems. The application context of our case study is the CoCarX project[1], in which several companies from the mobile communication sector and the automotive sector investigate the suitability of UMTS/LTE technologies for direct, targeted transmission of traffic data arising from both stationary and vehicle-based sensors. We have implemented the quality framework as an extension of the DSMS Global Sensor Networks (GSN [8]).

The structure of the paper is as follows. The next section discusses related work in the area of DQ management for data streams. Section 3 introduces the ontology-based data quality framework by stating the requirements and presenting the overall architecture. Section 4 presents the components related to the ontology. Section 5 then details the three types of data quality services, namely the *content-based* quality service, the *query-based* quality service, and the *application-based* quality service. In Section 6, we show the results of two case studies in traffic management: queue-end detection and traffic state estimation. Finally, Section 7 concludes our paper and points out future work.

[1] http://dbis.rwth-aachen.de/cms/projects/CoCar

RELATED WORK

DSMS can implement means to monitor the system performance during query processing and adapt the system configuration if certain quality criteria are not met. This is usually summarized under the term *Quality of Service* (QoS). For example, if the system is overloaded and the output delay or the throughput is low, a DSMS can drop tuples with sampling techniques (e.g., in the Aurora system [2]). In the QStream system [4], two descriptors for each output stream are defined - a content quality descriptor which includes the dimensions inconsistency and signal frequency defined by Schmidt [3], and a time quality descriptor consisting of values for data rate and delay. The quality dimensions are calculated throughout the whole query process. All operators in the query execution plan comprise functions used to calculate the value of each quality dimension when new tuples are processed. At the end of the processing chain quality values are output for the result streams of each continuous query. A user can pose requirements on the quality of the result streams by formulating a request describing thresholds for the dimensions. If the descriptors meet the quality request, this is reported to the user as a successful negotiation.

In Borealis [5], the QoS model of Aurora has been refined to rate QoS not only for the output, but also for each intermediate operator. To this end, each tuple includes a vector with quality dimensions, which can be content-related (e.g., the importance of an event) or performance-related (e.g., processing time for an event up to this operator). The vectors can be updated by operators in the query execution plan and a scoring function is provided [5]. A crucial limitation of the previous approach is that the quality dimensions in the vector are equal for each stream, which does not allow for an application-specific DQ rating of stream contents.

To achieve a more fine-granular rating of DQ on attribute, tuple and window level and to also include application-specific DQ dimensions, Klein and Lehner [6] extended the PIPES system [9] with modified operators. DQ information becomes thereby a part of the stream schema. Klein et al. distinguish four different types of operators based on the operator's influence on the stream data (modifying, generating, reducing or merging operators). Changes on the data can in turn result in updates of the DQ of an attribute, tuple, or window. In addition, the size of their DQ windows (the part of the stream for which data quality is evaluated) is dynamically adaptable based on an *interestingness factor*, e.g., the window size is decreased when there are interesting peaks in the stream data. A drawback of the approach is the deep integration of DQ features into the operators. The implementation of operators has to be changed substantially to include the quality information.

A logical consequence from identifying data quality problems is the elimination of these problems. In [10] the authors extend the Global Sensor Network (GSN) system by a data cleaning component. They identify outliers in the data by mathematical models. While this approach is suitable for numeric data, it does not take into account other types of data, such as nominal data. It also does not reflect a variety of quality problems, but focuses on the problem of outlier detection. Jeffery et al. [11] propose an extension of SQL to include automatic data cleaning tasks in the DSMS based on temporal and spatial characteristics of the data. The system supports detection of outliers, missing data and duplicate readings for single stream elements and windows (grouping and aggregating over time and/or space), but the user is restricted to the built-in data quality dimensions, assessments and cleaning mechanisms.

Probabilistic data streams represent the uncertainty that a tuple exists *(tuple existence uncertainty)* or that an attribute has a certain value *(attribute value uncertainty)* [12]. Uncertainty has been studied also for database management systems [13]. In the literature, mainly tuple existence uncertainty for data streams is addressed. To model uncertainty for attribute values, for each attribute of a stream a random variable with a corresponding distribution function has to be introduced [12]. The tuple existence uncertainty can be modeled by a binary random variable. While the approaches before assumed possible stream semantics for streams with discrete data, these semantics is not valid for streams with uncertain continuous data. The PODS system [14] addresses this issue by modeling each continuous-valued attribute as a continuous random variable with a corresponding probability density function.

ONTOLOGY-BASED DATA QUALITY FRAMEWORK

In this section, we will first detail the requirements on which we founded the design our framework. Subsequently, an overview of the core components will be given.

Requirements

As discussed in the previous section, the existing approaches for DQ management in data streams focus either on a limited set of DQ dimensions, and/or are very tightly integrated with the operation of the DSMS which hampers the extension of the DQ framework. DQ measurements can be very complex and might require more expressivity than offered in a simple SQL query. For example, in traffic information systems, positions from GPS sensors (Global Positioning System) have to be matched to a particular part of a road. This procedure is called map matching. Only the map matching method itself can provide the information whether a position was matched exactly to one road or whether it was matched only with a low confidence because the position is in the middle of two roads. Such applications require application-specific DQ metrics and DQ dimensions. Therefore, we aimed at a DQ framework which is easily extensible and is not restricted to certain quality dimensions.

Furthermore, DQ management in data streams should be holistic and not only focused on the quality of the output stream. It is important to monitor DQ throughout the whole DSMS: incoming tuples may contain already quality information which needs to be maintained as the data "flows" through the DSMS. Additionally, new DQ factors have to be added which can be computed based on existing DQ values, by application-specific function, or by semantic rules. Thus, to provide quality information in each data processing step, a framework must facilitate the extension of data stream elements with DQ values. DQ values provide additional information that must reflect the contents and characteristics of the data stream and its elements and should rate it according to different DQ dimensions.

DSMSs belong to the group of real time systems [15], which are operating in narrow time frames and memory bounds. Hence, the integration of DQ processing needs to satisfy real time requirements and may not pose significant additional overhead to memory usage or CPU time. Furthermore, the DQ rating should be optional - if DQ information is not required or desired, it should be possible to turn off the DQ features without affecting the data processing.

Since DQ is multi-dimensional (i.e., an element might have several DQ values in different DQ dimensions), the derivation of a total DQ value based on individual DQ values should be possible. This computation should be done by user-defined evaluation formulas. Users can then adjust the computation of the total DQ value according to their needs and personal preferences.

Finally, to allow easy customization and extension, the DQ framework must be metadata-driven, i.e., the processes for measuring DQ should be configured by metadata that is managed in a central component. This enables also the separation of the definition of the DQ-related functions and the processing of data streams. Based on these requirements, we designed an ontology-based DQ framework which is detailed in the following.

Framework Architecture

As discussed in the introduction, we identified three major aspects in data stream applications and data stream processing, for which the DQ of elements and attributes has to be calculated or recalculated. First, the semantic content of the data stream and its elements should be rated. This includes that we must be able to define semantic rules, which describe the acceptable ranges for values of one attribute or values of combinations of several attributes in one data stream element. For example, a rule could define an upper bound for speed measurements of vehicles. If a vehicle exceeds a speed of 280 km/h, the DQ value for this attribute should be very low. DQ measurements of this type are done by *Content-based Quality Services*. Second, when rating the DQ in each step of the data processing, also the query processing operators (e.g., join, aggregation) in a DSMS have to be taken into account, as they can alter the DQ values of stream elements flowing through the system. Hence, a framework should incorporate means to analyze the queries and recalculate quality values accordingly. We assign these tasks to the category

Query-based Quality Services. Finally, in data stream applications customized and application dependent data processing and analysis is implemented which may also add or change DQ values. These tasks are summarized under the term *Application-based Quality Services*.
We propose a DQ framework which extends a DSMS, i.e., the DQ Framework is an (optional) add-on to the DSMS. Basically, it is designed such that any extensible DSMS could be used. In this paper we adhere to relational DSMS and the well-known multiset or bag semantics for data streams [16]. Hence, a single data stream is defined by a set of attributes (the schema) and consists of data stream elements or tuples adhering to the schema. When incorporating DQ, the set of data stream attributes can be extended by a set of DQ attributes, where each attribute represents a DQ dimension. We define these streams as *quality-affine data streams*. Syntactically, quality attributes are not distinguishable from data attributes. They also are described by a name and a domain. Semantically, they are handled differently and must be recognizable during data processing. For instance, a Car-2-X message data stream (the messages include information from vehicle sensors) could be extended by DQ information in the following way:

$$\texttt{Message}_q = (\texttt{ID, Latitude, Longitude, Speed, Acceleration, RoadID}\ \texttt{Timeliness}_q, \texttt{SpeedAccuracy}_q)$$

$\texttt{Timeliness}_q$ rates the currentness of the message and $\texttt{SpeedAccuracy}_q$ rates the error of the measured speed. In Figure 1 the overall architecture of the framework is depicted. The aforementioned quality services manage the DQ attributes in a data stream element. The metadata is managed in an ontology that provides a Semantic DQ Model. The Semantic Quality Provider acts as an interface to this ontology. The DQ Processor invokes the Content-based and Application-based Quality Services as required by the definitions in the ontology. The Query-Based Quality Service interacts directly with the Semantic Data Quality Provider and the data processor, as queries have to be modified in the data processing steps. All grey boxes are DQ components and are added by the framework to the DSMS. We will detail the components in the following sections.

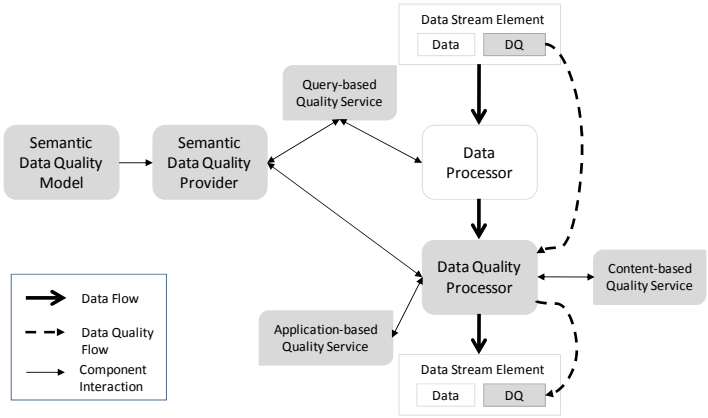

Figure 1: Architecture of the Ontology-based DQ Framework

SEMANTIC DATA QUALITY COMPONENTS

The *Semantic DQ Model* represents the basis for the DQ measurement. The model comprises a semantic description of information required for DQ assessment, including DQ dimensions and DQ metrics, linked to concepts in the domain of data stream applications. This allows the evaluation of DQ for tuples, windows, and attributes. We decided to use an ontology for structuring this semantic knowledge. The ontology enables us to define domain knowledge, DQ dimensions and metrics separately, and relate them to each other in a modular way. In the following, we will describe first the DQ dimensions and metrics used in our case study for traffic applications. Afterwards, we will detail the structure of the ontology and describe the components of the framework processing the ontology.

DQ Dimensions and Metrics

The type of the measured DQ is expressed by a *DQ dimension*. For example, the timeliness of data is a DQ dimension. For each DQ dimension a *DQ metric* defines a possible way to measure the quality within a dimension. There exists a plethora of classifications which structure and describe DQ dimensions, such as the Total DQ Management (TDQM) classification [17] [18], the Redman classification [19], or the Data Warehouse Quality (DWQ) classification [20]. For data streams, Klein and Lehner propose a dimension classification in [6]. In Table 1 we list a non-exhaustive set of DQ dimensions, which we think are of importance for DQ rating in a DSMS. Especially, we selected dimensions which we deemed relevant for traffic applications (based on [6] [21]). Other dimensions may be also relevant and our system is not restricted to the selected dimensions. Using a semantic data model, we are able to model an arbitrary set of dimensions and corresponding metrics. We distinguish two categories of dimensions. *Application-based DQ dimensions* rate the semantic content of data that may be dependent on the application. In addition, the DSMS performance has to be considered for the quality of query results. For example, when the output delay in the DSMS is too high, the estimated traffic state will identify a traffic situation that is no longer present. Furthermore, if too many tuples are dropped by a load shedder to gain performance, statements based on a low number of tuples may also harm the result. Therefore, also the Quality of Service (QoS) for multiple performance aspects of a system has to be tracked and taken into account in query processing. Hence, we define DQ dimensions related to the system performance as *system-based DQ dimensions.* There are also dimensions which are in both categories. DQ for a dimension can be measured on different levels. It can be measured system-wide, e.g., an output rate, on operator level, e.g., the selectivity of an operator, or on window, tuple, or attribute level.

DQ Dimension	Informal Description	Example Metric	Application-based/System-based
Completeness	Ratio of missing values or tuples to the number of received values/tuples	The number of non-null values divided by all values including null values in a window	Application-based
Data Volume	The number of tuples or values a result is based on, e.g., the number of tuples used to calculate an aggregation	Quantity of tuples in a window	System- and Application-based
Timeliness	The age of a tuple or value	Difference between creation time and current system time	System and Application-based
Accuracy	Indicates the accuracy of the data, e.g., a constant measurement error or the result of a data mining algorithm	An externally calculated or set value	Application-based
Consistency	Indicates the degree to which a value of an attribute adheres to defined constraints, e.g., if a value lies in certain bounds	Rule evaluation	Application-based
Confidence	Reliability of a value or tuple, e.g., the confidence to have estimated the correct traffic state	A weighted formula that is calculated from values for other DQ dimensions	Application-based

Table 1: Example DQ Dimensions

Data Quality Ontology

An ontology is a well suited tool to (1) model the knowledge about DQ concepts as well as about domain concepts and their relationships to each other, (2) modularize the knowledge and make it reusable, (3) be changed by users due to its human-readability and availability of a wide range of tools, and (4) be used as a configuration in an automatic process of evaluating DQ. In respect to the overall framework architecture, the DQ ontology has been modeled to link the components in data stream applications (tuples, windows, attributes) where DQ evaluation is desired with the corresponding quality dimensions and metrics. The DQ Ontology can be also seen as a metamodel for DQ information; it has been inspired by the DWQ metamodel for DQ management in data warehouse systems[20]. It differs to the DWQ metamodel in that it includes concepts from the data stream world and does not include data warehouse related concepts. Further it is designed along the trisection of our approach. Figure 2 delineates the structure of the DQ ontology, where each box represents a concept. Concepts highlighted in gray color describe the scope in data stream applications, e.g., data stream windows or attributes of data stream elements. White-colored concepts illustrate the DQ assessment components including metrics to rewrite SQL queries, evaluate semantic rules, as well as applying user-defined operations to applications. Dashed arrows denote object properties (i.e., relationships between two concepts defined by a reference to the related concept in a property of the other concept). The solid arrows represent isA-relationships, i.e., they denote the inheritance of one concept from the other.

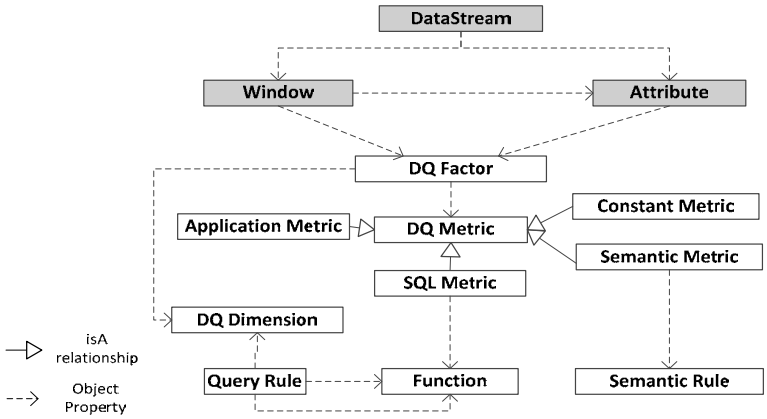

Figure 2: Visualization of the DQ Ontology

The concept of DQ factors, which has been adopted from the DQ evaluation methodology for data warehouses [20], provides the linkage of *DQ Metrics* and *DQ Dimensions* to DQ assessment tasks in the scope of data stream components, such as an attribute in a stream element. As mentioned before, a *DQ Metric* describes the way how a DQ value for a DQ dimension is measured. The metrics are categorized according to their purpose and the way and type of calculation. A *Semantic Metric* is related to one or multiple semantic rules. A *Semantic Rule* can be expressed by arbitrary mathematical expressions, e.g., Boolean expressions or arithmetic expressions (depending on the power of the mathematical expression parser used to evaluate the expressions during online quality assessment in the system). For example, we can define a Semantic Rule instance, which limits reasonable speed values by an expression `speed < 180`.

To characterize intrinsic properties of the data (e.g., according to EU directives[2] we fixed the intrinsic measurement error of a speedometer to 0.9), a *Constant Metric* can be defined whose instance will include constant values for a DQ dimension. The *Application Metric* concept provides an interface for user-defined functions and corresponding DQ dimensions. The calculation rule of an *Application Metric*

[2] http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31997L0039:EN:HTML

is not defined in the ontology – the value of its dimension will be calculated in the user-defined code in the DSMS. Finally, the *SQL Metric* accounts for two cases: (1) a DQ dimension is introduced, which is calculated using SQL operators (e.g., completeness), and (2) the changes of DQ values during the processing of SQL queries (e.g., when joins or aggregations are used in the query). Each *SQL Metric* instance is related to one or more *Function* instances, which can identify either the SQL operator which changes the quality value (*input*) or the SQL expression used to calculate the new DQ dimension value (called *output* or *replacement* function). The rewriting of SQL queries with the *input functions* using the defined *output functions* is defined by a *Query Rule* instance. These rules link DQ dimensions, SQL operators and functions to replacement functions. We will describe the rewriting of queries in the Section "Data Quality Services" below. Figure 3 outlines an example usage of an SQL metric to assess the completeness of an attribute in a relational tuple of a window. The stream attribute `Speed` is assigned with a DQ factor that links the DQ dimension `Completeness` with a SQL metric. `Completeness` is defined by an expression consisting of arithmetic operators and SQL functions (`count/all`). The variables `count` and `all` are references to functions. Each function has an SQL Mapping, which denotes the SQL expressions inserted into the query to calculate the variable value. The placeholder # in the definition of the mapping is replaced by the corresponding attribute name (in this case `Speed`).

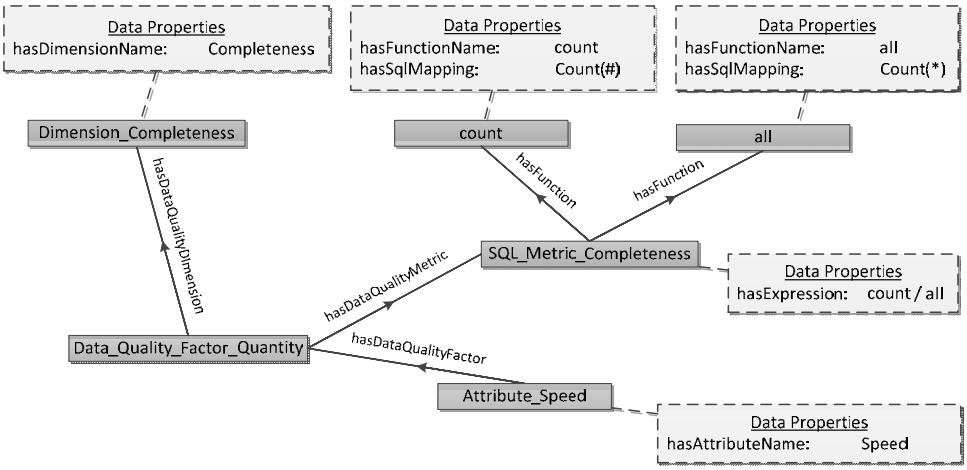

Figure 3: Illustration of a SQL DQ Metric in the ontology

In the following, we describe how the ontology is loaded during initialization of the system and provided to the DQ components of the framework. The definition of the attributes, sensors, dimensions, metrics, and rules has to be done only once for an application. Due to the modular design of the ontology it is possible to reuse instances for dimensions, metrics and so on.

DQ Information Provision and Distribution

The *Semantic DQ Provider* represents the interface between the DQ evaluation information that has been described in the semantic DQ model and the DQ processor, which will be outlined subsequently. It loads the ontology into main memory during initialization of the DSMS or when a continuous query is added or changed. Thereby, the required DQ information is constantly and fast accessible during the permanent DQ assessment process. The *DQ Processor* retrieves the DQ information from the *Semantic DQ Provider*, and enables the real time DQ processing in the DSMS. The processor analyzes the registered data stream components and looks up the corresponding DQ dimensions and metrics in the *Semantic DQ Provider*. For each DQ dimension and attribute a field is added to the data stream elements at hand (the stream is converted to a quality-affine stream). The processor identifies which metric is used for each DQ attribute. It delegates the processing of the *Constant Metrics, Semantic Metrics,* and *Application Metrics*

to the corresponding service components (*Application-based Quality Service* and *Content-based Quality Service)*. The *SQL Metrics* are processed by the Query-based Quality Service, which directly communicates with the *Semantic DQ Provider*. The Quality Services are detailed in the subsequent section.

DATA QUALITY SERVICES

As the framework focuses on the optional extension of a relational DSMS with DQ assessment features, all underlying components that contribute to the data processing and are part of the used DSMS are subsumed under the term *Data Processor*. This comprises the data manipulation and execution of queries (rewritten and original queries) in the system. The *Data Processor* acquires the stream tuples and propagates them to the system. When DQ processing is enabled, it delivers tuples to the DQ Processor, which will calculate DQ values according to the loaded metrics.
According to the three aspects identified for data processing, accordingly, three different DQ assessment services have been designed. The *DQ Services* are divided into two groups. First, the *offline* DQ processing is executed in the initialization phase of the system, which performs the rewriting of queries corresponding to the SQL Metrics that have been semantically modeled. Second, the *online* DQ services perform real time evaluation by semantic rules and integrate DQ measurement results from applications.

Query-based Quality Service

Since the relational DSMS allows data processing and filtering by relational operators, this also has to be taken into account for DQ measurement. Parsing and analysis of queries allows identifying operators that have influence on DQ values in data processing steps, such as aggregations, joins, or selections. Furthermore, new DQ attributes which can be calculated by SQL expressions, can also be inserted by rewriting the initial query. For that reason, in the initialization phase of the DSMS or when registering a new query, the *Query-based Quality Service* rewrites the continuous queries. These queries are executed by the *Data Processor* of the DSMS that correspondingly consider and process DQ information in stream elements. By this, we enable the system to handle data and quality attributes and thereby, data streams and quality-affine data streams, equally. For instance, if the number of Car-2-X messages used to calculate the average speed for a road is considerable, an SQL Metric `speed_datavolume` would be defined for the attribute `speed`. Analogue to the example from Figure 3, a variable `count` and an SQL Mapping `COUNT(#)` is defined, which denotes how the query should be rewritten to calculate the value of an attribute `SpeedDatavolume`. Based on this metric the original query `Q1` (see listing below) would be rewritten to a query `Q2`.

```
Q1:    SELECT RoadID, AVG(Speed) FROM message GROUP BY RoadID

Q2:    SELECT RoadID, AVG(Speed), COUNT(Speed) as SpeedDatavolume_DQ FROM message
       GROUP BY RoadID
```

Content-based Quality Service

In contrast to the rewritten queries, this service is executed as data stream elements arrive and when corresponding metrics and dimensions are available. The *Content-based DQ Service* computes DQ values based on data values in the stream elements' fields. These calculations are based on the evaluation of semantic rules of *Constant Metrics* and *Semantic Metrics* in the ontology. The rules are expressed by mathematical expressions using the name of attributes as variable identifiers. The total results of the calculations are inserted into the corresponding DQ attributes. The *Semantic Metrics* also allow defining rules using multiple attributes from one stream element. Equation 1 exemplifies a semantic rule that has been defined to identify speed values below or above reasonable bounds.

$$SpeedConsistency_q(Speed) = 0 \leq \text{Speed} \wedge \text{Speed} \leq 280 \quad (1)$$

Application-based Quality Service

The *Application-based Quality Service* does not perform quality assessment based on information from the ontology, but provides an interface for user-defined functionality to add DQ values to every data stream element. Since the definition of neither semantic rules nor the evaluation by query rewriting is a sufficient alternative to evaluate the impact of user code to DQ, embedding quality attributes into data stream elements enables applications to assess DQ independently. To take into account the complexity and unknown parameters of user-defined operations, the evaluation of DQ is dedicated to the particular application. For instance, a data mining classification algorithm could provide its accuracy (i.e., the correctly estimated elements compared to all classified elements) as DQ attribute. Figure 4 sketches the application of a user-defined operation such that a map matching algorithm that expects GPS coordinates as input and matches these to the corresponding road of a road network. Aside from the matching result, the user-defined function concurrently provides quality information according to its computations and configuration, respectively.

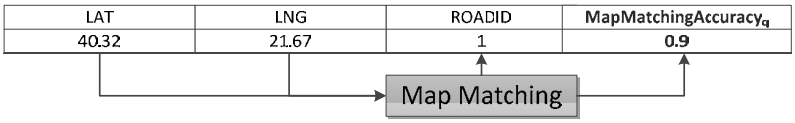

Figure 4: Illustration of user-defined operations

In our case study, the framework has been implemented around the GSN DSMS [8]. In the following, we will briefly present the implementation of the described components for the GSN system.

Proof of Concept

GSN comprises a flexible architecture: wrapper components are used to abstract from data sources, and virtual sensors are applied to combine data processing and filtering. Wrapper components are developed in Java code and create the stream elements that are propagated into the system. A virtual sensor is constituted of an XML configuration file, most importantly including the definition of the input SQL queries, an output SQL query of the sensor, the output structure of stream elements, as well as the linkage to a Java class providing the code executed when a stream element is processed. Depending on the configurations, the system arranges all virtual sensors in a dependency graph, which determines the data flow of stream elements. To take into account DQ processing, the semantic descriptions for the DQ assessment are loaded from the ontology at system startup. Based on this information, the virtual sensor configurations are rewritten recursively, including the queries and the output structure of each virtual sensor in the dependency graph. Aside from the offline DQ assessment that is rewriting the virtual sensor configurations, the evaluation of semantic rules and the interface for user-defined operations has been implemented into the Java class each virtual sensor inherits from. The extension allows adapting the output structure of stream elements and enables online DQ evaluation of rules by a math expression parser. Likewise, it provides an option to contribute DQ values to every stream element from user code.

EVALUATION

The evaluation of the DQ framework was carried out to test several aspects. First, we aim at a proof of concept in the domain of traffic applications to show the usefulness and adaptability of the system. For this, we use a hazard warning scenario detecting queue-ends on a highway. Second, we want to demonstrate, that the framework does not influence the real-time processing capabilities of the DSMS. Therefore, we examined the CPU usage, memory consumption and time complexity of the DQ framework. Finally, the effectiveness should be demonstrated: we varied the source quality of the data and tested, if the changes are reflected in the derived DQ values of the output streams. As generating the desired traffic situations is not possible in real life, we used the commercial traffic simulation software VISSIM for simulating traffic and the corresponding Car-2-X messages. We will first describe the evaluation scenario and setup, and subsequently present the evaluation results.

Case Study - Queue-end Detection

Many road fatalities and accidents are caused by drivers perceiving a queue-end too late. Queue-end detection is therefore an important traffic application which could reduce significantly the severe consequences caused by such accidents. However, queue-ends should be detected with high reliability as otherwise road users will ignore the information if there are too often false positives. In the queue-end scenario, messages provided by vehicles using Car-2-X communication are used to detect queue-ends. The messages are event messages, sent when a vehicle brakes very hard or turns on its warning flashers, and probe messages, which are periodically sent by the vehicles and include some probe data. The idea of the scenario is simple: roads (or *links*) are divided into equal-sized artificial sections. For each section the real-time messages are collected and the information of the messages, such as speed or acceleration, is aggregated. The aggregated information is fed into a data stream mining algorithm. The algorithm is a classifier which determines for each section if it contains a queue-end or not. If a queue-end was detected a hazard warning message is sent to road users in near vicinity of the queue-end's position.

As the DQ of the real-time messages may fluctuate, e.g., caused by measurement inaccuracies, low data volume, outdated information, missing values, misleading information, or inconsistencies, the reliability of the results of the data processing cannot be ensured at all times and in all aspects. It is desirable to disseminate only those messages to road users that correspond to a high level of DQ. Therefore, an overall confidence value based on multiple DQ values is calculated, which reflects the reliability of the queue-end estimation. Road users can then configure an individual filter such that they receive only queue-end messages above a certain confidence level.

The implementation of the queue-end detection scenario in the GSN system required the setup of two wrappers and six virtual sensors. We receive two independent data streams from the traffic simulation we use. One is the stream of real-time messages sent by the vehicles. The other one includes messages describing the real positions of the queue-ends in the traffic situation and provided by the traffic simulation (called ground truth in the further). The streams are received by the two corresponding wrappers and the data is converted to GSN stream elements. The positions in the real-time messages are distorted to simulate the error of GPS localization (we call this process *degrading*). Afterwards, the position is matched by another virtual sensor to a section and link on the road network (called *map matching*). The data of the messages is then aggregated over each section (we calculate average speed, average acceleration, number of messages for each message type) in another virtual sensor. In the same sensor, the data is integrated with the data of the ground truth messages. The integrated data stream can now be used as training data for the classification algorithms. We use a test-then-train approach, i.e., a data item is first used to test the quality of the data mining method, and then it is used as additional training data item. If the classification algorithm detects a queue-end in some section, a subsequent virtual sensor sends a message to the road users (in our simulation, the message is send to the simulation software to visualize the result). The data flow is depicted in Figure 5.

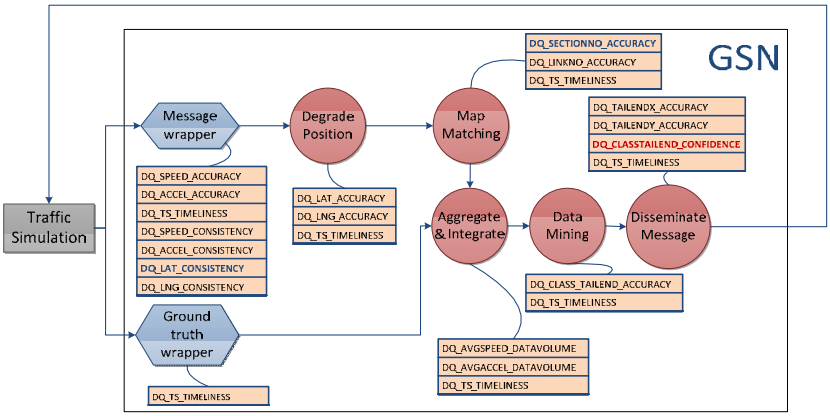

Figure 5: Data Flow of the Queue-end Detection Scenario

For each of the steps we identified DQ dimensions which influence the overall reliability. The DQ model had to be adapted and extended to model the required DQ dimensions and metrics. In Figure 5, the DQ dimensions for all components are shown. As an example, we describe the DQ dimensions of the message wrapper in more detail. It receives the messages from the traffic simulation. In real life, the inaccuracies introduced by the speedometer and the acceleration sensor have to be considered which we do in the simulation by introducing a statistical measurement error. The DQ measurement for these values can be based on general statistics or legal requirements (e.g., a speedometer must have an accuracy of 90% in the EU). Therefore, we use constant DQ metrics here. An important DQ dimension is timeliness. A timestamp (TS) is attached to each incoming message which will be used to calculate the age of the messages in the system. This is computed by the SQL DQ Metric. The data fields SPEED, ACCEL, LNG, and LAT will be assessed with a *Semantic DQ Metric* that calculates the DQ value using semantic rules for the dimension *Consistency*. The rules express the reasonableness and rationality of field values, and model domain-specific integrity constraints. The consistencies for the fields LNG and LAT are measured according to restrictions of the geospatial boundaries in the road network.
The total confidence value, which should indicate the reliability of the predicted hazard in the disseminated message, is handled as a common DQ dimension. It is also calculated by a *Semantic DQ Metric* for which a formula over various DQ values is defined in the ontology. The confidence formula is dependent on the application scenario and domain and can therefore be easily adjusted by changing the corresponding element in the ontology.

Results

After adapting the DQ model to include the DQ dimensions and relationships for the present data stream application, we ran traffic simulations on a road network and processed the data created by the simulation in the GSN system with the components described above. The simulated road network consists of one road of approximately 5 km length with two directions. The road is narrowed on one direction by a construction site which is surrounded by a reduced speed area. The volume of vehicles is set to 3000 vehicles per hour. 10% of the vehicles were equipped with Car-2-X communication technology, i.e., only these vehicles sent messages. Each simulation run had duration of 30 minutes. The experiments were executed on a machine with 8GB main memory, a 2.4GHz Intel Core I5 dual core processor, and Windows 7 64bit operating system. The first simulation run is considered as a reference run using the parameters and setup as described before.

Effect of DQ Changes in Derived DQ Information and Confidence Values

To assess how accurate the DQ information and the resulting confidence value reflect changes in the quality of the processed data, we simulated these changes in the data. First, we evaluated the effect of changes in the data volume of the incoming data on data quality factors by varying the sampling rate for the message wrapper. In Figure 6 the varied sampling rate (1.0 means 100% elements are forwarded) and its influence on the data volume quality dimension for speed in the Message Wrapper component is shown. The influence of this DQ factor is also reflected in differences in the varying confidence values. As this DQ factor has only a weight of 1/16 in the overall confidence value, the difference is not very significant.

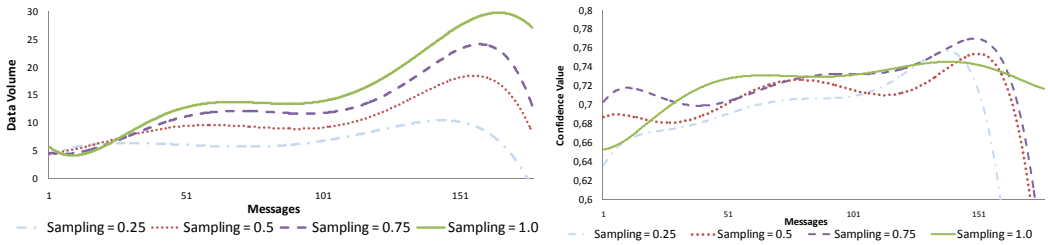

Figure 6: Data Volume Quality and Confidence Values for Varying Sampling Rates

In a second experiment we compared accuracies of positions and their influence on the overall DQ. We varied the error of positioning for the messages using a normal distribution for GPS localization accuracy with a deviation of 5 meters (which is the case for Floating Car Data, FCD) and cellular network localization using a deviation of 75 meters (which is the case for Floating Phone Data, FPD). This should have an impact on the map matching accuracy and of course also on the confidence values. Figure 7 shows the accuracy and confidence values for the two different cases with increasing message amount. It is easily observable, that the accuracy and confidence value of map matched messages using FCD positioning are both higher than the values for messages with FPD positioning. The difference for the map matching accuracy is higher, because again the confidence value is composed of several DQ values and the map matching accuracy influences it only with 2/16.

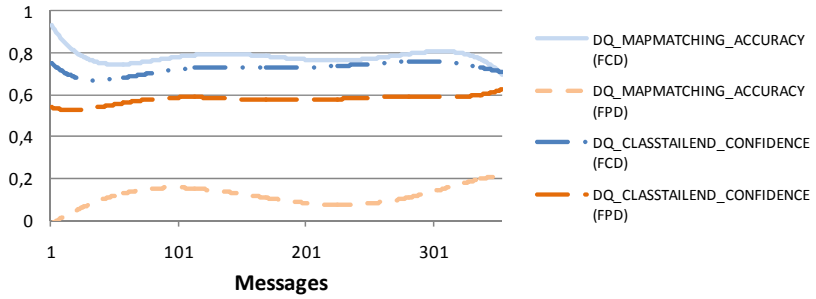

Figure 7: Influence of Positioning Accuracy on Map Matching Accuracy and Confidence Value

System Performance

One crucial requirement for the extension of a real-time data processing system is that the DQ assessment should not, or only slightly, influence the system performance. Therefore, we did several experiments comparing simulation runs with and without DQ assessment. Figure 8 shows the additional processing time for each virtual sensor of the scenario presented in the previous section. The figure reveals that the sensors, which include the evaluation of Semantic DQ Metrics (being Message Wrapper, Map Matching, Data Mining, and Disseminate Message), have a slightly higher additional processing time than the others, because they have to evaluate the Semantic Metrics. In contrast, the sensors Ground Truth Wrapper, Degrade Position, and Aggregate & Integrate only include SQL DQ Metrics and Constant DQ Metrics. The rewriting of the query is done as a preprocessing step; hence, the query execution is only marginally affected.

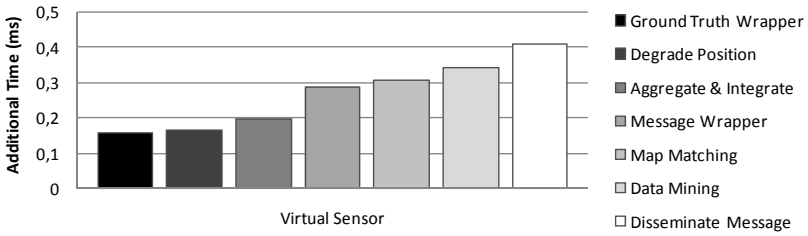

Figure 8: Additional Average Processing Times for each Virtual Sensor

We were also interested whether a DSMS using the DQ framework requires significantly more CPU power or main memory than a DSMS without DQ framework. Figure 9 shows, that the CPU usage is only higher during the initialization phase of the system. During initialization, the ontology is loaded and DQ information is processed by a reasoner. Also, the query rewriting is in this phase. After the initialization phase no substantial difference can be noticed. The same applies also for the memory consumption which is significantly higher during initialization due to loading the ontology. However, at the end of the simulation run the amount of memory is about the same in both situations. We assume that the slow decrease of the memory consumption is caused by the Java VM Garbage Collection.

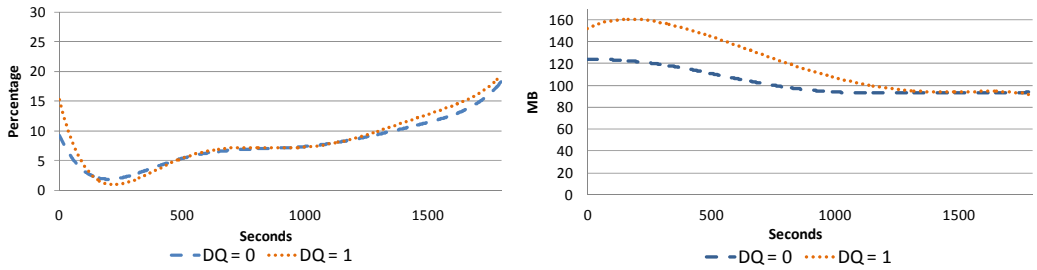

Figure 9: Comparison of CPU and Memory Usage between Runs with and without DQ Assessment Enabled

Case Study Traffic State Estimation

We demonstrated the usefulness and flexibility of our framework in a second case study which was a scenario for traffic state estimation. The traffic simulation was run on a real map (a part of the city of Düsseldorf). We adapted the virtual sensors and the data mining algorithms to estimate the traffic state on each road section. In line with the queue-end detection scenario, a confidence value for the information was calculated. The results were visualized in a web application, depicted in Figure 10, showing the most important values of the DQ assessment.

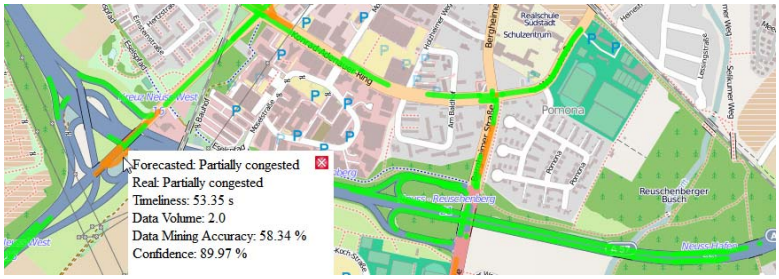

Figure 10: Visualization of DQ Information for the Traffic State Estimation Scenario

CONCLUSION AND OUTLOOK

We presented a flexible, holistic DQ framework for DSMS and showed its application for traffic information systems. Previous approaches for DQ management in data streams either focused on system-related aspects or were very tightly integrated into the DSMS which hampers the extension of DQ management with new DQ metrics or DQ dimensions. In contrast, our approach is based on a clear, semantic definition of the DQ related concepts in a semantic DQ ontology. In addition, the relationships between the DQ concepts and the components of the DSMS are defined in the ontology.

A key idea of our approach is the threefold notion of DQ in a DSMS, which enables a flexible and extensible DQ management. First, query-based DQ can be measured by integrating statements for computing DQ values into the query expressions in the DSMS. We avoid dependence on a specific DSMS by using a query rewriting approach: queries are rewritten to include DQ features before they are deployed in the DSMS. Second, content-based DQ refers to DQ dimensions that can be evaluated by semantic rules (or functions) using data that is contained in the data streams. Finally, application-based DQ offers the greatest flexibility by supporting DQ measurements by any kind of application-specific method. The evaluation of our framework in the context of traffic information systems based on Car-2-X communication has shown its flexibility and effectiveness. There is also no significant overhead in terms of CPU or memory usage due to our DQ framework.

Future work will address further applications in the traffic management scenario. We will use our framework to measure the DQ in the traffic applications while varying the system configuration. An interesting challenge is also to enable automatic adaptations of DSMS components (e.g., queries, windows, and sampling) if the DQ for certain dimensions is below a desired threshold.

Acknowledgements: This work has been supported by the German Federal Ministry of Education and Research (BMBF) under the grant 01BU0915 (Project Cooperative Cars eXtended, http://www.aktiv-online.org/english/aktiv-cocar.html) and by the Research Cluster on Ultra High-Speed Mobile Information and Communication UMIC (http://www.umic.rwth-aachen.de). We thank the PTV AG for kindly providing us with a VISSIM license. We also thank the reviewers for their valuable comments.

BIBLIOGRAPHY

[1] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom, "Models and Issues in Data Stream Systems," in *Proc. Symposium on Principles of Database Systems (PODS)*, 2002, pp. 1--16.

[2] Daniel J. Abadi et al., "Aurora: A New Model and Architecture for Data Stream Management," *The VLDB Journal*, vol. 12, no. 2, pp. 120--139, 2003.

[3] Sven Schmidt, "Quality-of-Service-Aware Data Stream Processing," Technische Universität Dresden, PhD Thesis 2006.

[4] Sven Schmidt, Henrike Berthold, and Wolfgang Lehner, "QStream: Deterministic Querying of Data Streams," in *Proc. of the 30th Intl. Conf. on Very Large Data Bases*, 2004, pp. 1365-1368.

[5] Daniel J. Abadi et al., "The Design of the Borealis Stream Processing Engine," in *Second Biennial Conf. on Innovative Data Systems Research*, 2005, pp. 277-289.

[6] Anja Klein and Wolfgang Lehner, "Representing Data Quality in Sensor Data Streaming Environments," *Journal of Data and Information Quality (JDIQ)*, vol. 1, no. 2, p. 10, 2009.

[7] Sebastian Röglinger and Christian Facchi, "How Can Car2X-Communication Improve Road Safety," University of Applied Sciences Ingolstadt, Working Paper 1612-6483, 2009.

[8] Karl Aberer, Manfred Hauswirth, and Ali Salehi, "A Middleware for Fast and Flexible Sensor Network Deployment," in *Proc. of the 32nd International Conference on Very Large Data Bases*, 2006, pp. 1199-1202.

[9] Jürgen Krämer and Bernhard Seeger, "PIPES: a Public Infrastructure for Processing and Exploring," in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 2004, pp. 925-926.

[10] Hoyoung Jeung, Sofiane Sarni, Ioannis Paparrizos, Saket Sathe, and Karl Aberer, "An End-to-End System for Cleaning Sensor Data: Model-Based Approaches," *Journal of Information Systems*, 2010.

[11] Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, Wei Hong, and Jennifer Widom, "Declarative Support for Sensor Data Cleaning," in *Lecture Notes in Computer Science, PERVASIVE 2006*, Kenneth Fishkin et al., Eds.: Springer Berlin / Heidelberg, 2006, pp. 83-100.

[12] Bhargav Kanagal and Amol Deshpande, "Efficient Query Evaluation over Temporally Correlated Probabilistic Streams," in *IEEE International Conference on Data Engineering*, 2009, pp. 1315-1318.

[13] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch, "Probabilistic Databases," *Synthesis Lectures on Data Management*, vol. 3, no. 2, pp. 1-180, 2011.

[14] Thanh T.L. Tran, Liping Peng, Boduo Li, Yanlei Diao, and Anna Liu, "PODS: A new Model and Processing Algorithms for Uncertain Data Streams," in *Proc. ACM SIGMOD*, 2010, pp. 159-170.

[15] Michael Stonebraker, Ugur Cetintemel, and Stan Zdonik, "The 8 Requirements of Real-time Stream Processing," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42--47, 2005.

[16] Arvind Arasu, Shivnath Babu, and Jennifer Widom, "The CQL Continuous Query Language: Semantic Foundations and Query Execution," *The VLDB Journal*, vol. 15, no. 2, pp. 121-142, 2006.

[17] Richard Y. Wang and Diane M. Strong, "Beyond Accuracy: What Data Quality Means to Data Consumers," *Journal of Management Information Systems*, vol. 12, no. 4, pp. 5--33, 1996.

[18] Diane M. Strong, Yang W. Lee, and Richard Y. Wang, "Data Quality in Context," *Communications of the ACM*, vol. 40, no. 5, pp. 103--110, 1997.

[19] Thomas C. Redman, *Data Quality for the Information Age*, 1st ed. USA: Artech House, 1997.

[20] Matthias Jarke, Manfred Jeusfeld, Christoph Quix, and Panos Vassiliadis, "Architecture and Quality for Data Warehouses: An Extended Repository Approach," *Information Systems*, vol. 24, no. 3, pp. 229-253, 1999.

[21] Norbert Baumgartner, Wolfgang Gottesheim, Stefan Mitsch, Werner Retschitzegger, and Wieland Schwinger, "Improving Situation Awareness in Traffic Management," in *Proc. Intl. Conf. on Very Large Data Bases*, 2010.