Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis Nr. 82

Creation of a Quality Concept for Situations within SitOPT

Rocco Schulz

Course of Study:	Softwaretechnik, Master of Science
Examiner:	Dr. rer. nat. habil. Holger Schwarz
Supervisor:	Dipl.-Inf. Mathias Mormul, Dr. rer. nat. Matthias Wieland
Commenced:	March 10, 2016
Completed:	September 9, 2016
CR-Classification:	K.6.4

Abstract

Situation aware applications rely on high level situations as context to adapt their behavior at runtime. Situations can be detected by aggregating and processing data and information from various sources. Just like its underlying data, situation information is uncertain and differs in quality.
This thesis presents generalistic methods that can be used to approximate the quality of detected situations. Its main contributions are a standalone fleet telematics simulation project as well as extensions to the SitOPT project which were created to implement the presented quality concept. The implementation has been tested with a simulated, non-trivial fleet telematics scenario to evaluate selected methods for their effectiveness. It has been shown that forming a weighted average from quality scores of multiple context information yields good results to quantify the quality of a detected situation. This simple assessment strategy can be improved by altering the quality of context scores in preprocessing steps.

Contents

List of Figures

List of Tables

List of listings

1. Introduction

Context-aware applications process gathered context information to adapt their behavior to changes in their environment. The majority of context-aware applications uses data from sensors, either directly or after processing and feature extraction [25]. The goal is to capture and represent data of real world objects. Typical domains where sensing is used include, but are not limited to, Smart Factories, fleet telematics [9], health-care and environmental monitoring [26]. Context providers such as sensors or management information systems provide data of varying quality. Many context-aware applications hence measure the Quality of Context (QoC) for better decision making [4, 22, 24]. Context-aware applications often deal with the same types of data but re-implement data aggregation and QoC quantification methods in an application specific way to get the information that they need and to filter information of low quality.

As opposed to context-aware applications, situation-aware applications do not directly process raw sensor data but require higher-level information inputs in the form of situations. This further decouples physical structures from the business logic and shifts the implementation of sensor data processing and situation recognition to an external service which submits detected situations to subscribed situation-aware applications. There might be multiple consumers interested in the same situation but with different quality requirements. The external situation recognition does not have the domain knowledge of the situation-aware applications and can't decide by itself which situations should be discarded. Instead, situations should be annotated with quality information (Quality of Situation - QoSit) so that situation-aware applications remain responsible to decide whether or not a situation should be used to adapt behavior.

This paper investigates different methods of assessing the Quality of Situations in order to allow situation-aware applications make better decisions. Quality assessment methods will be compared in the context of the SitOPT project.
SitOPT is a research project at University Stuttgart which aims at developing concepts and methods that will allow applications to adapt their behavior to fit the context in which they are running. The goal is to allow modeling situation-aware, adaptive workflows which behave differently when certain situations occur. Part of the project is a situation recognition platform.
This paper contributes a quality assessment framework to the project which annotates

detected situations with quality meta data. With these additional quality meta data in place, situation-aware applications can require a certain level of quality and base their decisions only on situations that meet their quality standards to reduce the risk of wrong decisions.

1.1. Problem Statement

Context information can be of varying quality for several reasons [27, 20]: Multiple error sources such as drift, environmental influences and aging reduce the reliability of sensor data. Information systems might contain inconsistencies, outdated or missing records. Situations which were detected using these unreliable data sources themselves cannot be used for business decisions without risking wrong decisions that potentially lead to financial losses or even safety issues.
An adoption of situation-aware applications is only justifiable in businesses when this risk has been reduced to a minimum. Ensuring that decisions are only based on situations of reasonable quality is one way to reduce this risk. Whether or not the quality of a detected situation is high enough to base a decision on it can only be decided by the situation-aware application if it has access to the Quality of Situation information.

Estimating the Quality of Context data is a task that has been studied for several years in the field of Wireless Sensor Networks, Ambient Assisted Living and other areas. However due to the different nature of situations and sensor readings the methods used for assessing sensor data quality cannot be applied to situations directly.
In the field of probabilistic complex event processing similar challenges have been faced. Methods which emerged from this field perform well on event streams, but often assume equal importance of atomic events [15, 16] and do not integrat well with other information sources such as management information systems.

Determining the Quality of Situations which were derived from multiple contexts remains a challenging issue [3]. In the case of situation-aware applications additional challenges arise from the logical decoupling of situation recognition and situation consumers. Domain specific knowledge can not be incorporated into the situation recognition directly, as it is often done in context-aware applications, but must be modeled in a simplified form as part of a situation template to account for the possibly diverse specifics of situation-aware applications.

Sensor values are usually only valid for a limited amount of time and situations might demand quick reactions. The quality assessment must thus not cause large delays. A key challenge is to find appropriate quality assessment methods that work well in scenarios

with data sources of differing quality while balancing demands for accuracy and performance. In order to evaluate under which circumstances they perform well, it is necessary to implement and compare these methods in a quality assessment framework.

1.2. Fleet Telematics Scenario

In the remainder of this thesis a notional fleet telematics scenario will be used to elaborate concepts and to present examples. Nonetheless, methods and concepts developed within this thesis should be applicable to various other domains and scenarios where situation awareness can be used in a similar manner to provide value.

Fleet telematics refers to the collection of telemetry data from a fleet of trucks or other vehicles [6]. Trucks are equipped with tracking devices which record data about position, speed, fuel level, engine temperature and various other properties. These data collections are sent to servers in fixed or variable intervals ranging from a few seconds to several minutes for further processing. Fleet telematics is used to optimize driving behavior of truck drivers, to reduce deterioration of mechanical components, to schedule maintenance in advance, to increase utilization, or to verify that drivers did not exceed legal working hours [6, 9]. In the notional scenario below, telemetry data is combined with traffic data to detect delayed deliveries.

An automotive component supplier manufactures car seats as ordered by the car manufacturer. There are different seat models and customers may choose between a range of configuration options, such as different colors, shapes and materials. The seat manufacturer has contracts for just in time delivery with his suppliers so that parts usually arrive a few hours before they are needed in the manufacturing process. Occasionally parts cannot be delivered in time as trucks might get stuck in traffic. Delays can cause drivers to exceed the legal working hours, so that they are obliged to take a break before they may continue driving, further delaying the delivery.
In these cases the manufacturer must reorganize its production process so that the production does not come to a halt. Seats which cannot be manufactured must be postponed in favor of others which can be manufactured with stocked components. In order to detect these situations, the manufacturer uses a fleet telematics provider's service that processes telemetry data of all trucks to detect when trucks are going to be delayed. Each truck locally collects data about speed, location and driving direction and pushes the collected values to the service provider's server every 15 seconds. The service provider uses situation templates to detect when a truck is stuck in a traffic jam and if subsequently the legal driving hours will be exceeded. To account for the uncertainty of detected situations, the provider annotates situations with Quality of Situation (QoSit) information and pushes them to the subscribed systems.

1.3. Methodology and Structure

First, methods to quantify qualities of situations in general will be developed. An evaluation of concrete method implementations will then be performed using a simulation of the scenario described in section 1.2.
For the simulation, a quality assessment component will be designed and implemented in the SitOPT project so that each developed method can be used in the component. Situation templates will be represented as Node-Red flows to recognize situations. These templates will be extended with meta data as needed to parametrize quality assessment methods. The underlying sensor layer will be emulated with fabricated data which has been altered to mirror the characteristics and unreliability of real GPS sensor data. Simulation results will be captured using different quality assessment methods. Those results will then be analyzed to evaluate the effectiveness of the designed quality concept.
Chapter 2 and chapter 3 review related work and provide the theoretical background on quality in information systems. chapter 4 presents approaches for determining situation quality in information streams. In chapter 5, system requirements and design decisions are elaborated that lead to the prototypical implementation which is presented in chapter 6. The test setup and evaluation of results is provided in chapter 7. Chapter 8 concludes this thesis and suggests changes to the SitOPT project which are based on the findings of this thesis.

2. Related work

In the field of Wireless Sensor Networks (WSN), determining sensor data quality is a common issue. Multiple methods and heuristics have been studied to estimate Quality of Data and Quality of Information in WSN setups. These usually only analyze sensor readings from a single source, or from multiple sensors which are attached to the same object. A common technique to increase data quality is sensor fusion, where readings of multiple sensors which measure the same property are combined to reduce noise [10]. Others have exploited the fact, that sensors suffer from several environmental influence factors and used cross validation to detect if the sensor can be trusted [17, 2]. Machine learning algorithms have been used successfully in some WSN research projects [31, 20] but those approaches are not feasible for situation quality assessment due to the lack of training data. [24] introduce a QoI metric which consists of sensor node sampling rate, network delay and loss rate. They constructed a framework which allows applications to demand information in a certain quality. However this setup is specific to WSNs and the metric only takes communication aspects into account.

Marie et al. describe a framework for filtering context information based on QoC contracts between context providers and context consumers with different QoC criteria, such as freshness, precision and correctness [23]. The underlying meta model with QoC criteria is described in [22]. They found none of the studied existing meta models to satisfy requirements for expressiveness, genericity and computability and thus developed the QoCIM meta model which aims at satisfying those requirements. Their meta model is mainly concerned with the quality dimensions which can be used to quantify overall context quality but does not compare concrete model instances for applicability in situation-aware applications.

Concrete implementations which are not strictly bound to a specific domain, as opposed to many context-aware applications, can be found in the domains of uncertain stream processing and probabilistic complex event processing.
In [16], the authors implemented an extension to the data stream management system (DSMS) *Odysseus*[1] to allow considering quality in all processing steps. They determine the quality of incoming sensor data using the Expectation Maximization algorithm over

[1] http://odysseus.informatik.uni-oldenburg.de

a sliding window. They adapted the operators of the DSMS to respect the meta data for quality aware processing.
They performed experiments with generated sensor data as well as with real sensor data to show that a probabilistic approach is capable of reducing the number of false positives. However they also outline that thresholds must be chosen on a per-application basis and that the quality awareness introduces an additional latency in the stream processing.

Nazário et al. have proposed a context processing middleware with modules for QoC quantification and evaluation. They split QoC into the parameters coverage, up-to-dateness, precision, completeness and significance and calculate the QoC as the average of the parameter values. However, they only determine QoC for each sensor individually[2] but not for more complex information which has been gained by combining multiple context providers [29].

Kawashima, Kitagawa, and Li present an extension of the SASE+ query language so that confidence thresholds can be specified as filters in queries. Their extension supports probabilistic input streams and outputs confidence annotations in output streams [15]. The confidence of detected complex events is calculated as the product of the probabilities of input events. They hence assume equal importance of all events making up the complex event. Chuanfei et al. use a similar approach to determine the probabilities of complex events [5].

Löffler, Mutschler, and Philippsen demonstrate an event based system middleware that processes probabilistic event streams to detect event combinations of interest. They calculate the probability of detected events using the probabilities of the underlying events. They also provide domain experts with the possibility to add additional logic that weights input events differently for a more sophisticated probability calculation [21].

[2]they call it QoC, however this is QoD according to the definitions used in this thesis

3. Theoretical Foundations

3.1. Situation Awareness

In event processing, a situation is often defined as an event which demands a reaction [7]. In this thesis a situation refers to an event or state which might be of interest for a situation-aware application. It is likely but not necessary, that situation-aware applications need to react to the occurrence of a situation.
Situation-aware applications are context-sensitive, but they do not rely on low level context information but on higher level aggregated information in the form of situations [35].

In the scenario described in section 1.2, the manufacturer's application which reorganizes production processes is situation-aware. It receives situation information from an external service and reorganizes the production schedule if the situation demands it.

In SitOPT situations are modeled in situation templates which are represented in XML. A situation template either evaluates to true or to false. It contains one to many condition nodes which are joined using a single operation node [11].

3.2. Quality Terms

Quality is an attribute that can be assigned to physical entities such as products, or intangible entities like services, data and information. While often used interchangeably, distinctions between *data* and *information* have been made in scientific literature [19]. A widely used approach considers data, information and knowledge as a hierarchical structure [19]. This approach is also adopted by Ackoff, who defines these terms as follows:

Data "Data is raw. It simply exists and has no significance beyond its existence (in and of itself). It can exist in any form, usable or not. It does not have meaning of itself. In computer parlance, a spreadsheet generally starts out by holding data." [1]

Information “Information is data that has been given meaning by way of relational connection. This "meaning" can be useful, but does not have to be. In computer parlance, a relational database makes information from the data stored within it." [1]

Throughout this thesis, Ackoff’s definitions will be used to provide the necessary foundation for the related terms *Quality of Data*, *Quality of Information* and *Quality of Context*.

3.2.1. Quality of Data

According to Wang et al. data consists of two related components: a data model and data values. Data models are usually abstractions of the real world. They argue that data correctness cannot be determined without a link to the modeled concept or human knowledge. Data is considered to be correct when it agrees with the real-world counterparts or with a standard that is assumed to be correct [33].

It is not possible to determine if a number is correct or wrong without knowing the intended use or model. Bit-sequences that cannot be parsed (e.g. due to transmission errors) are of low quality as they do not fit their model. A datum that contradicts its counterpart, e.g. a temperature representation that differs from the actual temperature, is also of low quality.

3.2.2. Quality of Information

Information includes data, meaning and context [19]. A common approach to define and measure Quality of Information is by breaking it down into multiple dimensions. An extensive list of quality dimensions, grouped into four categories, is presented in [12]:

intrinsic quality accuracy, objectivity, believability, reputation
accessibility quality access, security
contextual quality relevancy, value-added, timeliness, completeness and amount of data
representational quality interpretability, ease of understanding, concise representation, consistent representation

In the running scenario various service providers maintain databases of traffic reports and alerts. The quality of this information may be assessed using the aforementioned quality dimensions. While some dimensions, like timeliness, can be measured easily, other dimensions, such as value-added and ease of understanding, require human interpretation and cannot be measured objectively.

3.2.3. Quality of Context

Context information is information that has been provided to, or was obtained by a context-aware application. Buchholz and Schiffers define Quality of Context as follows:

> Quality of Context is any information that describes the Quality of Information that is used as context information. Thus, QoC refers to information and not to the process nor the hardware component that possibly provide the information. [4]

In other words, QoC is meta data which describes the quality of context information. In the exemplary scenario, the service provider's application which performs the situation recognition is context-aware. Its context is made up of sensor data, aggregated information and traffic information.

As mentioned in the previous section, not all QoI dimensions can be measured in an automated approach. In context-aware applications only a subset of possible quality dimensions is used to describe or measure QoC. Buchholz and Schiffers ranked common QoC dimensions by importance. The ranking is based on experience and previous literature. They found probability of correctness, trust-worthiness, resolution and timeliness to be of highest importance. Probability of correctness is an estimation by the original source of data that the values might be wrong due to internal problems. Trust-worthiness is an estimation obtained by observing the behavior of the original source [4]. The data source might state a probability of correctness of 80%, but a data processor might observe a rate of 50%. Resolution refers to the granularity of information. Timeliness describes the age of context information.

4. Assessing Quality of Situations

The quality of a detected situation depends on the quality of the underlying context information. As situations are usually detected by analyzing multiple contexts, it is necessary to find appropriate methods to aggregate the quality attributes of context objects into a higher level QoSit value.
There are two basic approaches which can be taken to assess the quality of a situation. The filtering approach is to only use sensor data which is known to be of good quality in the situation detection. Another approach is to quantify the quality of the situation so that consuming applications can decide whether the quality is high enough to be used for a decision.
The filtering approach guarantees a minimum quality of the detected situation, is easy to implement and performs well on large data streams. The quantification approach allows more fine grained modeling in situation-aware applications and allows a more sophisticated view on quality. Concrete methods for both approaches are elaborated in the following subsections.

In order to embody quality assessments the situation template schema as described in [34] needs to be extended to allow more expressive quality requirements in the modeling phase.

4.1. Selecting Relevant Quality Dimensions

Various quality dimensions have been introduced in section 3.2, however not all of these dimensions are relevant for the assessment of a situation's quality. Some dimensions cannot be used for practical reasons, e.g. because they are not measurable or require human interaction, others are not important from a conceptual perspective.

Not all context information that is available for situation recognition carry meta information about the same quality dimensions. Positions reported by a GPS device might be annotated with timestamps and accuracy in meters, but traffic incident reports might only be annotated with a timestamp. Obviously quality dimensions can only be incorporated into the overall quality of a situation if they are present in the underlying context. This implies that annotating detected situation instances with measured quality

dimensions is impractical as no guarantees can be made to situation-aware applications that such fine grained annotations will be present.

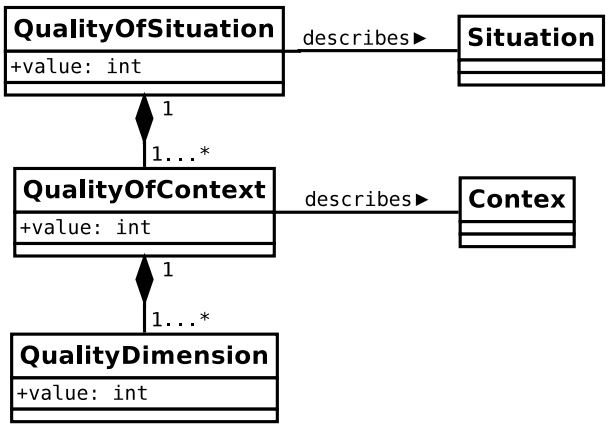

Figure 4.1.: Relationship between QoSit, QoC and quality dimensions

Instead, the situation recognition service should annotate situation instances with a total quality score, referred to as the Quality of Situation. The Quality of Situation should be composed of the QoC scores so that quality dimensions are only included indirectly through the QoC values. This is schematically depicted in Figure 4.1. This way domain specific dimension selection is separated from the situation recognition component. However this comes at the cost of losing differentiated meta information which might be required by some situation-aware applications.

Subsequently it will be assumed that context information have been preprocessed to carry a QoC score.

4.2. Input Stream Filtering

The situation recognition service consumes context information and produces situation events. Context information can be seen as a data stream which can be filtered to exclude information of low quality. Filtering all incoming measurements which are below a quality of 30% ensures a situation quality of at least 30%.
The filtering approach is easy to implement, yet does not properly quantify the situation quality. It can only guarantee a minimum quality, whether or not this is sufficient depends on the requirements of the situation-aware applications which have subscribed to the situation events.

In scenarios with high velocity/big data input streams, assessing total situation quality might not be feasible. Filtering the input stream before performing the situation recognition can reduce the system load at the cost of lower precision.
On the other hand, the filtering approach might be ill-suited for scenarios where messages arrive only sparingly or irregularly. In the fleet tracking scenario, telematics boxes push messages with their current position in intervals of 15 seconds. If the filter criterion for GPS position data is above the median quality, then the interval is effectively lengthened to 30 seconds or more on average. Entirely dismissing inaccurate messages might not be desired in such scenarios.

4.3. Weighted Averages

The simplest form to quantify situation quality is by averaging the quality scores of its underlying contexts. When equal weights are assigned to all context information, then the situation quality Q equals the arithmetic mean of all context quality values q:

$$Q = \frac{1}{n} \sum_{i=1}^{n} q_i$$

Using the arithmetic mean might not be well suited for all situations as not all conditions which make up a situation are equally important. Situation templates could be extended to carry information about weight distributions (w), so that the total quality is then calculated using the weighted average of the underlying quality scores:

$$Q = \frac{\sum_{i=1}^{n} q_i w_i}{\sum_{i=1}^{n} w_i}$$

Assume that one of the trucks in the fleet telematics scenario transports sensitive goods which must not be exposed to temperatures above 30°C. To account for various sensor faults the trailer is equipped with two temperature sensors (front and back). When both of them report a temperature of more than 30°C, then the situation should be recognized (see Figure 4.2).

The situation would evaluate to `true` when a message as shown in Listing 1 enters the situation recognition as both sensors report a value above 30°C. The resulting quality score for the situation would be $\frac{0.9+0.2}{2} = 0.55$, assuming that both sensors are weighted equally.

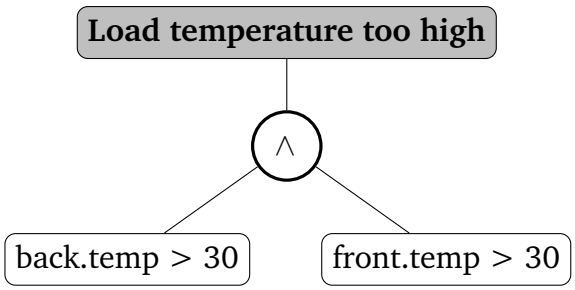

Figure 4.2.: Situation graph - load temperature above 30

```
{
  "timeStamp": 1463599886,
  "back": {"temp": 50, "quality": 0.2},
  "front": {"temp": 40, "quality": 0.9},
}
```

Listing 1: Example message with temperature values

4.3.1. Handling Disjunctions

The example above used a conjunction to connect multiple conditions, however disjunctions can be treated in multiple ways. When a situation template looks like $a \vee b \vee c$, and conditions a and b are `true`, then either all condition qualities can be taken into account or only a subset.
Taking the example above, assume that the situation only requires one of the sensors to report a temperature above 30°C. The situation would have a score of 0.55 if both sensors were used for the QoSit calculation, but a score of 0.2 if only the back sensor was used or a score of 0.9 if only the front sensor was used.
This simple example shows the limitation of the approach. Intuitively we'd want the quality of the situation to be higher if both sensors report a temperature above 30°C than when only one sensor reports a temperature above 30°C. One sensor is sufficient to satisfy the situation condition, but a second sensor can provide further evidence and should hence increase the quality instead of lowering it. The occurrence of a single condition already fulfills the expression and thus the total quality should be determined by taking the maximum quality of all branches that evaluate to true in conjunctions rather than the average. In this case the quality would be $max(q_{\text{back}}, q_{\text{front}}) = 0.9$ instead of $avg(q_{\text{back}}, q_{\text{front}}) = 0.55$.

4.3.2. Handling Missing Quality of Context Meta Data

It cannot be guaranteed that all context information that make up a situation are annotated with a QoC score. One way to deal with such values is to exclude them from the calculation. Another way to treat this case is to assign a default quality value. Such a default could either be assigned to the sensor or data source itself in the context provider so that QoC meta data is added transparently for the situation recognition. Alternatively a default value could be specified as a fallback in the situation template.

4.4. Pessimistic Approach

Quality of Situation can be determined with a pessimistic approach to reflect the idea that a detected situation is only as certain as the most uncertain underlying context information. The overall situation quality always equals the lowest quality score among all of its conditions.
This approach is especially interesting when situations are entirely composed of conjunctions but is inappropriate when disjunctions are used for the aforementioned reasons in subsection 4.3.1.

4.5. Confidence Intervals on Ordinal Scales

When a situation template defines a threshold for a measurement which is reported on an ordinal scale, then the measurement's quality is not relevant if it clearly exceeds the threshold. However if the measurement's value is close to the threshold, then the measurement's quality impacts the quality of the situation because of higher uncertainty. To model this observation in the calculation of the situation's quality, one can weight context qualities with a factor which depends on the value's distance from the threshold. This can be performed independently from the performed assessment method as a preprocessing step on each condition of the situation.

Consider a situation which states that a truck moves slower than 5 m/s when it is in a traffic jam. The telematics box reports a speed of 1 m/s with a quality of 70%. This speed is clearly lower, so even if the value is not correct it is very likely that the condition is actually met in reality. In a preprocessing step the QoC could hence be multiplied with a factor of 1.2, resulting in a QoC of 84%. On the other hand, if a speed of 4.9 m/s is reported a penalty factor of 0.9 could be used, resulting in a quality of 63%.

Obviously this approach is not suitable when matching strings[1] or using the $=$ operator.

4.6. Confidence Reinforcement in Event Driven Scenarios

Two types of information flows can be distinguished. Repeated, (periodic) polling for sensor data and event driven push approaches. In an event driven scenario situations are recognized in near real time with every update event from the underlying sensor layer.

If the same situation is recognized repeatedly, then the confidence that the situation has actually occurred should be raised for consequently detected situations as it is unlikely that multiple false positives occur in a short time frame.
An exemplary calculation with this reinforcement startegy is provided below. In the example below the quality of the previous situation is multiplied with a weight of 0.1 and added to the current situation to increase the situation's quality score:

$QoSit_{\text{reinforced}}(t) = QoSit(t) + weight * QoSit(t-1)$
$QoSit_{\text{reinforced}}(1) = 0.5$
$QoSit_{\text{reinforced}}(2) = 0.6 + 0.1 * QoSit(1) = 0.65$
$QoSit_{\text{reinforced}}(3) = 0.5 + 0.1 * QoSit(2) = 0.565$

This can be implemented with a time-bound sliding window which retains detected situations for a certain amount of time. This is a post-processing step which is independent from the actual calculation of the QoSit but it could also be applied on the condition nodes that make up a situation.

It is important to note, that this confidence reinforcement approach requires inputs to be time ordered. Out of order events could potentially lead to undesired behavior. Using this strategy also increases the systems memory consumption due to its need to cache old events.

[1] except for specific cases, where different algorithms for string comparisons might be applicable to calculate similarities between strings

4.7. Hybrid Approaches

The aforementioned methods are not necessarily exclusive but can be complementary. Hybrid approaches can be implemented to create systems that can suit requirements of various scenarios.

An exemplary three step process might filter the input stream to get rid of low quality events and to reduce the system load. A preprocessor, such as the confidence interval preprocessor could then be applied to raise or lower QoC with respect to the knowledge embodied in the situation template. The actual quality calculation could then be performed with the weighted averages strategy. The confidence reinforcement strategy could be applied as a final post processing step.
Which preprocessing and postprocessing steps are to be performed should be modeled individually in the situation templates to make allowance for domain specific requirements.

5. Designing the Quality Assessment Framework

The quality assessment framework or component should be integrated into the SitOPT project but should be generalistic enough so that the presented approaches can be easily integrated into other projects.
The design of the framework is hence bound to certain requirements and constraints which can be derived from the SitOPT project and typical usage scenarios, such as the fleet telematics scenario.

5.1. Requirements Analysis

5.1.1. Constraints

Limitations of SitOPT

The quality assessment should be integrated into the SitOPT project. The following limitations of SitOPT are present which might require adaptations in the SitOPT implementation and should be considered in the design of the quality assessment component.

Only works for single objects The SitOPT implementation does currently not allow putting multiple objects into relation with each other. An underlying assumption of SitOPT is, that only situations ought to be detected which originate from a single object. The introduced fleet tracking scenario however requires at least two different data sources: the truck and the traffic information system. Context information from both sources cannot be correlated via a unique ID but by location - an attribute with a constantly changing value that can't be modeled using SitOPT templates [13].

Uses polling SitOPT manages all context information sources in a component that exposes this information via an HTTP API. This component is called *Resource Management Platform (RMP)*. Polling is performed in intervals to obtain context information from the RMP [13]. A short polling interval unnecessarily creates a

high load on the RMP and a long interval potentially causes outdated information to enter the situation recognition. For the tested scenario, a push approach from the RMP to the situation recognition is more appropriate.

Time frames or sliding windows are not supported The RMP only returns the latest context information instance and does currently not allow getting historical data or aggregated data over specified time frames [13].

Uses Node-Red flows for situation recognition In order to execute situation templates, Node-Red flows are generated from the template files where each condition in the template is represented as a single node in the Node-Red flow [13]. Node-Red internally uses Node.js, which is asynchronous and single threaded.

Software Framework

The current SitOPT implementation is implemented in JavaScript using the asynchronous JavaScript runtime *Node.js*. The quality assessment component will be implemented using the same framework in order to maintain an easily manageable technology stack. One limitation of Node.js is its single threaded nature [30] which must be taken into account during the design to ensure adequate scalability. Blocking code must not be introduced into the Node-Red situation recognition flows as that would significantly reduce throughput.

5.1.2. Nonfunctional Requirements

Scalability Just like SitOPT, the quality assessment framework should be usable in various scenarios and must also support large deployments with big data and high velocity data streams. The components must be designed in a way which allows distribution over multiple machines to ensure horizontal scalability.

Extensibility The design of the quality assessment component should be easily extensible so that new quality assessment approaches can be registered.

Resilience The situation recognition and situation-aware applications should not be disrupted if the quality assessment component becomes unavailable. As a fallback detected situations should be delivered to situation-aware applications without a quality annotation.

Independence from Recognition System In the current SitOPT implementation, Node-Red is used for detecting situations. However a goal of SitOPT is to support other data stream processing or Complex Event Processing systems like Odysseus and Esper [34]. The quality assessment should hence be decoupled from the processing system to avoid the need of re-implementation for each system.

5.1.3. Functional Requirements

Push and poll support The underlying sensor layer should not be constrained by the quality component. The quality component should work with different sensor types and information flows (poll/push). I.e. captured sensor data should be retrievable using HTTP requests to a REST API as it was planned for the RMP (poll), but sensor data should also be fed into the situation recognition subsystem immediately after they have been processed in the Context Manager (push).

Support for multiple assessment methods Quality assessment methods should be interchangeable and adaptable. The quality assessment method should be specifiable in the situation template. The quality assessment component must use the method or list of methods as they are listed in the situation template.

Quality scores The quality assessment scores must be represented as percent values ranging from 0 to 100. Additional quality attributes, such as timeliness should be added where applicable.

5.2. Design

5.2.1. High Level Architecture

Figure 5.1 shows the SitOPT architecture as it was deduced from the current component implementations. Note that the architecture was deduced from the current prototype implementation of SitOPT as it can be found on Github[1] and hence differs from the architecture that has originally been proposed in previous publications in the SitOPT context.

The RMP manages the logical representation of the infrastructure in its internal resource store. Each sensor is expected to have a URL and to be accessible via the HTTP. The RMP regularly polls for sensor values and stores the most recent value for each sensor in its resource store. The stored sensor values are accessible for the situation recognition component (Node-Red) via an HTTP interface [13]. In its current form situation-aware applications need to query for occured situations by using the HTTP interface of the situation manager. This architecture will be slightly altered to consider quality attributes in all processing steps and to satisfy the requirements listed in section 5.1. The altered architecture is depicted in Figure 5.2.

The physical layer does not need to be changed for the altered architecture. The physical layer contains multiple objects which may have one to many sensors. Object is used as a

[1] https://github.com/mormulms/SitOPT

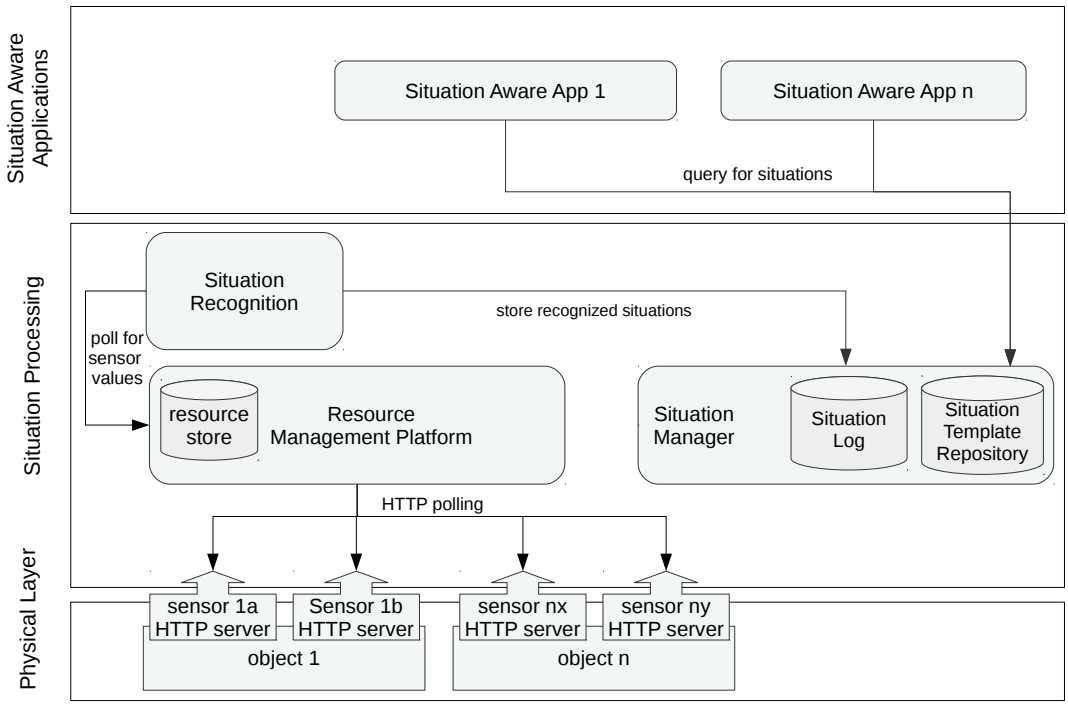

Figure 5.1.: High level architecture - SitOPT components (deduced from the current implementation)

general term for all sorts of observable units such as machines, vehicles or environmental factors.

Sensor data flows from the observed objects to the situation processing layer, where adapters read the sensor-specific messages. These adapters extract and transform information from the messages so that those can be stored as context by the context manager. Context information should be stored together with QoC meta data in the context store. If no quality attributes are provided by the sensors, then the context manager has to calculate quality meta data based on historical data. Possibilities to do so have been studied in various papers and will not be further elaborated here. In the running scenario of this thesis it is assumed that context information which enter the situation recognition system are already annotated with quality meta data. However in different scenarios the context manager component will need to address the lack of such meta data by calculating QoC before forwarding context to the situation recognition component. The context manager will be changed so that it stores historical sensor data. In its current form, the RMP only stores the most recent value for each sensor. Without access to historical data, there is no way to perform outlier detection, or to determine information quality attributes like trust-worthiness or variability.
Due to the use of adapters, multiple communication protocols and encodings can be

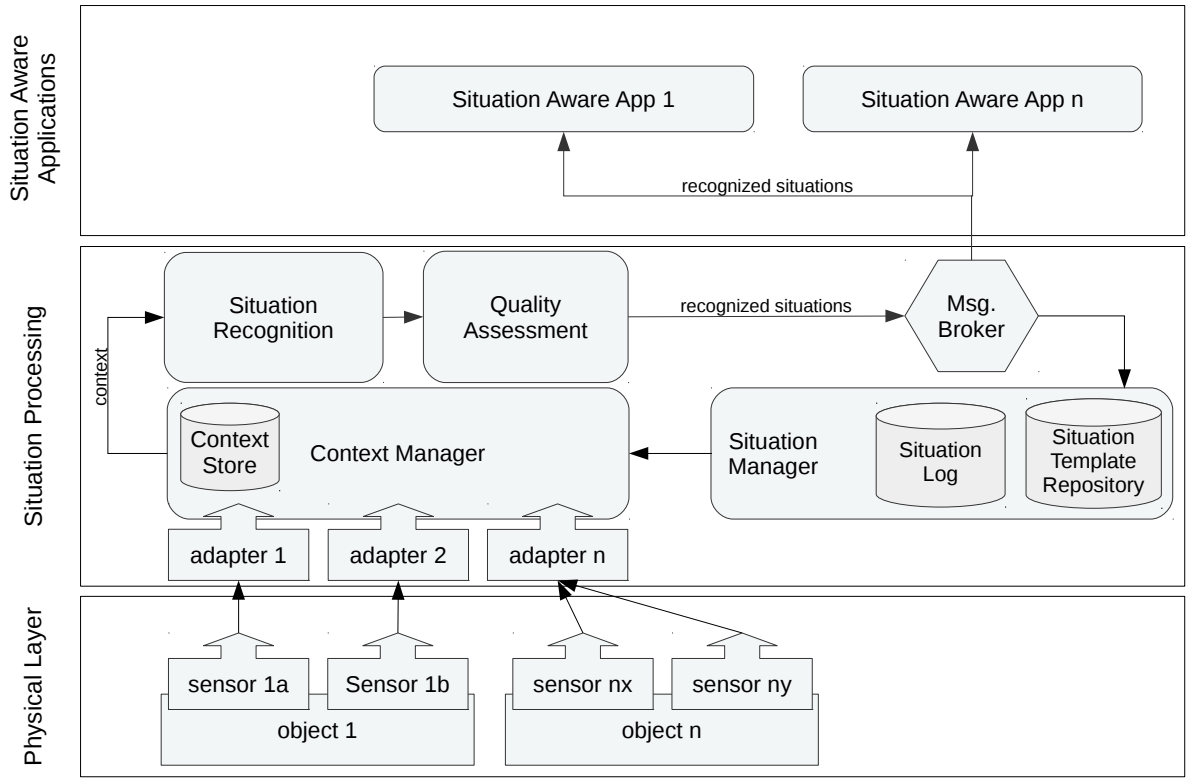

Figure 5.2.: High level architecture - component relationships

supported. In the running scenario each truck is an object and the telemetry boxes submit messages with all collected sensor values to an adapter via the HTTP. Other protocols such as MQTT or XMPP-IoT could be supported by registering appropriate adapters in the context manager.

Stored context information will be used in the situation recognition component, which is basically a wrapper for stream processing software such as Node-Red or Esper. In SitOPT situation templates are modeled in a system-agnostic way and are translated to system-specific executable expressions. Currently, only Node-Red is supported, but support for other systems might be added in the future.

An additional component is required to perform the quality assessment of the detected situations. This component can either be designed as a subsystem of the situation recognition or as a separate service which interacts with the situation recognition. The former requires that each supported stream processing system (Node-Red, Esper,...) has a separate specific subsystem implementation. A separate service though can be used by all stream processing systems as long as the message format is supported. The

situation quality assessment is thus introduced in a separate loosely coupled component which accepts messages from the situation recognition component.

The Context Manager component is responsible for persisting detected situations in the situation log. It also manages situation templates in a situation template repository. SitOPT situation templates allow using other situations as input, i.e. detected situations can be treated like sensor values. The quality assessment requires all sources to carry quality meta data. It is hence necessary that quality annotated situations are accessible by the situation recognition. This can be accomplished by exposing situations from the situation log to the Situation Recognition or by storing them as context information in the context manager. Using the latter approach allows re-using the context provisioning mechanisms of the context manager for situations as well, which is why this approach has been chosen in the architecture. Detected situations will be accessible via the Situation Manager in a request-reply fashion nonetheless.

Obviously this means that multiple components need to access detected situations. The context manager must receive situations to store them as context, the Situation Manager must receive situations to persist them in a situation log and situation-aware applications must receive notifications when situations have occurred. Components need to receive situations selectively, e.g. a third party application might only be interested in a specific situation type which occurs for a specific object with a high quality while the Situation Manager needs to receive all occurred situations.
A message broker with routing capabilities will be introduced into the system to provide these capabilities. The broker will manage subscriptions of components in the situation processing layer and of situation-aware applications. The quality assessment component will act as a producer which submits quality annotated situations to the broker. The situation manager and situation-aware applications act as consumers. The broker then routes messages that it receives from the producer to the consumers. The data flow is further elaborated in subsection 5.2.2.

5.2.2. Data Flow

The components described in the architecture are loosely coupled and message driven. Figure 5.3 provides a more detailed view on the interaction mechanisms used for integrating various components.

The situation recognition component acts as a producer which submits raw situation objects to a worker queue. Multiple worker instances of the quality assessment component consume those messages to annotate them with quality meta data. The worker queue is necessary to allow spawning multiple quality assessment instances. These instances will be implemented with Node.js and are single threaded. Spawning multiple

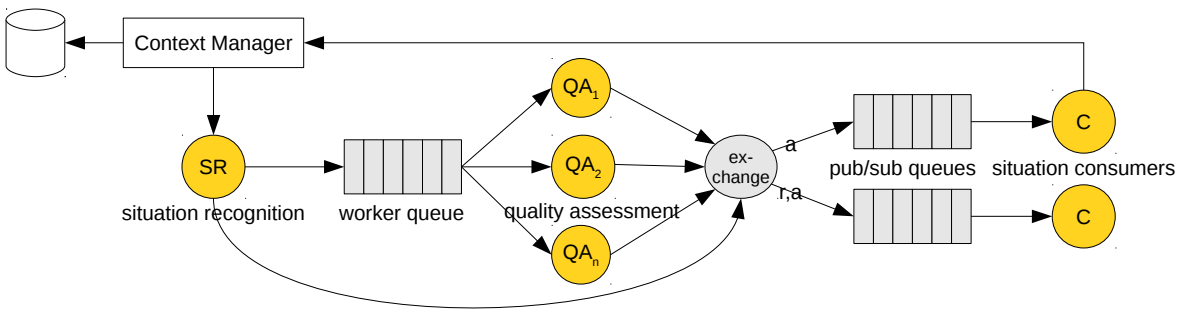

Figure 5.3.: Data flow from the situation recognition to quality assessment component via AMQP broker

instances on one machine ensures vertical scalability, spawning instances on multiple machines ensures horizontal scalability. The worker queue also allows to have multiple instances of the situation recognition in big data scenarios. An implication of this scalable architecture is that quality assessment components should be stateless.

Situations which have been annotated with quality meta data are then submitted to the message broker. The broker maintains an exchange that routes situations to the queues of subscribed consumer applications. Each consumer has its own queue which is created when a consumer registers itself with the message broker. Consumers tell the message broker which situation types they want to receive. The routing mechanism of the message broker ensures that each queue only receives situations that are relevant to the consumer.
One of those consumers is the Situation Manager component which submits annotated situations to the context manager, so that situations can be provided as context to the situation recognition component.

The situation recognition component not only submits situation messages to the worker queue but also submits raw situations to the message broker's exchange. These raw situations can be consumed by time critical consumers immediately to perform work preemptively. They also serve as a fallback to satisfy the resilience requirement in case that the quality assessment becomes unavailable.

5.2.3. Message Formats and Protocols

Multiple message brokers, messaging protocols and formats are available that can be used to realize the messaging between the components. The selection will be based on the already used technologies and aforementioned system requirements.

The message format will be JSON which is a natural choice for components that are based on Node.js and JavaScript. JSON is lightweight and supported by all major programming languages. A schema can be defined for message types using json-schema[2]. The schema has to be adhered by the communicating components to ensure interoperability. The schema can also be used to validate messages and automate testing.

To keep the technological complexity low, a message broker will be chosen that supports all messaging requirements within the architecture. The broker in question must hence support publish-subscribe messaging and push-pull messaging for the worker queues. Topic based routing should be supported in publish-subscribe messaging, so that consumers can selectively subscribe to certain situation types and specific objects. It should also be possible to horizontally scale the broker in big data scenarios, otherwise the broker will introduce a bottleneck into the architecture. Message delivery should be guaranteed for messages that are being sent from the situation recognition to the quality assessment. Appropriate counter-measures should be supported to prevent out of memory errors when message producers outpace message consumers. Client libraries for JavaScript should be available.

The popular open source message brokers *Kafka*[3], *RabbitMQ*[4] and *ZeroMQ*[5] have been evaluated based on those criteria. A brief overview is provided in Table 5.1.
Kafka is a “distributed, partitioned, replicated commit log service" [8], which provides the functionality of a messaging system but differs in its design.
“RabbitMQ is a messaging broker - an intermediary for messaging. It gives your applications a common platform to send and receive messages, and your messages a safe place to live until received.“ [14]
ZeroMQ is a lightweight messaging queue with support for a large number of languages and most messaging patterns [32].

ZeroMQ has not been chosen as it does not support persistent queues. RabbitMQ has been chosen over Kafka for its ease of deployment and easy integration with Node-Red thanks to the availability of an AMQP connector plugin for Node-Red. However, all three messaging systems have been found to be suitable for the given requirements and replacing RabbitMQ with a different messaging system is possible at any time.

[2] http://json-schema.org/
[3] https://kafka.apache.org
[4] https://www.rabbitmq.com/
[5] http://zeromq.org

Criteria	**Kafka**	**RabbitMQ**	**ZeroMQ**
License	Apache License 2.0	Mozilla Public License	LGPLv3
Publish-Subscribe	✓	✓	✓
Push-Pull	✓	✓	✓
Topic Routing	✓	✓	subscription filters
JavaScript Library	✓	✓	✓
Protocols	Kafka Wire Protocol	AMQP, MQTT, STOMP, HTTP	ZMTP
Clustering	✓uses Zookeeper for client and cluster coordination	✓load balancer, dynamic DNS service, mobile IP or similar service recommended for client-cluster connection	✓distributed
Delivery Guarantees	at-least-once	at-least-once, at-most-once	none

Table 5.1.: Brief comparison of messaging systems

6. Implementation of the Design in SitOPT

6.1. Implementation

6.1.1. Context Provider Component

The Context Provider serves three main goals:

1. It needs to receive or retrieve and process information about objects in the physical layer.
2. It has to persist sensor values and context information.
3. It must provide context information for other components in a unified way.

Previously the RMP handled these tasks, though as noted in section 5.1.1, various limitations of the RMP implementation make it a bad fit for the running scenario and integration with other components - especially the lacking support to push updated values upstream and its underlying assumptions about the accessibility of sensors in the physical layer. The RMP exposes sensor values via an HTTP interface. The interface is used to register objects and sensors but also by the situation recognition to retrieve sensor values. The RMP implementation is based on ActionHero[1]. The ActionHero framework adds a lot of complexity, but hardly any of its builtin features are needed for the tasks that the RMP or the context management component respectively need to handle. As the HTTP API only consisted of a few resources, it has been decided to completely replace the ActionHero based implementation with a new and lighter implementation.
The context management component has been re-implemented with the more lightweight *Express.js*[2] framework which provides basic features for building web services.

[1] http://www.actionherojs.com/
[2] http://expressjs.com/

The first task, processing messages from the physical layer, is handled by appropriate adapters. Express.js allows developers to write plugin modules that can be easily added to Express.js based applications. This plugin concept is exploited in the Context Management Platform to allow registering appropriate adapter modules for handling all types of physical sensors. For the running scenario an adapter has been implemented which receives and processes HTTP requests that are sent by the simulated telematics boxes.

The second task, persisting context, is handled in a persistence subsystem. Context information does not follow a strict schema, which is why the document oriented MongoDB has been chosen for this task. MongoDB does not enforce a certain schema and stores collections of documents in BSON format (binary JSON).
The physical layer is logically represented in the context manager. Each monitored object is represented as a document in MongoDB. Each object may have multiple sensors. These sensors are maintained in a nested collection for each object. The reasoning behind this modeling decision is elaborated in [28]. Sensors are usually retrieved in the context of an object. By embedding a sensor collection all relevant information can be retrieved in a single query. Frequent updates to the sensor collection are not expected.
With the logical representation of the physical structure in place, the context manager can associate sensor values from incoming messages with their corresponding logical sensors and objects via identifiers or other unique attributes that are present in the messages.
New sensor values are expected to be inserted very frequently. Depending on the domain, sensors may send messages in intervals of few milliseconds to several hours. It is thus a bad idea to store sensor values as nested collections inside sensor documents. The nested collections would grow quickly and eventually reach MongoDBs document size limit of 16 megabytes. Because of the expected growth, sensor values will be stored in a separate collection. Each document in this collection will be stored together with a reference to the corresponding object and sensor. Reference attributes and the timestamp attribute will be indexed to support efficiently querying for the latest values by object and sensor.
In the Node.js application, access to the MongoDB is handled with *Mongoose*[3]. Mongoose is an object document mapper which provides additional features on top of the native JavaScript driver for MongoDB [18]. Minimum schema requirements are modeled in the application and verified by Mongoose before documents are written to MongoDB.

The third and most important task for SitOPT is the provisioning of context information to other components. There are two primary use cases which must be supported by the context manager: The first one is the management of (logical) objects and sensors, the second one is pushing context information to the situation recognition component.

[3] http://mongoosejs.com/

Object and sensor management will be done by users via a graphical user interface. The management interface is hence exposed as an HTTP API. The previous API of the RMP has been redesigned, loosely following REST principles[4]. The API was first modeled using the Rest API Modeling Language (RAML[5]) and then implemented in the application so that objects and sensors can be registered by HTTP clients, such as applications in a browser or on mobile devices. The RAML API specification serves as a base for automated API testing and as documentation for the HTTP interface.
The HTTP API is not suitable for pushing context information to the situation recognition component but it can serve requests for specific information via the HTTP API. The push behavior has been implemented by exploiting a feature of Mongoose. Mongoose allows registering hooks which are automatically executed when a document is saved to the database. A hook has been implemented which takes the newly created document and submits it to the situation recognition. Whenever a new sensor value is stored in the context manager this hook is executed.

6.1.2. Quality Assessment Component

The quality assessment component is implemented as a stateless Node.js application which uses the strategy pattern in order to allow the use of different quality assessment approaches (see Figure 6.1). The quality assessment component receives messages which consist of a detected situation, its conditions and meta data that have been added by the situation detection component. A quality estimator class looks up the desired assessment strategy in the situation's meta data and then retrieves the quality context using the according concrete AssessmentStrategy. The obtained quality context is added to the situation's meta data and then the annotated situation is submitted to the message broker.

Not all methods that have been described in chapter 4 can be implemented as a strategy class in the quality assessment component. E.g. the confidence reinforcement method cannot be implemented in a strategy class without violating the constraint that quality assessment instances must be stateless. A stateful implementation would require a separate data store which can be accessed by multiple quality assessment instances simultaneously.
The filtering approach plays a special role here. It could be implemented as a strategy class, however the quality assessment component is not a good place to perform filtering. Filtering should be performed before the situation recognition, otherwise no performance gains will be achieved by filtering.

[4] the hypermedia constraint has been neglected for simplicity
[5] http://raml.org/

```
{
  "name": "memory",
  "id": "memory",
  "children": {
    "operation": "and",
    "items": [
      {
        "name" : "free RAM",
        "operator": "min",
        "value": 30,
        "context": {
          "freeMem2" : 40,
          "quality" : 0.5,
          "timestamp" : 876786
        },
        "fulfilled": true
      }
    ]
  },
  "meta": {
    "strategy": "weightedAvg",
    "timeDetected": 876966
  }
}
```

Listing 2: Example message sent by the situation recognition component to the quality assessment component

Due to possibly differing processing times in the Quality Assessment instances, situations might be submitted to the exchange out of order. The confidence reinforcement method as introduced in section 4.6 requires ordered inputs though.

6.1.3. Situation Recognition

The situation recognition component must be connected to the message broker, so that it can receive context and submit detected situations to the quality assessment component. In the current SitOPT implementation Node-Red flows are created using a mapping library which transforms situation definitions into JSON files that represent a Node-Red flow. For the running scenario a Node-Red flow has been created manually so that the new requirements could be incorporated into the flow. The basic flow subscribes to GPS context data and also to traffic context data. It includes a filtering step to discard context information with low quality and contains function nodes for caching the latest value and for the actual situation detection logic. The flow is depicted in Figure 6.2.

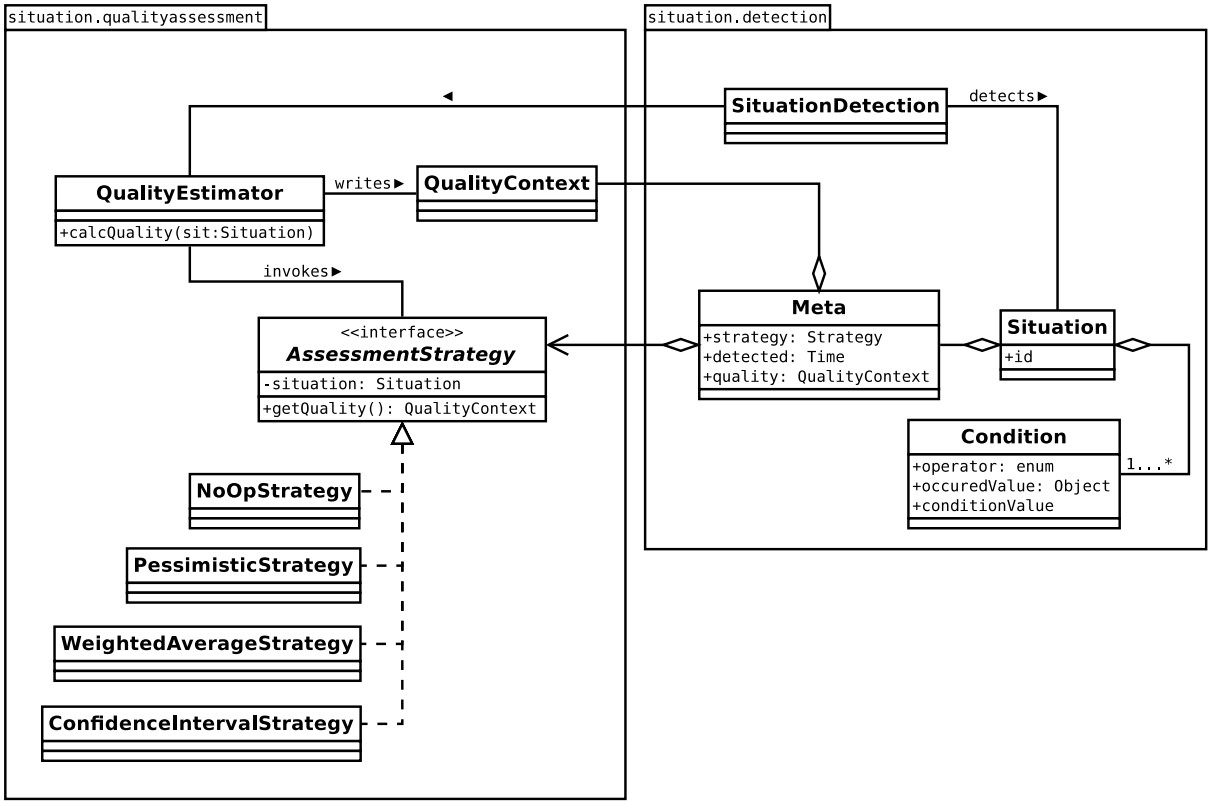

Figure 6.1.: Simplified class diagram of the quality assessment package

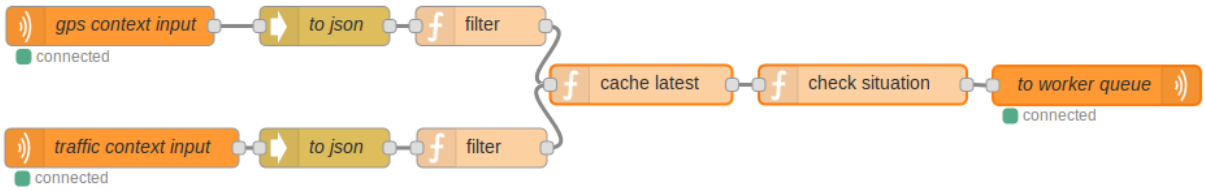

Figure 6.2.: Basic Node-Red flow for processing context data

An AMQP plugin has been used to add node types to Node-Red which can be used as sources and sinks in the flow. The AMQP source nodes inject new messages into the flow. The filtering nodes can be used to discard messages whose quality attributes are below a per context type configurable threshold to support input filtering as described in section 4.2.
A caching node stores the latest context object in the flow context, so that the situation check can always access the latest context information that were fed into the flow. The flow internally stores the situation template as a JSON representation and flags conditions as fulfilled or not fulfilled as they are processed until eventually the resulting

JSON object looks like the exemplary message given in Listing 2. This message is then passed to the final node in the flow. This final AMQP node submits the message to the worker queue.

6.2. Shortcomings of the Implementation

The implementation has several underlying assumptions which are specific to the test scenarios. As of now only Node-Red is supported for situation recognition and no mapping from situation templates to Node-Red flows is performed. This is deliberate to allow more flexibility when experimenting with different flows and message formats in Node Red. The previously used mapping library did not allow such flexibility.

The fleet simulation adapter is integrated into the prototypical context manager implementation. But in order to support multiple data sources, adapters should be pluggable modules/plugins. A possible plugin mechanism which exploits the Node Package Manager has been proposed in chapter 5.

The confidence interval preprocessor which has been implemented as part of the quality assessment component uses a fixed value of 5% of the indicated number range to determine if the QoC score should be lowered or raised. This number has been chosen to suit the data used in the simulated scenario. Rather than using a fixed number, the value should be based on standard deviation which can be obtained by observing the sensor data that flows through the system. Such statistical meta information could be provided by the context manager so that the quality assessment component can easily use this information at runtime.

A graphical interface was not needed for the simulation runs and thus has not been implemented for the context manager, so that values must be added manually via the HTTP interface or directly via a MongoDB client.

7. Evaluation

7.1. Simulated Scenario

The scenario as described in section 1.2 has been modeled for multiple simulation runs. The simulation has been implemented as a completely independent project and has been released on Github[1] under the MIT license.

The simulation uses the publicly available Open Street Map data for Germany and the open source library Graphhopper[2] to generate realistic routes. The generated routes and generated traffic incidents serve as the simulation model. The driving behavior and locations of the simulated trucks is controlled by the underlying simulation model. The model's values are assumed to be the correct values. The model also has information about traffic incidents, so that trucks can be slowed down when they enter a traffic jam. Not all traffic jams will be provided via traffic reports to the situation detection service and some traffic reports will be of low quality on purpose. Traffic information have been made available via a RESTful API which serves JSON responses.
Each simulated truck sends status updates with telemetry data to the situation recognition service. These telemetry data diverge from the real values to imitate behavior of real sensors and do not cover all information which are present in the simulation model. How the model's values have been used to derive realistic telematics box messages and a mocked traffic information service is further described in section 7.2. The scenario has been modeled with three trucks which all drive to the same destination (the factory). The routes are shown in Table 7.1.

The *South* route is used to verify that situations are properly detected when they occur. The *West* route contains two traffic incidents which will slow down the truck, however none of those incidents are reported via the traffic API. This route is used to test if situations are detected when one context provider is not reporting any data. The *North* route does not contain any traffic jams, but the traffic API reports two traffic incidents

[1] https://github.com/fleetsim
[2] https://graphhopper.com

Route	modeled incidents	reported incidents
South	2	2
West	2	0
North	0	2

Table 7.1.: Modeled routes and traffic incidents

on the route. This route has been modeled to test if false positives are detected, and if so, if the quality score reflects this circumstance.

All implemented quality assessment methods are fed with the fabricated sensor data and traffic information. The quality assessment results have been captured for evaluation.
A second simulation has been performed with 2000 trucks on random routes, a push interval of 1 second and a simulation interval of 100ms, resulting in 20000 messages/s. This time throughput and resource utilization has been captured for a performance comparison.

7.2. Test Data Generation

In the fleet telematics scenario, the situation recognition service has two sources of information - the telemetry data provided by the hardware boxes mounted to each truck and a traffic information service.

Obviously the model's data cannot be exposed to the situation recognition service directly as that would not measure the quality assessment methods but only how well the situation recognition resembles the model. Instead, a subset of data must be provided to the situation recognition in a way that imitates realistic telematics boxes and traffic information services. For this purpose, an additional layer has been used on top of the simulation model to fabricate derived data. This is depicted in Figure 7.1.

Each message sent by a telematics box contains a boxId, latitude, longitude, speed and a timestamp. It is assumed that clocks are synchronized and that messages always arrive in order.
Telematics boxes also report the horizontal accuracy of the GPS sensor via the `horizontalAccuracy` field in meters. This field contains the distance of the reported point in meters to the model's position with a granularity of five meters. An exemplary message is shown in Listing 3.

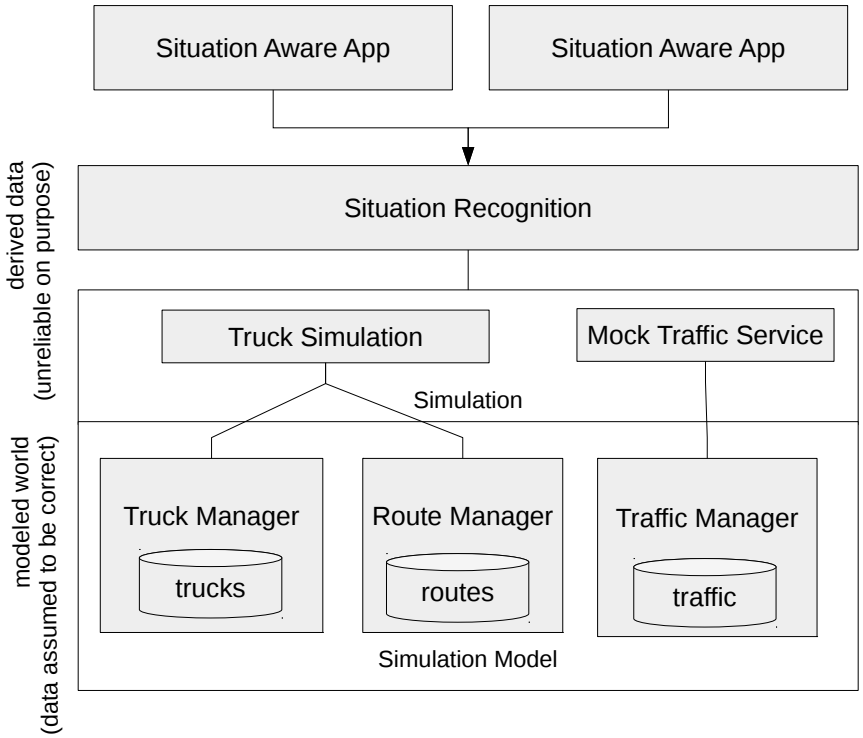

Figure 7.1.: High level view of the components involved in the simulation runs

```
{
  "boxId" : 68769876,
  "temp" : 22.45,
  "payloadTemp" : -4.32,
  "humidity" : 40.12,
  "latitude" : 12.0898765,
  "longitude" : 55.9798756,
  "altitude" : 145.23,
  "horizontalAccuracy": 5,
  "verticalAccuracy": 10,
  "timestamp" : 1461180274
}
```

Listing 3: Example JSON message emitted from a simulated telematics box

7.3. Simulation Results

7.3.1. Weighted Averages Strategy

All routes listed in Table 7.1 have been assessed with the Weighted Averages strategy in a simulation run. For simplicity wieghts have been chosen to be equally distributed between both situation conditions.

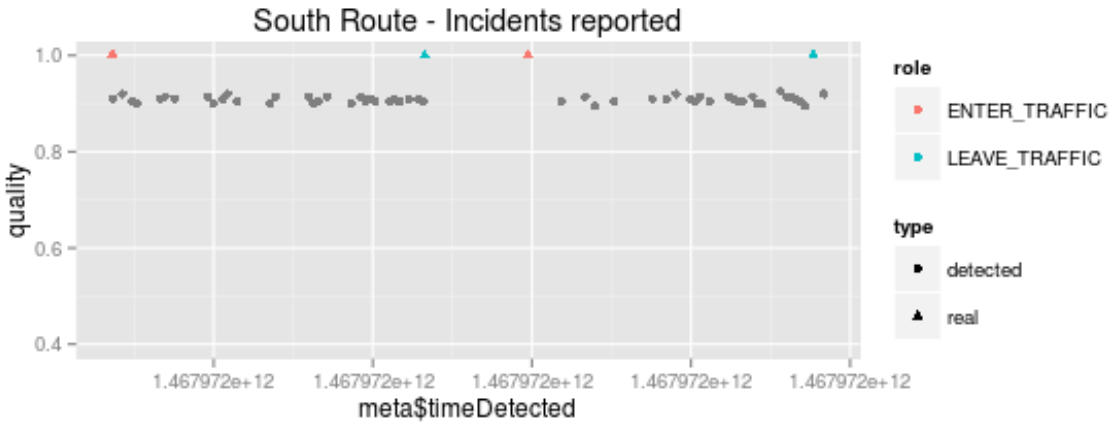

Figure 7.2.: Simulation results with the average strategy for the south route

Figure 7.2 shows detected situations' qualities for the south route on the y-axis and time on the x-axis. Markers indicate when the truck actually entered and left a traffic jam as reported by the simulation. Comparing these markers with the detected situations one can see that only one false positive has been detected after the truck has left the second traffic incident. The second traffic incident has first been reported with a small delay. This delay is a consequence of the update interval with which the distance of the trucks position to the reported traffic incident is calculated in the Context Manager component. Detected situations were reported with high confidence. The reported Quality of Situation was close to 90 %. However the situation has not been detected with each interval. This is due to the speed threshold used in the situation template. Inaccurate GPS context data resulted in reported velocity which exceeded the threshold and as a consequence situations were not detected at all.

Figure 7.3 shows the results for the west route. On this route two traffic jams were present but not reported by the traffic service. Due to the `AND` connection of both nodes in the situation template no situations were detected at all.

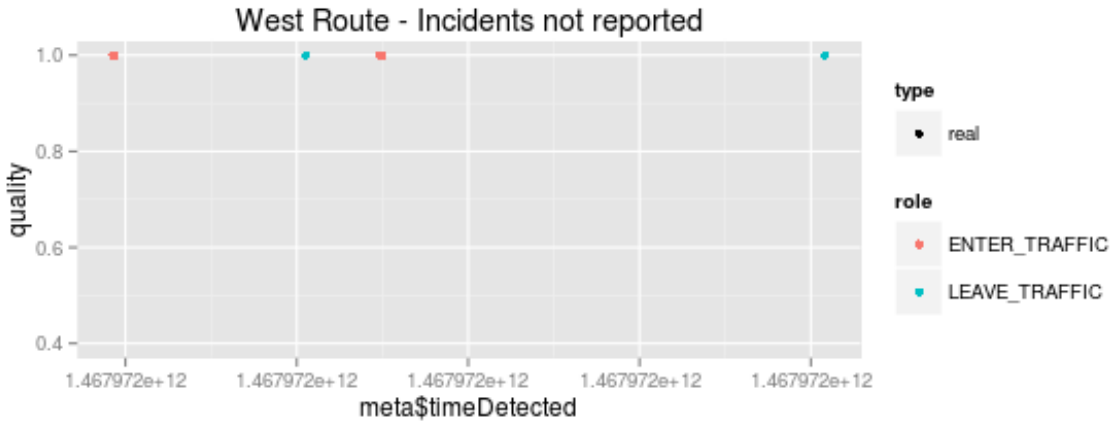

Figure 7.3.: Simulation results with the average strategy for the west route

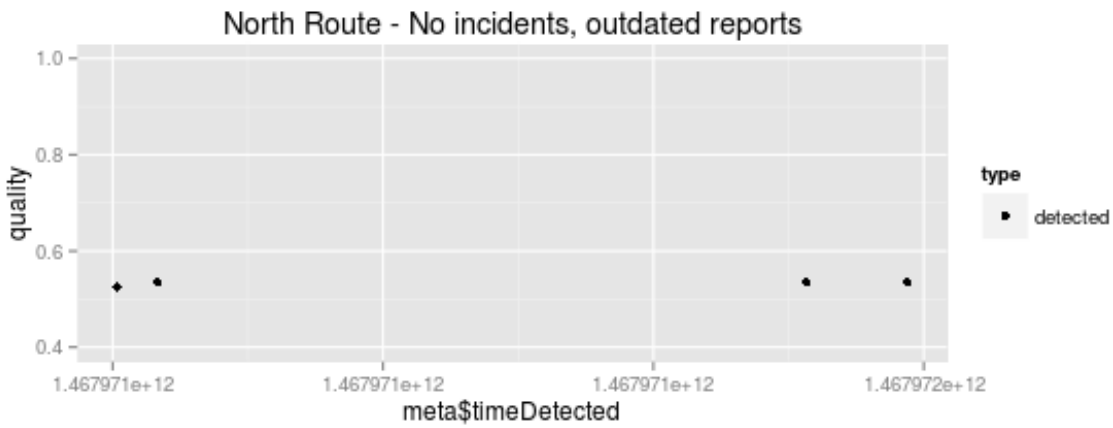

Figure 7.4.: Simulation results with the average strategy for the north route

Figure 7.4 shows the north route. On this route no traffic incidents were present, but two incidents with an age of 60 minutes were reported by the traffic service. Only a few false positives were reported by the situation recognition as the truck's speed was higher than the threshold most of the time. The quality of these false positives was reported with slightly more than 50 %.

7.3.2. Weighted Averages Strategy with Confidence Intervals Preprocessing

The same simulation has been run with the confidence intervals method as a preprocessing step. The situation template has been extended with expected value ranges that can be reported by the sensors. These value ranges are used in the preprocessing step to determine if the confidence scores should be unchanged, lowered or raised. The threshold values have been increased to 5 m/s and 1500m respectively in an attempt to reduce the number of false negatives. Measurements which are close to the threshold (5% of the indicated sensor range or less) will be preprocessed to have a lower quality. This accounts for the less restrictive threshold in comparison to the previous testrun without preprocessing. When measurements were further away from the threshold than 5% of the indicated range, then they received a quality score improvement. The results of the simulation run with preprocessing are provided in Figure 7.5 and Figure 7.6. The plot for the west route has been omitted as it is identical with the previous run (no situations were detected).

Figure 7.5.: Simulation results with the average strategy and preprocessing for the south route

Figure 7.5 reveals that the situation has been recognized more frequently than before. Additional points have been recognized with slightly lower quality due to the changed threshold and the quality penalty which was applied in the preprocessing step when measurements were close to that threshold.

Figure 7.6 shows the recognized situations with the averages strategy and the confidence interval preprocessing step. More false positives have been recognized in this simulation

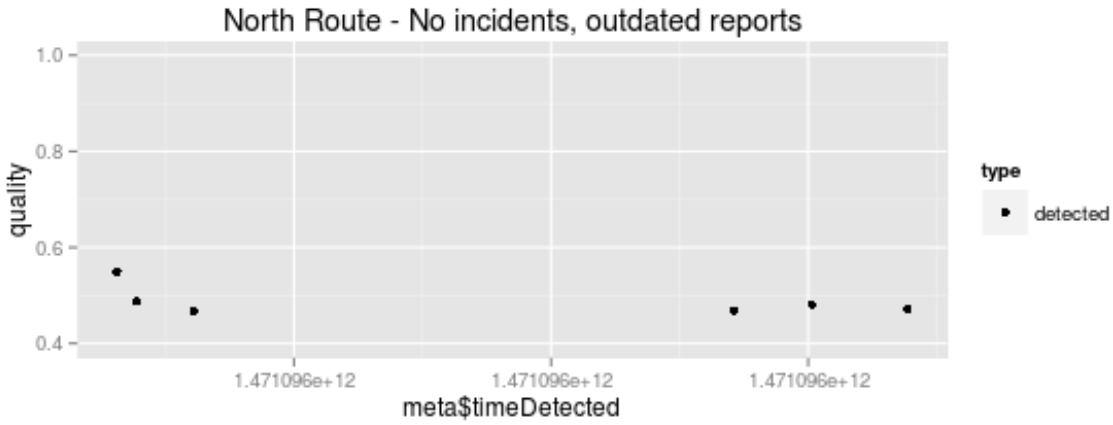

Figure 7.6.: Simulation results with the average strategy and preprocessing for the north route

run, however it can also be seen that quality scores are slightly lower than without preprocessing.

8. Conclusion

A collection of methods has been introduced which can easily be integrated in all types of stream processing systems to make them quality aware. Using a prototypical implementation it has been shown that these methods can be combined to determine the Quality of Situation in a dedicated component which is decoupled from the actual situation detection step. It has thus been shown that the tested approaches are not bound to Node-Red as a stream processing system, but are suitable for other stream processing systems as well.
A standalone fleet telematics simulation project[1] has been created to generate realistic telematics data. The simulation project itself is a major contribution which allows creating more fleet telematics related test scenarios in the future. The effectivenes of the introduced quality concept has been evaluated in a realistic fleet telematics scenario which has been modeled using the simulation project.
Furthermore the SitOPT project has been extended with new components and a new context management component to overcome limitations that prevented testing more complex scenarios like the fleet telematics scenario which was used in this evaluation. The created components have been released as open source projects on Github[2]. The newly created context manager now allows adding adapters for various scenarios and physical structures and introduces the possibility to create context resources to combine sensor data from multiple objects into a single collection.

The simulation results have shown that using AND connections, one risks that situations are not detected at all due to unreported (traffic) events. In such cases situation templates should use less restrictive thresholds combined with the Confidence Interval Preprocessor. This reduces the number of false negatives while adjusting the QoC scores appropriately.

The most important component in the overal system is the context manager. Wrong or inappropriate QoC scores, which leave the context manager cannot be corrected throughout the following process and impact the resulting Quality of Situation. This was the case with the speed values in the simulation. The QoC scores for GPS data was based

[1] https://github.com/fleetsim
[2] https://github.com/sitQa

on the horizontal GPS accuracy, speed however has been calculated using the distance to the previous point. The quality of the previous measurement thus impacted the accuracy of the speed value. This fact is not reflected in the QoC score provided by the context manager, which resulted in inappropriate results. Domain specific knowledge is required to properly design and implement adapters in the context manager and has the largest quality impact on the system outputs.

SitOPT's goal of providing a generalistic way to model situations with situation templates is achievable, even in more complex settings. However, during the implementation and evaluation of the quality concept it became obvious, that one elementary prequesite for this goal is a thoroughly designed context manager component. A generalistic approch in the situation recognition is only possible if complexity is abstracted on a lower level and when domain specific knowledge is embodied in adapters which provide the unified access to sensor information that are required by the situation detection system.
The existing schema for situation templates has to be extended to support quality aspects. The author of this thesis proposes adding additional fields for meta data on the situation level, as well as on the condition level, so that strategies, preprocessors (including input stream filtering) and postprocessors (such as the confidence reinforcement strategy) can be specified on a per situation template basis.
In the prototypical implementation of the quality concept, the limitation that only single objects can be used in situation templates has been overcome by creating artificial context objects in the context manager and a manual flow in Node-Red. Changes need to be made to the situation template schema if such scenarios should be supported. Depending on the complexity of situations that ought to be detected in SitOPT, the author would like to encourage considering a domain specific language as a more powerful alternative to the XML based situation templates.
In this work, only a single scenario has been used to evaluate the developed quality concept. Further scenarios, ideally with real world data should be tested to gain further knwoledge about the types of scenarios and situations where which quality assessment method is most appropriate.

A. Appendices

A.1. NodeRed Flow

A.1.1. AMQP Topology Configuration

```
{
        "exchanges": [
                {"name": "context", "type": "topic", "options": {"durable": false}},
                {"name": "situations", "type": "direct", "options": {"durable": false}}
        ],
        "queues": [
                {"name": "traffic-queue-south", "options": {"durable": false }},
                {"name": "gps-queue-south", "options": {"durable": false }},
                {"name": "situations.raw"}
        ],
        "bindings": [
                {
                        "source": "context",
                        "queue": "traffic-queue-south",
                        "pattern": "577e7328a7f7277602814769.traffic"
                },
                {
                        "source": "context",
                        "queue": "gps-queue-south",
                        "pattern": "577e7328a7f7277602814769.gps"
                }
        ]
}
```

Listing 4: Topology configuration in the NodeRed AMQP input node to receive context from RabbitMQ

A.1.2. Function Node for Caching Context Data

```
if(msg.payload.type == 'traffic') {
    flow.set('traffic',msg.payload);
} else {
    flow.set('gps', msg.payload);
}
return msg;
```

Listing 5: Code of the function node in the NodeRed flow for storing context information in the flow's context

A.1.3. Function Node for Situation Detection

```
var trafficContext = flow.get('traffic');
var gpsContext = flow.get('gps');

if(trafficContext !== undefined && gpsContext !== undefined) {
    var sitTemplate = {
      "name": "truckInTraffic",
      "id": "truckInTraffic",
      "objectId": gpsContext.objectId,
      "children": {
        "operation": "and",
        "items": [
          {
            "name" : "speed low",
            "operator": "max",
            "value": 5,
            "context": gpsContext,
            "fulfilled": gpsContext.speed <= 5,
            "meta": {"range": [0,35], "valueKey": "speed"}
          },
          {
            "name" : "traffic nearby",
            "operator": "max",
            "value": 1500, // increased from 1000 to 1500 for preprocessor test
            "context": trafficContext,
            "fulfilled": trafficContext.trafficObj.distance <= 1500,
            "meta": {"range": [0,10000], "valueKey": "traffic.distance"}
          }
        ]
      },
      "meta": {
        "strategy": "weightedAvg",
        "preprocessor": "confidenceInterval",
        "timeDetected": (new Date()).getTime()
      }
    };

    if(sitTemplate.children.items[0].fulfilled && sitTemplate.children.items[1].fulfilled) {
        return sitTemplate;
    }
}
```

Listing 6: Code of the function node in the NodeRed flow for detecting a situation

A.2. Zusammenfassung

SitOPT ist ein Forschungsprojekt um den Situationskontext für situationssensitive Anwendungen zur Verfügung zu stellen. Im Rahmen der Arbeit wurde ein Qualitätskonzept entwickelt und im SitOPT Projekt beispielhaft implementiert. Die bestehenden SitOpt Implementierungen wurden so erweitert, dass Situationskontext mit Qualitätsdaten annotiert werden bevor sie den situationssensitiven Anwendungen zur Verfügung gestellt werden. Diese Metainformationen werden als *Quality of Situation (QoSit)* bezeichnet.
Zwar existieren bereits Systeme zur Ermittlung der Qualität von Kontextdaten, diese eignen sich jedoch nicht für die Verwendung mit erkannten Situationen, welche aus verschiedenen Kontextinformationen ermittelt werden.
Basierend auf der Annahme, dass Kontextinformationen, welche für die Situationserkennung verwendet werden, bereits mit Qualitätswerten annotiert sind wurden verschiedene Methoden entwickelt um diese Werte zu einem *Quality of Situation* Wert zusammenzuführen. Mittels Filterung von eingehenden Nachrichten mit niedrigen QoC Werten kann eine minimale Qualität der erkannten Situation sicher gestellt werden. Bei der *Weighted Averages Methode* werden im Situationstemplate Gewichte für die einzelnen Kontextinformationen vergeben um schließlich über ein gewichtetes Mittel einen finalen Wert für die erkannte Situation zu ermitteln. Der pessimistische Ansatz wählt den niedrigsten Qualitätswert aus den verwendeten Kontextinformationen zur Bestimmung der Situationsqualität. Zudem wurden Methoden vorgestellt, die zusätzlich zu den oben genannten Methoden vor bzw. nach der Qualitätsberechnung durchgeführt werden können um das Ergebnis der Qualitätsberechnung zu beeinflussen. Die vorgestellten Methoden können zu hybriden Ansätzen kombiniert werden um domänenspezifische Anforderungen besser bedienen zu können.

Für die Implementierung des Qualitätskonzeptes wurde der bestehende Webserver, welcher Kontextinformationen für die Situationserkennung bereitstellt mit einer Neuentwicklung ausgetauscht. Die Qualitätsbewertung wurde in einem neuen Teilprojekt implementiert welches komplett Zustandslos arbeitet und von den anderen Komponenten entkoppelt ist. Es wurde eine Message Queue in die Architektur eingeführt um die SitOPT Komponenten zu integrieren und weitere Anforderungen zu erfüllen, die sich aus dem gewählten Testszenario ergeben haben.
Für die Evaluierung wurde eine Flotte von LKW simuliert die in regelmäßigen Abständen Telematikdaten an die neue Kontextmanagerkomponente senden. Zusätzlich wurde ein Webservice mit aktuellen Staumeldungen simuliert. Die Telematiknachrichten sowie die Staumeldungen wurden in SitOPT ausgewertet um zu erkennen ob sich Trucks in einem Stau befinden. Qualitätswerte wurden für erkannte Situationen mit der *Weighted Averages Methode*, sowie mit einer hybriden Methode von der neuen Komponente berechnet. Die Ergebnisse wurden mit dem tatsächlichen Status der simulierten LKW verglichen um die Zuverlässigkeit der Qualitätswerte zu prüfen.

Die Auswertung der Ergebnisse hat gezeigt, dass der hybride Ansatz die Stausituation häufiger erkennt. Die Qualitätswerte der Kontextinformationen haben einen großen Einfluss auf die Korrektheit der berechneten QoSit Werte. Bei einem Ausfall des Webservices mit Staumeldungen bzw. fehlenden Stauinformationen wurden Situationen nicht erkannt. False Positives waren mit deutlich schlechteren Qualitätswerten annotiert als True Positives.

Bibliography

[1] Russell L Ackoff. "From data to wisdom." In: *Journal of applied systems analysis* 16.1 (1989), pp. 3–9.

[2] A. Artikis et al. "Self-adaptive event recognition for intelligent transport management." In: *Big Data, 2013 IEEE International Conference on*. Oct. 2013, pp. 319–325. DOI: 10.1109/BigData.2013.6691590.

[3] Payam Barnaghi, Maria Bermudez-Edo, and Ralf Tönjes. "Challenges for Quality of Data in Smart Cities." In: *J. Data and Information Quality* 6.2-3 (June 2015), 6:1–6:4. ISSN: 1936-1955. DOI: 10.1145/2747881. URL: http://doi.acm.org/10.1145/2747881.

[4] Thomas Buchholz and Michael Schiffers. "Quality of Context: What It Is And Why We Need It." In: *In Proceedings of the 10th Workshop of the OpenView University Association: OVUA'03*. 2003. DOI: 10.1.1.147.565.

[5] X. Chuanfei et al. "Complex Event Detection in Probabilistic Stream." In: *Web Conference (APWEB), 2010 12th International Asia-Pacific*. Apr. 2010, pp. 361–363. DOI: 10.1109/APWeb.2010.56.

[6] Jonathan Coe, Robert Prime, and Rebecca Jest. *A comprehensive guide to fleet tracking systems*. 2015. URL: http://www.telematics.com/a-comprehensive-guide-to-fleet-tracking-systems (visited on 04/16/2016).

[7] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Pubs Co Series. Manning, 2011. ISBN: 9781935182214.

[8] Apache Foundation. *Apache Kafka Documentation*. 2016. URL: https://kafka.apache.org/documentation.html (visited on 07/10/2016).

[9] A. Goel. *Fleet Telematics: Real-time management and planning of commercial vehicle operations*. Operations Research/Computer Science Interfaces Series. Springer US, 2007. ISBN: 9780387751054.

[10] F. Hermans, N. Dziengel, and J. Schiller. "Quality estimation based data fusion in wireless sensor networks." In: *Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on*. Oct. 2009, pp. 1068–1070. DOI: 10.1109/MOBHOC.2009.5337006.

[11] Pascal Hirmer et al. "SitRS - A Situation Recognition Service based on Modeling and Executing Situation Templates." Englisch. In: *Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing*. Ed. by Johanna Barzen et al. Vol. RC25564. Technical Paper. IBM Research Report, Dezember 2015, pp. 113–127. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2015-34&engl=.

[12] Kuan-Tse Huang, Yang W. Lee, and Richard Y. Wang. *Quality Information and Knowledge*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. ISBN: 0-13-010141-9.

[13] Armin Hüneburg et al. *SitOPT Repository on Github*. 2016. URL: https://github.com/mormulms/SitOPT (visited on 08/24/2016).

[14] Pivotal Software Inc. *RabbitMQ Documentation*. 2016. URL: https://www.rabbitmq.com/features.html (visited on 07/10/2016).

[15] H. Kawashima, H. Kitagawa, and X. Li. "Complex Event Processing over Uncertain Data Streams." In: *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on*. Nov. 2010, pp. 521–526. DOI: 10.1109/3PGCIC.2010.89.

[16] Christian Kuka and Daniela Nicklas. "Quality Matters: Supporting Quality-aware Pervasive Applications by Probabilistic Data Stream Management." In: *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*. DEBS '14. Mumbai, India: ACM, 2014, pp. 1–12. ISBN: 978-1-4503-2737-4. DOI: 10.1145/2611286.2611292. URL: http://doi.acm.org/10.1145/2611286.2611292.

[17] Christian Kuka and Daniela Nicklas. "Scalable Uncertainty Management: 6th International Conference, SUM 2012, Marburg, Germany, September 17-19, 2012. Proceedings." In: Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Chap. Approximating Complex Sensor Quality Using Failure Probability Intervals, pp. 287–298. ISBN: 978-3-642-33362-0. DOI: 10.1007/978-3-642-33362-0_22. URL: http://dx.doi.org/10.1007/978-3-642-33362-0_22.

[18] LearnBoost. *Mongoose Documentation*. 2016. URL: http://mongoosejs.com/docs/guide.html (visited on 06/10/2016).

[19] Paul Lillrank. "The quality of information." In: *International Journal of Quality & Reliability Management* 20.6 (2003), pp. 691–703. DOI: 10.1108/02656710310482131. URL: http://dx.doi.org/10.1108/02656710310482131.

[20] H. Liu et al. "Faulty sensor detection, identification and reconstruction of indoor air quality measurements in a subway station." In: *Control, Automation and Systems (ICCAS), 2011 11th International Conference on*. Oct. 2011, pp. 323–328.

[21] Christoffer Löffler, Christopher Mutschler, and Michael Philippsen. "Approximative Event Processing on Sensor Data Streams." In: *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*. DEBS '15. Oslo, Norway: ACM, 2015, pp. 360–363. ISBN: 978-1-4503-3286-6. DOI: 10.1145/2675743.2776767. URL: http://doi.acm.org/10.1145/2675743.2776767.

[22] Pierrick Marie et al. "Modeling and Using Context: 8th International and Interdisciplinary Conference, CONTEXT 2013, Annecy, France, October 28 -31, 2013, Proceedings." In: ed. by Patrick Brézillon, Patrick Blackburn, and Richard Dapoigny. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Chap. QoCIM: A Meta-model for Quality of Context, pp. 302–315. ISBN: 978-3-642-40972-1. DOI: 10.1007/978-3-642-40972-1_23. URL: http://dx.doi.org/10.1007/978-3-642-40972-1_23.

[23] Pierrick Marie et al. "QoC-aware Context Data Distribution in the Internet of Things." In: *Proceedings of the 1st ACM Workshop on Middleware for Context-Aware Applications in the IoT*. M4IOT '14. Bordeaux, France: ACM, 2014, pp. 13–18. ISBN: 978-1-4503-3234-7. DOI: 10.1145/2676743.2676746. URL: http://doi.acm.org/10.1145/2676743.2676746.

[24] M. Mathew, Ning Weng, and L. J. Vespa. "Quality-of-information modeling and adapting for delay-sensitive sensor network applications." In: *Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International*. Dec. 2012, pp. 471–477. DOI: 10.1109/PCCC.2012.6407659.

[25] Rene Mayrhofer, Harald Radi, and Alois Ferscha. "Recognizing and Predicting Context by Learning from User Behavior." In: *Radiomatics: Journal of Communication Engineering, special issue on Advances in Mobile Multimedia* 1.1 (May 2004), pp. 30–42. ISSN: 1693-5152.

[26] M.J. McGrath and C.N. Scanaill. *Sensor Technologies: Healthcare, Wellness and Environmental Applications*. Expert's Voice in Networked Technologies. Apress, 2013. ISBN: 9781430260134.

[27] James McNaull et al. "Data and Information Quality Issues in Ambient Assisted Living Systems." In: *J. Data and Information Quality* 4.1 (Oct. 2012), 4:1–4:15. ISSN: 1936-1955. DOI: 10.1145/2378016.2378020. URL: http://doi.acm.org/10.1145/2378016.2378020.

[28] MongoDB, Inc. *MongoDB Manual - embedding documents*. 2016. URL: https://docs.mongodb.com/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/ (visited on 06/10/2016).

[29] Débora Cabral Nazário et al. “An Enhanced Quality of Context Evaluating Approach in the e-Health Sensor Platform.” In: *Proceedings of the 11th ACM Symposium on QoS and Security for Wireless and Mobile Networks*. Q2SWinet ’15. Cancun, Mexico: ACM, 2015, pp. 1–7. ISBN: 978-1-4503-3757-1. DOI: 10.1145/2815317.2815320. URL: http://doi.acm.org/10.1145/2815317.2815320.

[30] Node.js Foundation. *Node.js v4.4.5 Documentation*. 2016. URL: https://nodejs.org/dist/latest-v4.x/docs/api/all.html (visited on 06/10/2016).

[31] A. Rahman, D. V. Smith, and G. Timms. “A Novel Machine Learning Approach Toward Quality Assessment of Sensor Data.” In: *IEEE Sensors Journal* 14.4 (Apr. 2014), pp. 1035–1047. ISSN: 1530-437X. DOI: 10.1109/JSEN.2013.2291855.

[32] tbd. *ZeroMQ Guide*. 2016. URL: http://zguide.zeromq.org/page:all (visited on 07/10/2016).

[33] X. Wang et al. *Information Quality*. Taylor & Francis, 2014. ISBN: 9781317467991. URL: https://books.google.de/books?id=fEnfBQAAQBAJ.

[34] M. Wieland et al. “Towards situation-aware adaptive workflows: SitOPT 2014; A general purpose situation-aware workflow management system.” In: *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*. Mar. 2015, pp. 32–37. DOI: 10.1109/PERCOMW.2015.7133989.

[35] Stephen S. Yau, Yu Wang, and Fariaz Karim. “Development of situation-aware application software for ubiquitous computing environment.” In: *Proc. 26th Int’l Computer and Software Applications Conf. (COMPSAC 2002*. 2002, pp. 233–238.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature