Open Access

Open Journal of Internet of Things (OJIOT)
Volume 8, Issue 1, 2022

http://www.ronpub.com/ojiot
ISSN 2364-7108

Generating SPARQL-Constraints for Consistency Checking in Industry 4.0 Scenarios

Simon Paasche[A], Sven Groppe[B]

[A]Automotive Electronics, Robert Bosch Elektronik GmbH, John.-F.-Kennedy-Strasse 43-53, Salzgitter, Germany, simon.paasche@de.bosch.com

[B]Institute of Information Systems, University of Lübeck, Ratzeburger Allee 160, Lübeck, Germany, groppe@ifis.uni-luebeck.de

Abstract

A smart manufacturing line consists of multiple connected machines. These machines communicate with each other over a network, to solve a common task. Such a scenario can be located in the Internet of Things (IoT) area. An individual machine can be perceived as an IoT device. Due to machine to machine communication, a huge amount of data is generated during manufacturing. This emerging data flow is an essential part of today's industry, as analyzing data helps improving processes and thus, product quality. To adequately make use of the collected data, we require a high level of data quality. In our work, we address the issue of inconsistent data in smart manufacturing and present an approach to automatically generate SPARQL queries for validation.

Type of Paper and Keywords

Regular research paper: *consistency checking, generating constraints, SPARQL, process optimization, industry 4.0*

1 Introduction

In the Internet of Things (IoT), many individual devices communicate with each other via a network [22] [23]. The purpose of this communication is to exchange data between one another. A data stream is emerging. Possible domains of such IoT scenarios range from energy supply [15] over healthcare [13] to smart industry [26]. In all of these domains, IoT devices are used to increase efficiency, conserve resources or optimize processes and medical treatments.

As data is a decisive factor in such smart environments [20], this work focuses on monitoring the data stream from an IoT application to identify inconsistencies. Therefore, we refer to a real IoT scenario in smart manufacturing at Bosch. Robert Bosch GmbH is a globally operating group with diverse markets. In the automotive electronics sector, among other things, highly complex control units for vehicles are manufactured. To ensure high quality standards of control units and thus, a safe end product, a core task at MFT1 department in Salzgitter is to evaluate data from manufacturing processes. By analyzing this data, we can avoid errors, conserve resources and make the overall manufacturing process more sustainable.

The lower half of Figure 1 shows the current data flow: Data is generated during manufacturing and propagated by the machines. This data flow is collected and stored. Directly storing data leads to a decrease in data quality, since the machine data may contain inconsistencies. Inconsistencies occur in such scenarios,

This paper is accepted at the *International Workshop on Very Large Internet of Things (VLIoT 2022)* in conjunction with the VLDB 2022 conference in Sydney, Australia. The proceedings of VLIoT@VLDB 2022 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

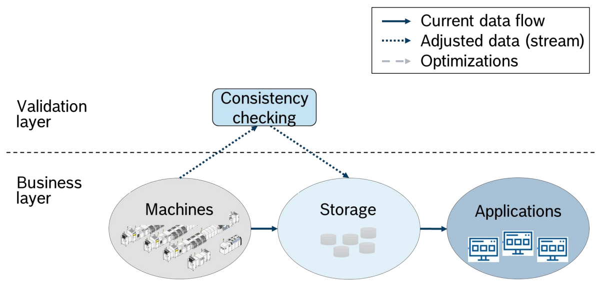

Figure 1: Application environment

due to a heterogeneous landscape [18] consisting of machines from various fabricators with diverse software versions. The existing data flow causes subsequent applications to access inconsistent data.

Currently, there is effort at Bosch to identify invalid data. This work is performed manually by creating SQL based constraints. This manual step is time-consuming [27] and has the drawback that only known inconsistencies can be identified.

Our work addresses the issue with potential inconsistent data. Therefore, we created SPARQL and SHACL constraints in a first step to identify limitations of current specifications. In [16], the authors pointed out that however SPARQL and SHACL are expressive enough, the constraints become very complex and error-prone. In addition, users need deep insights into the application of these technologies to their domain knowledge [10]. To overcome this problem, we aim to develop a framework to automatically generate constraints from a simple definition of consistency. To accomplish the problems, we pick up the challenges from [16] and adapt them in the following way: We still have to handle large data streams from heterogeneous sources. This task is splitted up into the challenges C1 to C3. In challenge C4, in contrast to improving the overall manufacturing processes, we solely focus on increasing data quality through consistency checking. Further, we add C5 to point out the necessity of validating new machine and line configurations.

C1 Handle Big Data. A typical manufacturing line at our plant includes at least six machines. Due to multiple lines at one plant, the machines generates over thousand messages each day.

C2 Handle continuous data streams. The machine in one line communicate via network. This communication results in a continuous data stream, which has to be processed immediately to receive a fast feedback [9].

C3 Handle heterogenous data sources. In the SMT area at Bosch, the worldwide production network includes more than 200 lines with over 3000 machines. Each machine can be considered as an IoT device and thus a data source.

C4 Enhance data quality. Data quality has an impact on subsequent analyses and thus, can reduce manufacturing costs, save resources, such as energy and raw materials and improve product quality.

C5 Fast deployment of machines and lines. After completion of machine and line configurations as well as new setups of entire lines, it should be ensured as quickly as possible that no inconsistencies occur during production.

To address our five challenges, the remainder of this paper is as follows: Section 2 provides an overview of related work in the field of data validation using semantic methods and automatic constraint generation. Afterwards, Section 3 describes our IoT environment and introduces the setup of a manufacturing line for printed circuit boards. Section 4 is the core of our work, as it describes our understanding of consistency and how we aim to automatically generate correct constraints to validate the data. Subsequently, Section 5 briefly presents preliminary results. Section 6 discusses our approach, before we finally conclude in Section 7.

2 Related Work

At the beginning of our work, we present related work in the area of data validation and automatic generation of SPARQL queries and constraints.

Baclawski et al. [1] and Steyskal et al. [19] present approaches on validating semantic models. Both works focus on violation detection between various models of the same physical system. In contrast, we have one semantic model of our system (SMT line) and continuously validate machine data against our model. Haav et al. [11] and Xuanyua et al. [25] semantically check product configurations to see if configurations are created according to defined specifications. Validation is performed using SHACL or SWRL. As in our scenario, knowledge is represented in ontologies. In contrast, we use SPARQL to create validation constraints on continuous data streams. Furthermore, [4] presents another practical example. The work focusses on medical sensor data stream sourced from healthcare apps. Cortes et al. use a similar data pipeline consisting, e.g. of a stream based cleaning and an analysis step. The authors do not provide concrete architectures and methods. [5] describes a distributed concept to handle inconsistent streams. The work assumes a fixed ontology and does not provide an application scenario. In [21], the authors focus on detecting and cleaning inconsistent data. As in our case, the validation constraints are exchangeable. As a disadvantage, Bleach can only detect and fix violations if the competing data tuples are in the same window. Our system checks for inconsistencies both within and between events. Through various case distinctions, each state can be uniquely classified as consistent or inconsistent.

In summary, we can state that for different domains systems exist in which data validation is performed using semantic web methods. The significant difference are on the one hand processing of a continuous data stream: most systems work with offline data, where full information is given. On the other hand, systems working on streams mostly process simple events, such as a sensor data flow. In contrast, our system handles complex JSON files.

The work of Corman et al. [3] describes an approach to automatically generate SPARQL queries from SHACL graphs. Following this approach, constraints can be defined using SHACL, even if just a SPARQL endpoint is present In contrast to SPARQL, SHACL was initially developed for validation purposes and, among others, offers a detailed violation report. Although their framework provides a good approach to the creation of queries, defining constraints in SHACL is not less complex than in SPARQL.

Further, there exist approaches to extract queries from ontologies. Chen et al. [2] give a detailed overview over existing approaches. Since we ask very specific queries regarding the consistency of the collected data, such an approach would produce too much overhead.

Jung et. al [12] and Sander et. al [17] present approaches to transform natural language into SPARQL queries. Applying NLP seems to be an interesting direction. In further work, we aim to explore this area, especially with regard to adaptability and overhead.

In contrast to all these contributions, we focus on an efficient, lean, and at the same time simple approach to validate continuous data streams in manufacturing scenarios. In particular, the creation of new constraints, even by employees without IT background, is of major interest for our business.

3 Connected Machines at Bosch

At Bosch plant in Salzgitter, printed circuit boards for control units are manufactured. Manufacturing such control units includes placement and soldering of electronic components on the boards This process is called surface-mounting technology (SMT).

A modern SMT line consists of machines to implement the SMT processes. These are mainly a machine for each sub-step: solder paste printing (SPP), solder paste inspection (SPI), surface mounted device (SMD), reflow soldering (RFL), and solder joint inspection (SJI). A product passes through these machines via a conveyor belt. The entire SMT process is largely automated.

During this automatic process, machines communicate with each other via a network. This communication results in a message flow. One message contains information about the finished process step. The data of an SMD machine contains, e.g. the placed components. An SJI machine propagates information about possible erroneous solder joints. These relevant process parameters are of interest, since analyzing them helps to improve product quality and enhance internal processes. During ongoing manufacturing and especially when configuring and adding new machines or lines, it must be ensured that these relevant parameters are correctly collected (C5). Therefore, we aim to validate the machine data to identify and eliminate inconsistencies. In particular, we are interested in inconsistencies that are currently unknown.

The following requirements result from the scenario and earlier work [16]:

R1 **Simplify the step of constraint creation.** The main intention of this work is, to make it easier for a user to create constraints and thus, validate the machine data stream.

R2 **Identify as much inconsistencies as for manually created queries.** With our new approach, we do not want to deteriorate in terms of inconsistency detection.

R3 **Identify unknown inconsistencies.** As an intensification to the previous requirement, we would like to reveal both known and unknown inconsistencies.

R4 **Reduce overhead while generating queries.** In total, we aim to not increase the computational overhead by comparing our framework against manually created queries. By computational overhead, we refer to execution time and memory resources of the overall algorithm. If we are able to decrease resource consumption, our algorithm becomes more sustainable in context of environmental impact.

R5 **Parallel query generation.** We want to ensure that our algorithm is scalable and will work for multiple parallel consistency checks. This requirement is an extension of R9 from [16].

R6 Expandability. In order to adapt to changes in manufacturing environment and to apply our approach in related domains, we have to ensure that our framework and especially, the query generation step, is expandable.

R7 Automatic creation of correct queries. The simplified language supports the user in the creation of constraints. In conjunction with the automated generation of SPARQL queries, this eliminates the error-proneness that exists with the manual creation of constraints.

To justify our constraint generation approach, it is mandatory that the framework is not only applicable to a few inconsistencies. In particular, we strive for the applicability of the approach to other manufacturing areas within and outside the Bosch group.

4 SPARQL Constraints for Data Validation

In our scenrio, consistency checking is the step of validating a continuous data stream, which emerges from the communication of connected machines By performing this step, we aim to identify inconsistent data to sustainably improve the overall data quality. To meet challenges C1 to C4 and our requirements R1 to R6, we have to handle the following three tasks:

T1 Building a query. In previous work [16], we compared SPARQL-ASK[1] queries and SHACL[2] shapes as semantic web technologies to validate consistency of messages from connected machines in manufacturing scenarios. Due to the fact that we had to include SPARQL queries in our SHACL shapes, we decided to solely use SPARQL-ASK for building up our constraints.

T2 Optimizing the created query. In order to fulfill requirement R3, we optimize the initially created query, e.g. to remove unnecessary parts from the query and adapt the triple patterns.

T3 Execute the query. The final task is to execute our optimized query. By executing the query, we check our constraint against available machine data. As a result, we receive an answer whether our data is consistent or not.

Figure 2 shows our architecture for consistency checking. We separately collect data from the processes of an SMT line. Machines send their data in JSON format. The data of each process is stored

[1] https://www.w3.org/TR/sparql11-query/
[2] https://www.w3.org/TR/shacl/

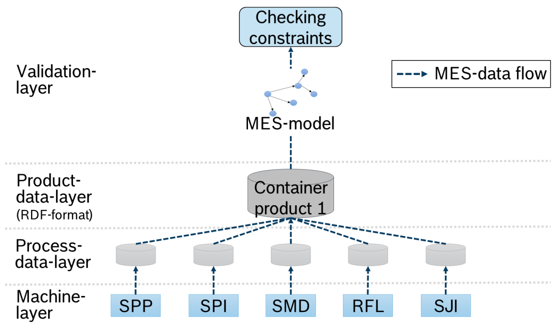

Figure 2: Data pipeline for consistency checking

in one container, allowing to draw conclusions about individual process steps in further analyses. To identify inconsistencies, we require a product based view, as inconsistencies refer to the data of a single product.

As discussed in [16], our overall architecture is scalable and can handle a continuous data stream. This ability is given through the layered architecture. Furthermore, heterogeneous data sources do not pose a difficulty due to semantic data access. Thus, we address the challenges C1 to C3. This section, focusses on enhancing our data quality (C4), especially under consideration of new machine and line deployments, as well as, changes in manufacturing environment.

4.1 Validating Consistency

In [16], the authors already present a definition of consistency. The four main aspects of consistency are:

1. **Completeness.** A single data set contains all expected messages. Expected messages depend on the current situation.

2. **No additional messages.** Receiving multiple messages of the same manufacturing step may indicate an error in a line.

3. **Correct content.** The content of a single message should not contain discrepancies.

4. **Without contradictions.** Messages referring to one product are consistent in their content.

As can be derived from the above definition, some consistency constraints refer to the overall data set (category 1,2, and 4), other constraints focus on a single message (category 3). The definition is based on the consideration of existing inconsistencies. Each of the inconsistencies presented below can be assigned to one of the above categories.

For the first category, the absence of an entire message can be cited. As already mentioned, the completeness of a data set is based on the respective current situation. Messages can be missing for different reasons. These can be, among other things, that a machine does not function correctly, the communication network is disrupted, or a product is prematurely removed from the manufacturing line because, for example, errors occurred during production. In the first two cases, the data set must be considered inconsistent according to the above definition, since not all expected messages are present. In the third case, the data set is consistent, as all possible messages are contained. For constraint creation, we need to make various case distinctions at this point, which quickly makes a constraint complex.

The second category deals with multiple messages of a process. At this point, it is also not possible to make a general decision as to whether a data record is consistent or inconsistent. If duplicate messages originate from the value-adding processes (SPP, SMD, RFL), they can generally be regarded as inconsistent. In the case of the inspecting processes (SPI, SJI), it must be checked whether the messages refer to the same inspection or whether a new inspection has taken place.

For the third and fourth category, we validate the content of the messages. In the former, consider the messages separately from each other. Possible inconsistencies are duplicated contents or wrong labeling of the contents. In SMD process it has happened that within the messages several components have been placed at the same position. At this point it must be ensured that the multiple placed components do not affect the real product. Also in the SMD process a wrong labeling of the placed components occurred. Unlike before, the components are correctly placed according to the data, but the area on the printed circuit board is incorrectly labeled.

When comparing the messages of a data set with each other (category 4), multiple identifiers or incorrect processing times were noticed, among other things. In the most frequent cases, the cause of these inconsistencies was software adjustments and changes in machine locations.

In [16], the authors already defined SPARQL and SHACL constraints, to automatically detect some known inconsistencies. The created constraints become complex and thus, error-prone, due to various case distinctions. Thus, we decided to enhance this manually creation process to further improve consistency checking. The following section describes our approach to automatically generate SPARQL queries from our simple definition of consistency.

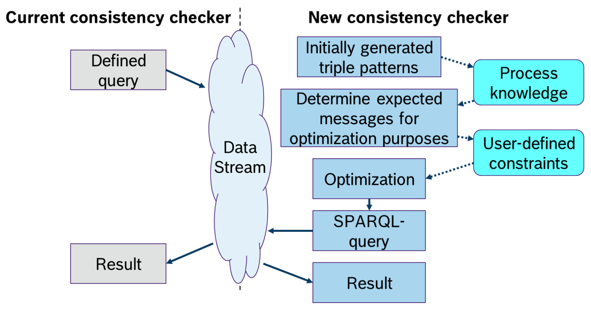

Figure 3: Current approach vs. automatic query generation. With our new approach, we are able to include additional process knowledge and user input.

4.2 Automatically Generating Constraints

With our approach of automatically generating SPARQL queries, we aim to simplify data validation. Figure 3 compares our previous consistency checking with the new approach. The left side of the figure shows the creation and execution so far: Creating the constraint is done manually. The constraint so created contains all triple patterns, to be able to query all information from the collected machine data. In addition, it includes all the case discrimination necessary to recognize the situations presented in Subsection 4.1. The long and complex query is executed on the data stream and gives us a result whether the checked data contains inconsistencies. Thus, we already fulfill task T1 and T3 and enhance the data quality (C4). Furthermore, we can execute multiple queries in parallel on stream (C1 and C2). Integrated into our existing architecture, we are thus able to meet all challenges in principle.

Problems arise when taking a closer look at the previous constraints. Since the SPARQL query has a complex structure, customizations are difficult. The query is also specifically designed for a few known inconsistencies. Thus, we do not meet the requirements R3 and R6. Furthermore, the manually created query contains many unnecessary triple patterns and FILTER operations (not fulfill T2)

The right side of Figure 3 shows this approach. In a first step, we generate an initial query, by extracting the relation triple patterns from our ontology. The generation of the query causes the error-prone structures to be created automatically. At this point, the query still contains unnecessary triple patterns. To optimize our initial query, we include user input and process knowledge. The additional process knowledge simplifies the validation of categories 1 and 2 in particular by determining which messages are to be expected and from which processes multiple messages will arrive. In

```
...

?event rdf:type smt:smdEvent .
?event rdf:type smt:rflEvent .
?event rdf:type smt:sjiEvent .
...
?processedComps smt:singleComp ?smdComp .
?smdComp smt:smdPosX ?smdPosX .
?smdComp smt:smdPosY ?smdPosY .
?smdComp smt:smdErrorCode ?errorCode .
?smdComp smt:componentMatId ?compMatID .
...
```

Listing 1: Except from initial query. The red marked lines represent unnecessary triple patterns. We can eliminate them due to incorporating knowledge.

this way, the previous complex case distinctions are eliminated. The user can define in advance via input of logical expressions for which properties a data set should be validated. The input of logical expressions requires only little knowledge about the structure of the constraint (e.g. elements from the logical expression must correspond to node names of the semantic models). Internal complexity is thus hidden from the user. Process knowledge and user input modify the initial query so that the optimized query contains only relevant triple patterns and FILTER expressions. The final step is the execution of our optimized query. As a result, we receive, as before, whether there are inconsistencies in a data set.

In the following, we exemplify our approach. In a first step, we use already existing triple pattern templates. We put these patterns together to create an initial query. The resulting query is shown in Listing 1.

Afterwards, we incorporate current process knowledge, for example to find out which messages are to be expected. In our example, we get the information that messages RFL and SJI are missing. This means, we do not have to query for RFL and SJI properties. Further, we know that an SMD message is available. Thus, we check the data set, among others, for multiple components placed on a single position. With this constraint, we check whether at most one component (e.g. resistor, port) is placed on each position on a printed circuit board. We can formalize this constraint by using a logical expression:

$$\begin{aligned}&((smdPosX1 = smdPosX2)\\&\wedge (smdPosY1 = smdPosY2))\\&\Rightarrow (smdComponent1 = smdComponent2)\end{aligned}$$

This logical expression is internally transformed into a valid SPARQL query. During the transformation process, unnecessary triple patterns are also removed

```
...
FILTER NOT EXIST {
    ?event rdf:type smt:smdEvent .
    ...
    ?processedComps smt:singleComp ?smdComp1 .
    ?smdComp1 smt:smdPosX ?smdPosX1 .
    ?smdComp1 smt:smdPosY ?smdPosY1 .

    ?event2 rdf:type smt:smdEvent .
    ...
    ?processedComps smt:singleComp ?smdComp2 .
    ?smdComp2 smt:smdPosX ?smdPosX2 .
    ?smdComp2 smt:smdPosY ?smdPosY2 .

    ...
    FILTER(?smdComp1 != ?smdComp2)
    FILTER(?smdPosX1 = ?smdPosX2)
    FILTER(?smdPosY1 = ?smdPosY2)
}
...
```

Listing 2: Except from final constraint for mulitple placement inconsistency.

from the query. In our example, these are the highlighted lines from Listing 1. The result of this transformation process is depicted in Listing 2. The red marked redundant triple patterns (see Listing 1), which for example ask for properties of the remaining processes, are removed in the transformation step.

A multistage generation, as in our approach, leads to a division of tasks into separate modules. Due to this division, it is possible to generate independent queries depending on the current state. This characteristic forms the first step to develop a state machine as described in [16]. In such a machine, a constraint is split into small lightweight queries, which are executed one after the other. As soon as an inconsistent state is reached, the state machine terminates. In the event of an error, this enables faster intervention.

In summary, it can be stated that the described way allows to check all inconsistencies known so far (R2 fulfilled). The additional user input also allows to specify various logical expressions. On the one hand, an existing query can be extended by further known inconsistencies (R6). In addition, the logical expressions can be generalized, so that a verification of previously unknown inconsistencies is possible (R3). In particular, the simplified language, as shown above, supports the user in creating complex constraints. Due to the correct automatic generation of queries from simple logical expressions, the error rate during the creation process is reduced (R7). The fulfillment of requirement R7 has a decisive impact on challenge C5. Since the overall architecture is designed for parallel operation, it is still possible to execute the presented steps in parallel and to execute multiple generated queries in parallel (R5). The following experimental results will provide information about the overhead of query generation and show whether requirement R4 is fulfilled. In any case, the query creation process is simplified, which leads to the fulfillment of R1.

5 Experimental Results

In our first experimental results, we compare the automatically generated queries with our manually created ones. Doing this, we aim to determine the overhead which results from the generation.

We perform our evaluation on a local computer with 16 GB main memory and 11th generation Intel i5 processor. The program code for the generation as well as the execution of the queries is written in Python. We use the python package *rdflib*[3] for both the manually created and the generated queries. This provides comparability between the approaches. For resource monitoring we use *tracemalloc*[4] . For the manually created query, we use a slightly modified query than the one described in [16].

Table 1 illustrates our results. We compare the runtime and memory overhead of the newly generated queries against the manually created ones. To illustrate the overhead, we choose four different situations:

(1) No inconsistency: In this scenario, all messages are available. There is no inconsistency in the format of our definition (see beginning of Section 4.1).

(2) Missing message, no inconsistency: In our data set for this scenario, messages for RFL and SJI processes are missing. As discussed in Subsection 4.1, not every missing message leads to an inconsistent data set.

(3) Inconsistency in single message: This scenario includes an inconsistency in an SMD message. We exemplify the overhead for our multiple placement check (see example in Section 4.2).

(4) Inconsistency due to missing message: In this scenario, an SMD message is missing. our data set contains messages from the remaining processes, the data set is inconsistent.

For the evaluation, we use pre-collected datasets that have the required properties of our scenarios. At this point, we do not test the stream capability and overhead of our overall architecture. The resulting overhead

Table 1: Comparison of constructed against generated constraints. The values in the brackets represent the overhead for manually creating or automatically generating the queries.

Case	Runtime	Memory
(1) No inconsistency		
- *Constructed*	0.57 s (1 h)	9.93 MB
- *Generated*	0.55 s (0.75 s)	9.64 MB (12.48 MB)
(2) Missing message no inconsistency		
- *Constructed*	0.51 s	9.92 MB
- *Generated*	0.49 s (0.84 s)	9.52 MB (12.49 MB)
(3) Inconsistency in single message		
- *Constructed*	0.92 s	9.94 MB
- *Generated*	0.87 s (0.76 s)	9.64 MB (12.47 MB)
(4) Inconsistency due to missing message		
- *Constructed*	0.53 s	9.92 MB
- *Generated*	0.42 s (0.92 s)	9.29 MB (12.49 MB)

is identical in both applications and can therefore be neglected for our evaluation.

Table 1 contains the mean values of 20 measurements for each of the four scenarios. As can be seen in the table, runtime and memory overhead for the execution of the constructed query are very close to each other in the four scenarios. Individual runtime differences can for example be explained by different case executions. The time required for the manual creation of our constraint is very high compared to the creation effort of the generated queries. Since we use the same query for all scenarios, this effort is only required at the beginning or after changes. In particular, the time includes finding and arranging the required triple patterns.

For the generated queries, we consider separately the overhead of execution and that of generation (shown in the brackets). As the table shows, the memory overhead of execution is reduced compared to the manually generated query. Only very small differences can be observed between the scenarios. However, in the case of the generated queries, the optimization overhead is added. Since the optimizations must be accomplished however only once at the beginning and then in each case with changes of the constraints, the memory overhead is accepted. The same applies to the run times of the optimization steps. Regarding the overhead of the optimization steps, it can be stated that the more steps

[3] https://pypi.org/project/rdflib/

[4] https://docs.python.org/3/library/tracemalloc.html

Table 2: Number of triple patterns and expressions in manually constructed and automatically generated SPARQL query.

Case	Triples	Expressions
(1) No inconsistency		
- *Constructed*	74	61
- *Generated*	67	48
(2) Missing message no inconsistency		
- *Constructed*	74	61
- *Generated*	47	41
(3) Inconsistency in single message		
- *Constructed*	74	61
- *Generated*	67	48
(4) Inconsistency due to missing message		
- *Constructed*	74	61
- *Generated*	21	32

have to be performed, the higher it is.

When looking at the execution times, it is noticeable that the executions of the optimized queries are lower than those of the manually created queries. Due to the optimizations within the queries, some case distinctions are already omitted in scenarios one and three. For scenarios two and four, specific checks and the triple patterns of missing messages are omitted. In all four scenarios, the optimization steps make the queries leaner and thus faster to execute.

Table 2 gives an overview of the size and thus the complexity of our created and generated SPARQL queries. Column *expressions* shows FILTER, BIND, and UNION expressions, as well as case distinctions (IF-ELSE) in sum. The expressions for the manually created queries are divided into 16 FILTER, 14 BIND, 18 UNION, and 7 IF-ELSE. Since this created query remains unchanged for all four scenarios, the number of triple patterns and expressions do not change either.

For the generated queries, we can generally observe that the number of triple patterns and SPARQL expressions is reduced. Since we know which messages to expect, all case distinctions can be eliminated. Especially in scenarios two and four, large differences to the manually generated query are visible. The reason for this is that there are no RFL and SJI messages in scenario two. For this reason, both the specific checks for these messages and the corresponding triple pattern in the remaining checks are omitted. In scenario four, many specific checks are omitted due to the missing SMD message. This massively reduces the number of triple patterns and the required SPARQL expressions, which results in a shorter query runtime (see Table 1).

6 DISCUSSION

There are two primary points we want to discuss in this section. These are: (1) adaptability of the approach and (2) overhead of the application.

In our new approach, the user has the possibility to specify constraints via logical expressions. The logical expressions result from the definition of a property to be checked. The approach thus no longer only allows to check known inconsistencies, but to create a definition of a consistent data set via logical expressions. Our application enables to internally generate valid SPARQL queries from user input. This generation eliminates the error-prone part of triple pattern construction. The simplified process can help to identify errors in current production as well as new lines, thus improving data quality and processes (C4 and C5). The used ontologies and query templates are interchangeable, so that the approach can be applied in diverse domains. With regard to a heterogeneous production landscape, as it exists at Bosch, this property is of high importance and impact.

Currently, however, our generation approach is limited to a restricted domain vocabulary, as the expressions must contain the identifiers from the used ontology. Logical expressions that do not fulfill the naming conventions are ignored so far. In order to give the user more freedom and to reduce the required prior knowledge, an NLP-based approach is conceivable at this point. Evaluations must show how well such an approach performs in our manufacturing scenario.

However, the current experimental results have shown that the resource consumption of our newly developed approach is lower than when running our manually constructed query. This is due to the elimination of case distinctions or entire constraints through integrating additional process knowledge. In addition, there is the effort to generate the optimized queries. Our evaluaion has shown that these steps are cost intensive with regard to runtime and memory usage. However, these optimization steps need to be done only once after constraints have been changed. Afterwards the generated queries can be used directly without having any additional costs of optimization. By defining consistency, we assume that in the future our approach will be able to detect not only already known but also unknown inconsistencies in the data. Thus, changes to the constraints are usually not needed frequently. If changes are incorporated, the runtimes pay off after ten to 40 checks, depending on the scenario. Taking into account that the queries can in principle be used in all plants worldwide, we achieve resource savings after only a few minutes of running production. The generated queries themselves can be further optimized by rewriting SPARQL queries with general optimization transformation rules as e.g. given in [6]. For our application scenario, however, the reduced complexity of the constraint creation process is of primary importance. In this way, a separation of responsibility takes place depending on the expert. Domain experts can thus concentrate completely on the definition of consistent states. Deeper knowledge of SPARQL and other languages is not necessary.

One aspect we need to address in future work is to further optimize our approach to reduce runtime and memory overhead. To reduce the runtime, a state machine is a good choice. In such a machine, a new state is reached based on incoming messages. The state machine is then able to determine for each state whether there is still consistency or whether a consistent state can be reached at all. With this ability, the machine offers the possibility to detect inconsistencies at an early stage so that appropriate interventions can be initiated. Hardware-accelerating [24] [8] [7] [14] the state machine might further help to meet latency constraints. One disadvantage of the state machine is that the memory overhead is expected to increase. Our future task is now to optimize our queries, and in particular our query generation algorithm, to the point where we get a good tradeoff between latency and memory overhead.

7 Conclusion

This paper presents an approach for the automatic generation of SPARQL constraints. With the help of these constraints, data streams can be validated and data quality can be improved. As our experimental results show, the pure executions of our generated queries have lower runtime and memory overhead, compared to our manually created queries. The effort to generate the queries occurs once after adjusting the constraints. Considering our goal of simplifying constraint creation, this overhead is not significant.

In summary, our approach already improves consistency checking by allowing domain experts with less semantic know-how to validate data (streams). However, further simplifications are necessary so that optimal SPARQL queries can be generated from a simple and general definition of consistency. As already discussed in the previous section, NLP based approaches may lead to an improvement. The focus of our future work is on the one hand this simplification. On the other hand, we would like to investigate the aforementioned creation of a state machine in more detail. Such a machine would allow earlier interventions in case of errors.

Acknowledgements

The work of this paper has been supported by AE/MFT1 department of Robert Bosch Elektronik GmbH.

References

[1] K. Baclawski, M. M. Kokar, R. Waldinger, and P. A. Kogut, "Consistency checking of semantic web ontologies," in *International Semantic Web Conference*. Springer, 2002, pp. 454–459.

[2] Y. Chen, M. M. Kokar, and J. J. Moskal, "Sparql query generator (sqg)," *Journal on Data Semantics*, vol. 10, no. 3, pp. 291–307, 2021.

[3] J. Corman, F. Florenzano, J. L. Reutter, and O. Savković, "Validating shacl constraints over a sparql endpoint," in *International Semantic Web Conference*. Springer, 2019, pp. 145–163.

[4] R. Cortés, X. Bonnaire, O. Marin, and P. Sens, "Stream processing of healthcare sensor data: studying user traces to identify challenges from a big data perspective," *Procedia Computer Science*, vol. 52, pp. 1004–1009, 2015.

[5] S. Gao, D. Dell'Aglio, J. Z. Pan, and A. Bernstein, "Distributed stream consistency checking," in *International Conference on Web Engineering*. Springer, 2018, pp. 387–403.

[6] J. Groppe, S. Groppe, and J. Kolbaum, "Optimization of sparql by using coresparql." in *ICEIS (1)*, 2009, pp. 107–112.

[7] S. Groppe, "Semantic hybrid multi-model multi-platform (shm3p) databases," in *Proceedings of the International Semantic Intelligence Conference (ISIC), New Delhi, India*, 2021.

[8] S. Groppe and J. Groppe, "Hybrid multi-model multi-platform (hm3p) databases." in *DATA*, 2020, pp. 177–184.

[9] S. Groppe, J. Groppe, D. Kukulenz, and V. Linnemann, "A SPARQL engine for streaming RDF data," in *Third International IEEE Conference on Signal-Image Technologies and Internet-Based System (SITIS), Shanghai, China*, 2007, this paper received an honorable mention at the SITIS'07 Conference. [Online]. Available: https://doi.org/10.1109/SITIS.2007.22

[10] D. Guo, E. Onstein, and A. D. L. Rosa, "An approach of automatic sparql generation for bim data extraction," *Applied Sciences*, vol. 10, no. 24, p. 8794, 2020.

[11] H.-M. Haav, R. Maigre, A. Lupeikiene, O. Vasilecas, and G. Dzemyda, "A semantic model for product configuration in timber industry," in *Databases and Information Systems X*. IOS Press, 2019, vol. 315, pp. 143–158.

[12] H. Jung and W. Kim, "Automated conversion from natural language query to sparql query," *Journal of Intelligent Information Systems*, vol. 55, no. 3, pp. 501–520, 2020.

[13] M. H. Kashani, M. Madanipour, M. Nikravan, P. Asghari, and E. Mahdipour, "A systematic review of iot in healthcare: Applications, techniques, and trends," *Journal of Network and Computer Applications*, vol. 192, p. 103164, 2021.

[14] M. Koppehel, T. Groth, S. Groppe, and T. Pionteck, "Cuart-a cuda-based, scalable radix-tree lookup and update engine," in *50th International Conference on Parallel Processing*, 2021, pp. 1–10.

[15] T.-Y. Ku, W.-K. Park, and H. Choi, "Iot energy management platform for microgrid," in *2017 IEEE 7th International Conference on Power and Energy Systems (ICPES)*. IEEE, 2017, pp. 106–110.

[16] S. Paasche and S. Groppe, "Enhancing data quality and process optimization for smart manufacturing lines in industry 4.0 scenarios," in *Proceedings of The International Workshop on Big Data in Emergent Distributed Environments*, 2022, pp. 1–7.

[17] M. Sander, U. Waltinger, M. Roshchin, and T. Runkler, "Ontology-based translation of natural language queries to sparql," in *2014 AAAI fall symposium series*, 2014.

[18] G. Schuh, C. Thomas, A. Hauptvogel, and F. Brambring, "Achieving higher scheduling accuracy in production control by implementing integrity rules for production feedback data," *Procedia CIRP*, vol. 19, pp. 142–147, 2014.

[19] S. Steyskal and M. Wimmer, "Leveraging semantic web technologies for consistency management in multi-viewpoint systems engineering," in *Semantic Web Technologies for Intelligent Engineering Applications*. Springer, 2016, pp. 327–352.

[20] F. Tao, Q. Qi, A. Liu, and A. Kusiak, "Data-driven smart manufacturing," *Journal of Manufacturing Systems*, vol. 48, pp. 157–169, 2018.

[21] Y. Tian, P. Michiardi, and M. Vukolić, "Bleach: A distributed stream data cleaning system," in *2017 IEEE International Congress on Big Data (BigData Congress)*. IEEE, 2017, pp. 113–120.

[22] B. Warnke, J. Mantler, S. Groppe, Y. C. Sehgelmeble, and S. Fischer, "A sparql benchmark for distributed databases in iot environments," in *Big Data in Emergent Distributed Environments (BiDEDE'22), Philadelphia, PA, USA*, 2022.

[23] B. Warnke, Y. C. Sehgelmeble, J. Mantler, S. Groppe, and S. Fischer, "Simora: Simulating open routing protocols for application interoperability on edge devices," in *6th IEEE International Conference on Fog and Edge Computing (ICFEC22), Taormina (Messina), Italy*, 2022.

[24] S. Werner, D. Heinrich, M. Stelzner, V. Linnemann, T. Pionteck, and S. Groppe, "Accelerated join evaluation in semantic web databases by using fpgas," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 7, pp. 2031–2051, 2016.

[25] S. Xuanyuan, Y. Li, L. Patil, and Z. Jiang, "Configuration semantics representation: A rule-based ontology for product configuration," in *2016 SAI Computing Conference (SAI)*. IEEE, 2016, pp. 734–741.

[26] H. Yang, S. Kumara, S. T. Bukkapatnam, and F. Tsung, "The internet of things for smart manufacturing: A review," *IISE Transactions*, vol. 51, no. 11, pp. 1190–1216, 2019.

[27] M. Zou, B. Lu, and B. Vogel-Heuser, "Resolving inconsistencies optimally in the model-based development of production systems," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2018, pp. 1064–1070.

AUTHOR BIOGRAPHIES

Simon Paasche is a PhD candidate in the Automotive Electronics department at Robert Bosch Elektronik GmbH in Salzgitter. His research focus is on the use and further development of semantic methods for stream-based data validation and process optimizations in industry 4.0 scenarios. The work in this area is based on real manufacturing lines in Bosch environment. The PhD research is supervised by Prof. Sven Groppe from University of Lübeck.

Sven Groppe is Professor at the University of Luübeck, Germany. He was a member of the DAWG W3C Working Group, which developed SPARQL. He was the project leader of the DFG project LUPOSDATE and two research projects on FPGA acceleration of relational and Semantic Web databases, and is a member of the Hardware Accelerator Research Program by Intel. He is currently the project leader of German Research Foundation projects on GPU accelerated database indices and on Semantic Internet of Things. Furthermore, he is leading a project about quantum computer accelerated database optimizations and he is project partner in a project about COVID-19 high-quality knowledge graphs, visualizations and analysis of the pandemic with 2 french partners. His research interests include Internet of Things, Semantic Web, query and rule processing and optimization, Big Data, Cloud Computing, peer-to-peer (P2P) networks, hardware acceleration (FPGA, GPU, QPU), quantum computation, data visualization and analysis, and visual query languages. He is the workshop organizer and chair of the Semantic Big Data workshop series (2016 to 2020) in conjunction with ACM SIGMOD. In 2021 and 2022 he organizes the International Workshop on Big Data in Emergent Distributed Environments (BiDEDE) @ SIGMOD and the International Workshop of Internet-of-Things (VLIoT) in conjunction with VLDB since 2017. He is the general chair of the International Semantic Intelligence Conferences in 2021 and 2022, and of the International Healthcare Informatics Conference (IHIC) in 2022.