Distributed Situation Recognition in Industry 4.0

Mathias Mormul, Pascal Hirmer, Matthias Wieland, and Bernhard Mitschang

Institute of Parallel and Distributed Systems
University of Stuttgart, Universitätsstr. 38, D-70569, Germany
email: firstname.lastname@ipvs.uni-stuttgart.de

***Abstract*—In recent years, advances in the Internet of Things led to new approaches and applications, for example, in the domains Smart Factories or Smart Cities. However, with the advantages such applications bring, also new challenges arise. One of these challenges is the recognition of situations, e.g., machine failures in Smart Factories. Especially in the domain of industrial manufacturing, several requirements have to be met in order to deliver a reliable and efficient situation recognition. One of these requirements is distribution in order to achieve high efficiency. In this article, we present a layered modeling approach to enable distributed situation recognition. These layers include the modeling, the deployment, and the execution of the situation recognition. Furthermore, we enable tool support to decrease the complexity for domain users.**

Keywords–Industry 4.0; Edge Computing; Situation Recognition; Distribution Pattern.

I. Introduction

This article is a revised and extended version of the SMART 2018 paper "Layered Modeling Approach for Distributed Situation Recognition in Smart Environments" [1]. The emerging paradigm Industry 4.0 (I4.0), describing the digitization of the manufacturing industry, leads to the realization of so-called Smart Factories [2]. In I4.0 and Internet of Things in general, devices equipped with sensors and actuators communicate with each other through uniform network addressing schemes to reach common goals [3][4]. One of these goals is situation recognition, which enables monitoring of I4.0 environments and, consequently, the timely reaction to occurring situations. For example, the occurrence of a machine failure in a Smart Factory, recognized by sensors of the machine, could lead to an automated notification of maintenance engineers.

Situations are recognized using context data that is usually provided by sensor measurements. In current approaches, such as the one we introduced in our previous work [5][6], situations are recognized in a monolithic IT infrastructure in the cloud. Consequently, involved context data needs to be shipped to the processing infrastructure in order to recognize situations. However, especially in domains where efficiency is of vital importance, e.g., Smart Factories, this approach is not feasible. In order to fulfill important requirements, such as low network latency and fast response times, the situation recognition needs to be conducted as close to the context data sources as possible and, therefore, in a distributed manner. Processing data close to the sources is commonly known as Edge Computing [7].

In this paper, we introduce an approach to enable a distributed situation recognition. By doing so, we introduce so-called *distribution patterns*. These patterns represent common ways to distribute the recognition of situations, i.e., exclusively in the *edge*, in on-premise or off-premise cloud infrastructures, or based on a hybrid approach. We provide a layered approach for modeling and executing the situation recognition based on these distribution patterns. Our approach builds on a set of requirements we derive from a use case scenario in the manufacturing domain. We validate the approach by applying it to our previous non-distributed situation recognition [5][6] that is based on the modeling and execution of so-called Situation Templates [8]. Furthermore, we introduce a modeling tool for Situation Templates as well as an automated distribution of the templates in the edge and backend cloud.

This article is a revised and extended version of the SMART 2018 paper "Layered Modeling Approach for Distributed Situation Recognition in Smart Environments" [1]. In addition to the previous paper, we describe how distributed situation recognition can be realized from the modeling of the situation using Situation Templates to the actual deployment. This is done by the introduced tool-based modeling support and the automated distribution of Situation Templates among the edge and backend cloud.

The remainder of this paper is structured as follows: Section II describes related work and foundational background. In Section III, we introduce a motivating scenario, which is used to derive requirements for our approach. In Section IV, we present the main contribution of our paper. Section V describes the process from modeling Situation Templates using a tool-based modeling support to the automated distribution of the situation recognition. Finally, Section VI concludes the paper and gives an outlook to future work.

II. Related Work and Background

In this section, we describe related work, as well as foundational concepts of our previous work that are necessary to comprehend our approach.

A. Related Work

In related work, approaches exist for distributed situation recognition using ontologies, e.g., by Fang et al. [9]. These approaches do not achieve the latency required in real-time critical scenarios, such as Industry 4.0 [2], due to time-consuming reasoning. The goal of our approach is to achieve low latency for distributed situation recognition in the range of milliseconds. Many approaches using ontologies are in the range of seconds to minutes, even without distribution [10], [11]. Using machine learning leads to similar limitations regarding latency [12].

In the area of distributed Complex Event Processing (CEP), Schilling et al. [13] aim at integrating different CEP systems

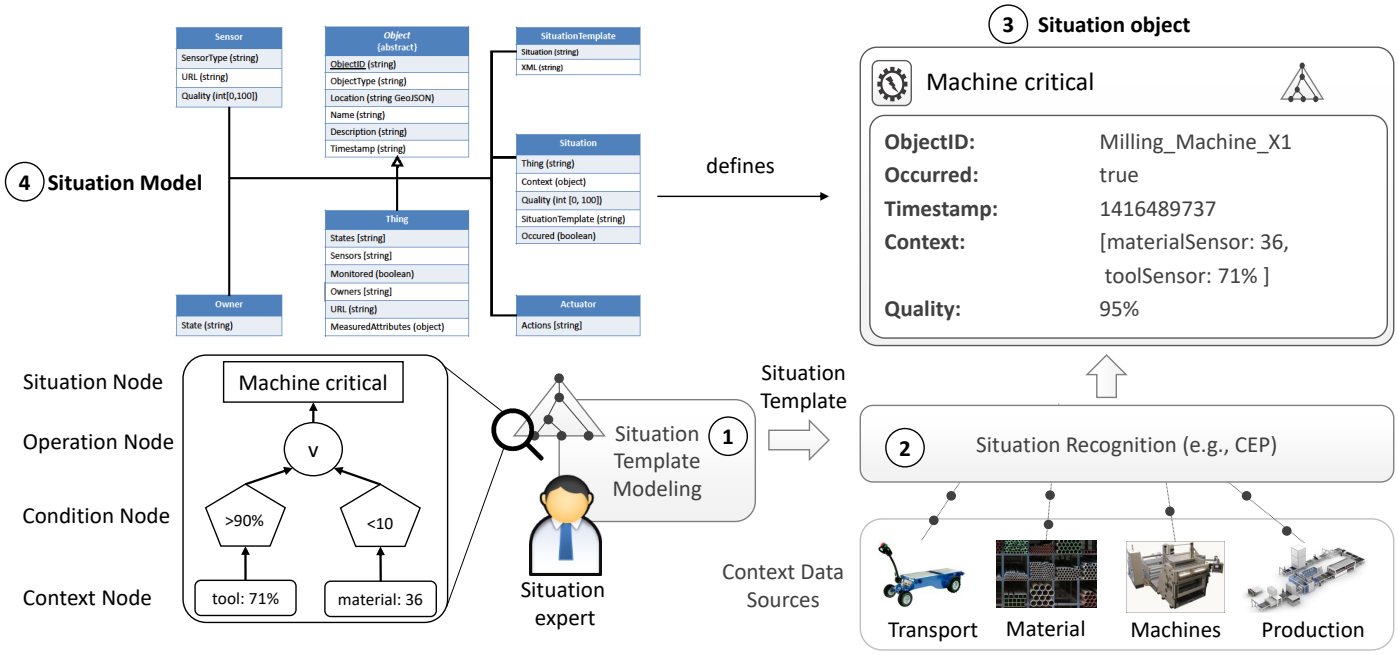

Figure 1. Previous approach for situation recognition [1]

using a common meta language. This allows to use different CEP systems and integrate the results. This could be beneficial for our distribution because we would not be limited to one execution environment. However, in [13], the queries have to be hand-written and distributed. This is difficult, especially for domain experts, e.g., in Industry 4.0, who do not have the necessary skillset. In our approach, we provide an abstraction by Situation Templates that can be modeled using a graphical user interface. Furthermore, the users are supported in splitting up these template as well as in the distribution decision.

Other approaches in distributed CEP, e.g., by Schultz-Moller et al. [14], follow the concept of automatic query rewriting. Here, CEP queries are split up using automated rewriting and are distributed on different operators based on a cost model, which is mostly based on CPU usage in the different nodes. In our approach, we want to support the user to select the desired distribution type. Since there are many aspects, such as data protection or security, that play a role in distributing the CEP queries correctly, this only can be known by a responsible expert user.

Finally, approaches exist that enable a massive distribution of sensors, e.g., by Laerhoven and Gellersen [15] in cloths, to detect activities of the person wearing the cloth. This is similar to detecting the situation in the edge cloud, but there is no concept presented in [15] to integrate the activities with other activities from different edge clouds or create a global situation involving different locations.

B. Background

In this section, we describe our previous work. Our first approach for situation recognition, this paper builds on, is depicted in Figure 1. This approach is a result of the issues of related work, as discussed in the previous section.

An important fundamental concept are Situation Templates, introduced by Häussermann et al. [8]. We adapted the Situation Templates in [6] to model and recognize situations. Situation Templates (see Figure 1 on the buttom left) consist of *context, condition* and *operation* nodes, which are used to model specific situations. Context nodes describe the input for the situation recognition, i.e., the context data, based on the definition of Dey et al. [16]. Context nodes are connected to condition nodes, which define the conditions for a situation to be valid. Operation nodes combine condition and operation nodes and represent the logical operators *AND, OR*, or *XOR*. Operation nodes are used to aggregate all condition nodes of the Situation Template into a single node, the situation node.

After modeling a Situation Template (Figure 1, Step 1), it is transformed into an executable representation (not depicted), which is realized using CEP or light-weight execution languages, such as Node-RED. The advantage of this transformation is that it provides a flexible means to recognize situations. These transformations can be found in [17][18]. Consequently, we are not limited to specific engines or data formats. Once the transformation is done, the executable Situation Template is handed over to the corresponding execution engine.

On execution (Figure 1, Step 2), context data originating from the context sources is validated against the conditions defined by the Situation Template, for example, through pattern recognition in CEP. On each validation, we create a so-called situation object [19], defining whether the situation occurred and containing the involved context data (Figure 1, Step 3). We created a Situation Model [19] (Figure 1, Step 4) to define the attributes of those situation objects. This leads to a better understanding of how context data led to the situation.

This previous approach for situation recognition works well, however, there are still some limitations this paper aims to solve. First, the current approach was built to monitor single things (e.g., devices). However, as the complexity of nowadays IT infrastructure rises, means need to be enabled to monitor more than one thing using the introduced Situation Templates.

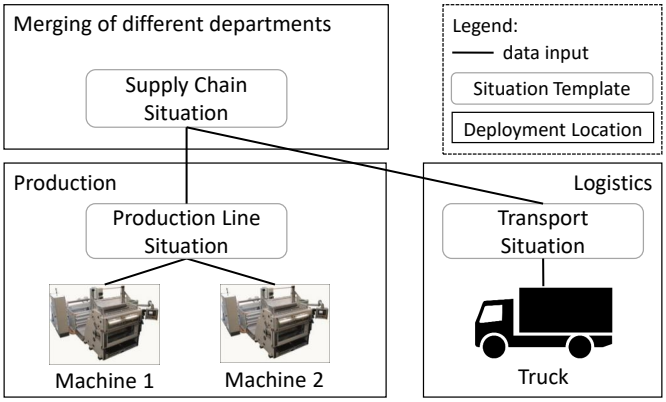

Figure 2. Motivating scenario for distributed situation recognition [1]

Furthermore, currently, the Situation Templates are executed in a monolithic manner because in former scenarios, distribution was not necessary. In current approaches, e.g., involving I4.0, however, this is necessary. Therefore, in this paper, we aim for enhancing our approach in order to be more fitting to recent scenarios.

III. Scenario and Requirements

In this section, we introduce a practical motivating scenario from the I4.0 domain, which is used throughout the paper to explain our approach. In the scenario, depicted in Figure 2, a specific part of the supply chain of a production company should be monitored. As depicted, there are several entities involved: (i) production machines, assembling products based on parts, and (ii) trucks, delivering the parts to be assembled. The monitoring should detect critical situations that could occur, for example, the failure of at least one of the machines, or a delivery delay of parts, e.g., caused by issues with trucks or with the supplier. Situations that could occur are: (i) *Production Line Situation*, indicating that one of the production machines is in an erroneous state, (ii) *Transport Situation*, indicating a problem with the truck, and (iii) *Supply Chain Situation*, indicating a problem with either the production line or the truck.

When applying our previous approach, described in Section II, to this scenario, new requirements arise that need to be coped with. We divide these requirements into ones that concern the modeling of Situation Templates and ones that concern the execution of the situation recognition. We derived eight requirements R_1 to R_8 for this scenario.

Modeling Requirements

Three requirements focus on the modeling of the situation recognition using Situation Templates.

- R_1 **- More powerful Situation Templates:** With our previous approach (cf. Section II-B), single machines can be monitored in an efficient way as evaluated in [6], which was sufficient for previous scenarios. However, in recent scenarios involving Industry 4.0, the requirements are increasing. In our motivating scenario, it is important to model dependencies between multiple entities within a single Situation Template, e.g., to recognize the *Production Line Situation*.
- R_2 **- Low modeling complexity:** In our previous approach, modeling Situation Templates involving a lot of context data has led to a cumbersome task and, consequently, to a high complexity and error-prone modeling. To cope with this issue, a new modeling approach is required that enables the reutilization of already existing Situation Templates to lower the modeling complexity of new Situation Templates.
- R_3 **- Domain-independence:** A consequence of the issue described in R_2 is domain-dependence. Large Situation Templates usually consist of a wide range of context data sources, e.g., Trucks or the Production Line of our scenario. However, these context data sources require domain experts of these specific areas. Consequently, Situation Templates need to be modeled by these experts together. This leads to high costs due to the communication overhead. Hence, our goal is to enable domain-independence for Situation Template modeling.

Execution Requirements

- R_4 **- Low latency:** In many domains, latency plays a crucial role. Especially in Smart Factory environments, the industrial automation layer has strong requirements regarding end-to-end latency up to 1 ms or even lower [20]. Therefore, the execution of the situation recognition needs to adapt to those requirements, so that critical situations like machine failures can be recognized in a timely manner.
- R_5 **- Low network traffic:** In modern scenarios, large amounts of data are produced that need to be stored and processed in order to recognize situations. For example, an autonomous car produces about 35 GB/hour of data [21]. In comparison, Budomo et al. [22] conducted a drive test and recorded a maximum and minimum upload speed of 30Mbps (13.5 GB/hour) and 3.5Mbps (1.58 GB/hour), respectively, using the current mobile communication standard LTE-A. Therefore, transferring all data of an autonomous car to the cloud is currently impossible. Consequently, reducing the network traffic is an important issue when recognizing situations.
- R_6 **- Reduced costs:** Costs are always an essential factor when it comes to data processing. Operating and maintaining an in-house IT infrastructure could lead to high costs. In contrast, using the pay-as-you-go approach of Cloud Computing, costs could be reduced. Enabling the lowest possible costs when recognizing situations is an important requirement for this paper.
- R_7 **- Data security & privacy:** Especially the processing of company data needs to be secure and, furthermore, privacy needs to be ensured. However, especially when processing data in the Public Cloud, companies need to trust the Cloud providers that they provide the security they require. Alternatively, companies can keep their data close, i.e., in a trusted environment. Additionally, in many countries and federations, data protection directives are in place to guarantee the protection and privacy of personal data that may not allow to send personal data to the cloud [23].

- R_8 **- Cross-company situation recognition:** Modern products and their components are rarely built completely by one company. Therefore, most actual scenarios are very complex, involve multiple companies, and require a cross-company situation recognition. Our motivating scenario in Figure 2 can be regarded as such an example, in which a manufacturing company cooperates with a logistics company. For example, a delayed delivery caused by a failure of the truck must be communicated to the manufacturing company. Consequently, our situation recognition approach needs to enable a cross-company situation recognition.

IV. Distribution of Situation Recognition

In our previous work, we already solved challenges regarding sensor registration and management [24], efficient solutions for a situation recognition [18], and the management of recognized situation [19]. Now, we concentrate on extending our previous approach by introducing a distribution of the situation recognition to fulfill the above-mentioned requirements R_1-R_8. For this, we first present (i) the modeling improvements for our approach to support the distribution we aim for. On this basis, we present (ii) the execution improvements to enable the distribution based on three distribution patterns including a decision support for each of those patterns.

The distributed situation recognition was implemented based on the existing prototype of our previous work, introduced in [18][19] by following adaptations: (i) the modeling for Situation Templates was extended, (ii) the transformation was enhanced to accept multiple things, and (iii) the communication between the distributed locations is enabled by messaging systems.

A. Modeling Improvements

In the following, we present the improvements regarding the modeling of Situation Templates to fulfill the requirements R_1-R_3. The extension of the Situation Templates, i.e. its schema, comprises (i) the modeling of multiple things within a single Situation Template, and (ii) a layered modeling by reutilizing already modeled Situation Templates. These extensions are depicted in Figure 3. Requirement R_1 describes the need for the modeling of more powerful situations, e.g., *Production Line critical*. However, a production line itself does not contain any sensors but rather describes the coherence and arrangement of multiple machines. Therefore, to model a situation describing the production line, we need to model all machines of the production line into a single Situation Template. By extending the Situation Template Schema to allow the modeling of multiple things, therefore, we fulfill requirement R_1.

It is obvious that from a certain amount of things in a single Situation Template and each thing having multiple sensors, the complexity of modeling such a Situation Template is becoming a problem. An excessive complexity restricts the usability of our modeling approach, hence, the reduction of the modeling complexity is required (cf. R_2). To cope with the increasing complexity of Situation Templates, we introduce the layered modeling approach. Instead of modeling everything within a single Situation Template, we use situations as context input for further situation recognition. Thereby, we implicitly reuse already modeled Situation Templates. Furthermore, we divide situations in two classes: *local situations* and *global situations* whereby local situations are recognized at the edge and global situations in the cloud. An equivalent modeling of the situation *Production Line critical* using the layered modeling approach is shown in Figure 3 (right side). Based on this comparison, we show the benefits of this approach:

- **Reusability:** By using situations as input, we reutilize existing Situation Templates. When modeling a global situation, we only need to model the relation between the already modeled local situations similar to putting together building blocks. A further advantage is that the local Situation Templates possibly were already used and tested for correctness, which lowers the error-proneness for modeling global situations.
- **Reduce complexity:** The reusability directly leads to less complex Situation Templates, since the modeling is based on the Divide and Conquer paradigm. By using the layered modeling approach, we fulfill the requirement R_2.
- **Distribution:** Since we do not have one single and complex Situation Template, but instead, multiple smaller ones, we already have a beneficial starting point for the distribution of the situation recognition as we can simply execute the different Situation Templates at different locations.
- **Support for specific domains:** Having multiple things within a single Situation Template could lead to the problem that knowledge from different domains is required. For example, our motivating scenario contains three domains - manufacturing, logistics, and their dependencies. Using the layered modeling approach, different domains can model Situation Templates independently. Consequently, the requirement R_3 is fulfilled as well.

As a result, by introducing an extended Situation Template Schema to enable the modeling of multiple things within a single Situation Template and the layered modeling approach, we fulfill all modeling requirements R_1-R_3.

B. Execution Improvements

The modeling improvements we presented in the last section serve as the foundation for the distribution of the situation recognition. As mentioned above, in our previous approach, the situation recognition was executed centralized in the cloud. Hence, all context data was sent to this cloud and was used as input for the situation recognition. However, lately, the term *Edge Computing* gains more and more attention. Shi et al. [7] define *the edge* as *"any computing and network resources along the path between data sources and cloud data centers"*. Therefore, in our context, Edge Computing refers to the processing of context data close to the data sources.

By introducing Edge Computing to our approach, a distribution of the situation recognition to the cloud and the edge can be performed. In the scenario of Figure 2, the distribution of the situation recognition seems obvious. Using the layered modeling approach, we can model the local situations *Production Line Situation* and *Transport Situation* and the global situation *Supply Chain Situation*. The situation recognition for the local situation is executed at the edge, i.e., locally in the factory or truck, respectively. The global situation is executed

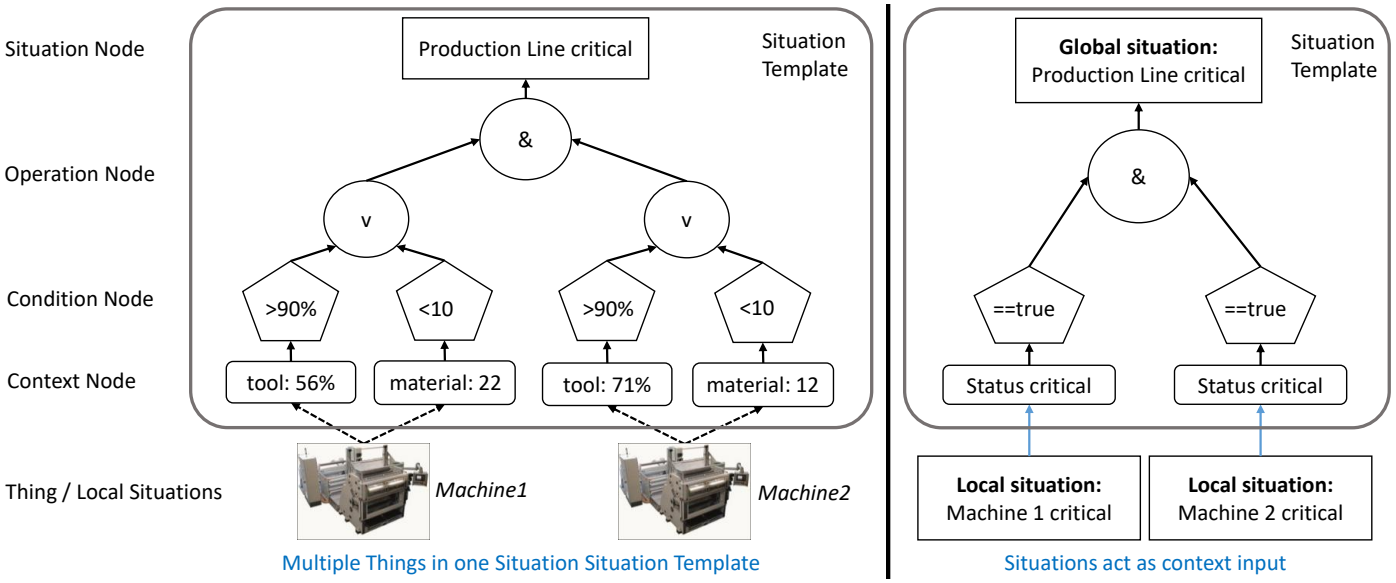

Figure 3. Modeling improvements for Situation Templates (legend see Figure 4) [1]

in the cloud and receives the local situations as input. However, based on the execution requirements R_4-R_8, this distribution might not always be ideal.

Therefore, in the following, we present the execution improvements resulting from the distribution of the situation recognition. First, we present the concept of context stripping and its benefits. Afterwards, we introduce three distribution patterns and a decision support for choosing the most suitable distribution pattern for a certain scenario.

1) Context Stripping: As presented in Section II-B, when a situation recognition is executed, situation objects are created that are defined by the Situation Model [19]. This situation object contains all context data that were used for the evaluation of this specific situation. In [19], we approximated the data volume of situation objects based on the amount of used context data. The results showed that the appended context data presents the majority of the data size of a situation object. Now, when using the layered modeling approach, we may use local situations that we recognized at the edge as input for the recognition of global situations in the cloud. That causes us to send all context data to the cloud again within the situation object. However, based on the scenario, we might not be interested in the context data of a situation object but only if the local situation occurred or not, so we can evaluate the global situation. Therefore, we introduce the concept of *context stripping*. By using context stripping, the context used for the situation recognition is not sent within the situation object. It only contains the most vital data for a further situation recognition in the cloud. Therefore, content-wise, a local situation only contains a boolean value, which describes if the local situation occurred or not and the required meta data for further processing.

This leads to a trade-off the user has to make based on his requirements. By using context stripping, the data size of a situation object can be strongly reduced. However, the context data that led to the evaluation of a specific situation object is discarded after processing. In our first approach, we explicitly wanted to store the context data within situation objects for a detailed historization of situations. This historization, for example, can be used afterwards for a root cause analysis of detected situations based on the involved context data. We are planning to conduct a performance evaluation regarding the degree of context stripping to be used in specific scenarios.

2) Distribution Patterns: As mentioned above, the distribution of the situation recognition is dependent on the execution requirements R_4-R_8. Therefore, a general solution for the distribution of the situation recognition is not possible. Instead, we introduce three different distribution patterns, depicted in Figure 4 based on the scenario shown in Figure 2. The *Type I* distribution pattern describes our previous approach. All context data, i.e., in this scenario, context data from a truck and two machines, are sent to the cloud. The situation recognition is executed in the cloud and all context data is available. In contrast, the *Type II* pattern describes the execution of the situation recognition at the edge, close to the data sources. In this case, it is often impossible to gather all context data from all sources, e.g., from the truck, since it is not part of the local network of the factory, where the machines are located. Therefore, only parts of the situation recognition may be executed at the edge. The *Type III* pattern is a hybrid solution based on both the *Type I* and *Type II* pattern and enables the execution of situation recognition at the edge, which results in local situations (i.e., *Production Line* and *Transport*) and the execution of situation recognition in the cloud, where the local situations are used to evaluate the global situation.

In the following, the different distribution patterns are described in more detail with regard to the execution requirements R_4-R_8. Each pattern comprises advantages for certain use cases and might not fulfill every execution requirement. Additionally, the presented distribution patterns are applicable to the distribution of data processing in general.

Type-I: Cloud-only (Figure 4, left)
Despite many advantages of Edge Computing, the Type-I distribution pattern still is a viable option. Introducing

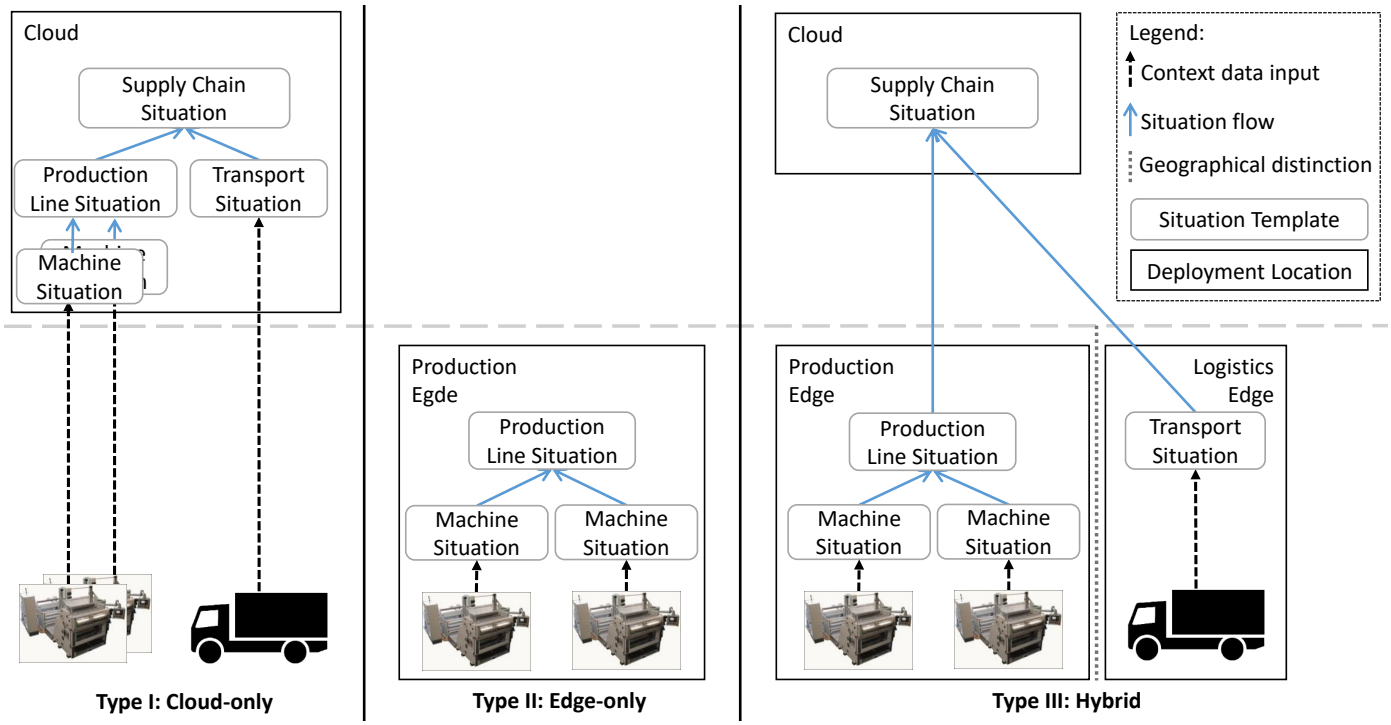

Figure 4. Distribution patterns [1]

Edge Computing is no trivial task and comprises multiple challenges [7]. Companies with low IT experience or no IT department benefit from outsourcing IT infrastructure and expertise to third-party cloud providers. This oftentimes is the case for SMEs, which then can solely focus on their products and the pay-as-you-go model provides a cost-effective and scalable infrastructure. Type I has the following implications regarding our requirements:

- R_4 **- Low latency:** Currently, when using an off-premise cloud, the requirement of 1 ms is already violated by the network latency itself. Therefore, requirement R_4 cannot be fulfilled.
- R_5 **- Low network traffic:** Since all context data must be sent to the cloud first, network traffic cannot be reduced. Requirement R_5 is not fulfilled.
- R_6 **- Reduced costs:** The calculation of costs is always very use case specific. If a company already outsourced its IT infrastructure to the cloud, then the introduction of Edge Computing results in new costs for hardware and IT staff. For scenarios, in which the data amount is relatively small or irregular, the gained advantages may not be worth the expenses. Therefore, the Type-I pattern can fulfill requirement R_6.
- R_7 **- Data security & privacy:** Since all context data is sent to the cloud, new security risks are introduced. Furthermore, company policies might prohibit sending sensitive or personal context data to the cloud. Therefore, requirement R_7 is not fulfilled.
- R_8 **- Cross company situation recognition:** Since all data is available in the cloud, companies can work together to execute a collaborative situation recognition. Requirement R_8 is fulfilled.

As shown, the Type-I pattern does not fulfill most requirements. Still, in non-critical scenarios where high latency is acceptable, the network traffic is low or fluctuating and the data is allowed to be sent to the cloud by the companies' policies or government regulations, the Type-I pattern is a sensible option. Especially for SMEs, the cost model of a public cloud is very attractive in comparison to self-managed data centers [25].

Type-II: Edge-only (Figure 4, middle)
In comparison, the Type-II distribution pattern describes the execution of the whole situation recognition at the edge. As already mentioned, this is only possible if all context data is available at the edge. Therefore, the situation recognition of local situations is best-suited for an edge-only execution. Type II has the following implications regarding our requirements:

- R_4 **- Low latency:** Yi et al. [26] show that latency can be reduced by 82% by moving an application to the edge of the network. As the situation recognition is executed as close as possible to the data sources, the requirement R_4 is fulfilled. With an execution time of 3ms for our situation recognition [18], the overall latency is kept comparably low.
- R_5 **- Low network traffic:** No context data is sent to the cloud, therefore, network traffic stays low and requirement R_5 is fulfilled.
- R_6 **- Reduced costs:** Floyer [27] presents a scenario of a wind-farm to project potential costs savings by additionally using Edge Computing with Cloud Computing instead of a cloud-only solution. An assumed 95% reduction in network traffic results in a cost reduction of about 64%, already including the on-site equipment for Edge Computing. For a cost-effective usage of Edge Computing, the data of the wind-farm had to be

reduced by at least 30% at the edge in order to reduce the costs for network traffic and thereby lower the overall costs. However, continuous IT staff and management of the on-site equipment as well as security measurements are not included. Therefore, again, costs are strongly use case dependent. Introducing Edge Computing does not increase the costs in general but they depend on the amount of saved network traffic. Therefore, requirement R_6 can be fulfilled.

- R_7 **- Data security & privacy:** One of the main concerns regarding the adoption of Cloud Computing still is security, especially in companies with few experience with Cloud Computing. Security and privacy of data is increased, since all context data and situations remain at the edge, i.e., a local network controlled by its company. Requirement R_7 is fulfilled.
- R_8 **- Cross company situation recognition:** In general, the data sources of different companies are geographically distributed and not in the same local network. Therefore, a cross company situation recognition is not possible. Requirement R_8 is not fulfilled.

Most requirements are fulfilled. However, more complex scenarios (cf. Figure 2) cannot be mapped to this pattern because of geographically distributed data sources. Therefore, the Type-II distribution pattern is best suited for company-internal situation recognition that fulfills critical requirements, such as latency and security. Especially in mobile environments, e.g., an autonomous truck, with high-volume data, the Type-II pattern is a good option.

Type-III: Hybrid (Figure 4, right)

Neither a Type-I nor a Type-II distribution pattern presents a viable option for our motivating scenario, since the truck produces too much data for a cloud-only solution and the geographical distribution of the data sources prevents an edge-only solution. Therefore, in the Type-III distribution pattern, the situation recognition is distributed to both the cloud and the edge. This leads to the recognition of local situations at the edge and global situations in the cloud and their advantages.

- R_4 **- Low latency:** The latency for local situations is reduced as described in Type-II. However, global situations are evaluated in the cloud and the latency is as described in Type-I. Therefore, the requirement R_4 is fulfilled only for local situations.
- R_5 **- Low network traffic:** As in Type-II, network traffic can be saved by shifting the situation recognition to the edge. The situation objects of the local situations must be sent to the cloud for the evaluation of global situations, thereby increasing network traffic. However, by using context stripping, the data size of situation objects can be massively reduced and still enable further processing of global situations. Therefore, requirement R_5 is fulfilled.
- R_6 **- Reduced costs:** The potential cost savings correspond to the cost savings of Type-II. However, by using context stripping for local situations, we reduce network traffic and thereby costs and still enable a situation recognition for complex scenarios. Therefore, requirement R_6 can be fulfilled.
- R_7 **- Data security & privacy:** Security and privacy of local situations match the Type-II pattern. Again, when using context stripping for local situations, we support complex scenarios and do not have to send sensitive context data within situation objects to the cloud. Therefore, R_7 is fulfilled.
- R_8 **- Cross-company situation recognition:** As in the Type-I distribution pattern, a collaborative situation recognition is possible. However, a big advantage is gained by using context stripping. Possibly sensitive context data of each company remains at their respective edge. Only context-stripped local situations are sent to the cloud for the collaborative evaluation of the global situation. Therefore, requirement R_8 is fulfilled.

TABLE I. FULFILLMENT OF EXECUTION REQUIREMENTS BY THE DISituation TemplateRIBUTION PATTERNS

	R_4	R_5	R_6	R_7	R_8
Type-I: Cloud-only	X	X	(✓)	X	✓
Type-II: Edge-only	✓	✓	(✓)	✓	X
Type-III: Hybrid	✓	✓	(✓)	✓	✓

Except reducing the latency for the evaluation of global situations, all requirements are fulfilled by this hybrid approach. Especially the usage of context stripping presents multiple advantages when transferring local situations to the cloud. The Type-III distribution pattern is best-suited for complex scenarios with multiple data sources that require a fast reaction to local situations and a centralized situation recognition of global situations without increasing the network traffic. Multiple companies can collaborate without sharing sensitive data or infringing government regulation.

Table I summarizes the analysis of the different distribution patterns. As shown, the Type-III hybrid approach fulfills all execution requirements. However, the potential costs are very use case specific and cannot be generalized (therefore, depicted in brackets). Consequently, if the fulfillment of all execution requirements is not mandatory, choosing a different pattern might be more cost-effective.

V. From Modeling to Deployment

In this section, we describe how distributed situation recognition can be realized from the modeling of the situation using Situation Templates to the actual deployment in order to recognize the modeled situation.

A. Tool-based modeling support

In Section IV-A, we present the layered modeling approach, resulting in modeling improvements to enable the reusability and distribution of Situation Templates. However, especially modeling complex Situation Templates is still a cumbersome and error-prone task without any tool support. To prevent this, we introduce the Situation Template Modeling Tool (STMT), a graphical web-based tool to support users with the modeling of Situation Templates. The previously introduced Situation Template Schema ensures the validity of a Situation Template and is integrated in the STMT to ensure the modeling of valid Situation Templates. Figure 5 depicts the modeling of the Situation Template *Production Line critical*. For illustration purposes, in contrast to the previously introduced Situation Template, one situation input was changed to a sensor input to

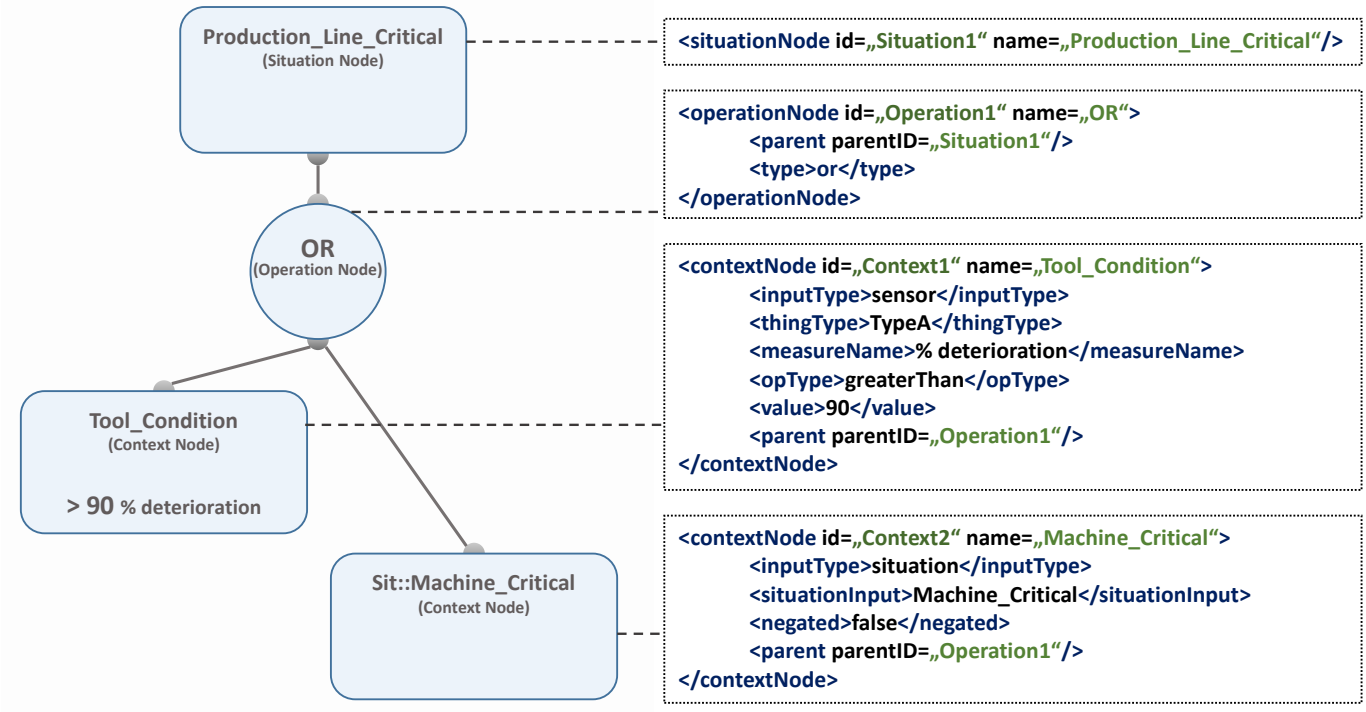

Figure 5. Modeling a Situation Template with a situation as context input and corresponding XML snippets

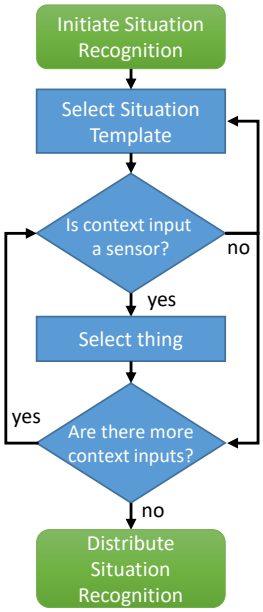

Figure 6. Flowchart for initializing the situation recognition

show the syntactical differences between a situation and a sensor input. As depicted, a Situation Template has a tree structure whereby the root node is the situation node. The only child of a situation node must be an operation node and represents one of the logical operators *AND, OR* or *XOR*. This operation node combines multiple children, which are either context nodes or additional operation nodes. The leaf nodes always constitute context nodes, either sensor input (*CPU_Load)* or situations (*Sit::Machine_Critical)*. Furthermore, on a conceptual layer, we divided the context nodes and condition nodes into separate nodes. However, for more clarity, we combined both nodes into a single one in the STMT.

On the right side of Figure 5, the corresponding XML snippets of the nodes are shown. Situation nodes and all other nodes contain an *id* and the *name* of the node. The id is used for the internal linking of nodes within a Situation Template. The name of the situation node is carried on as the name of the resulting situation object. The operation node further contains the element *parent* to enable the linking of nodes using the id. The element *type* describes the modeled logical operator. As mentioned, context nodes can have either sensor input or situations as input. This option is defined by the element *inputType*. Using a sensor input, the next element is *thingType*. Again, Situation Templates are generic and not defined for a specific thing. Therefore, we define *thing types*, i.e., a class of structurally identical things for which the same Situation Template can be used, since the things possess the same sensors. The condition that has to be met by the sensor input is defined by the elements *value*, i.e., the threshold, and *opType*, i.e., the operation type. The element *measureName* is for visualization purposes only. In comparison, a context node using situations as input constitutes the element *situationInput*. Since a situation is defined by a Situation Template, the value of this element refers to the name of a Situation Template. Using the element *negated*, we have the possibility to negate the Boolean value of a situation object. Additionally, to support the modeler, a database is connected to present possible thing types as well as previously modeled Situation Templates. Afterwards, Situation Templates can be stored in and loaded from a Situation Template repository.

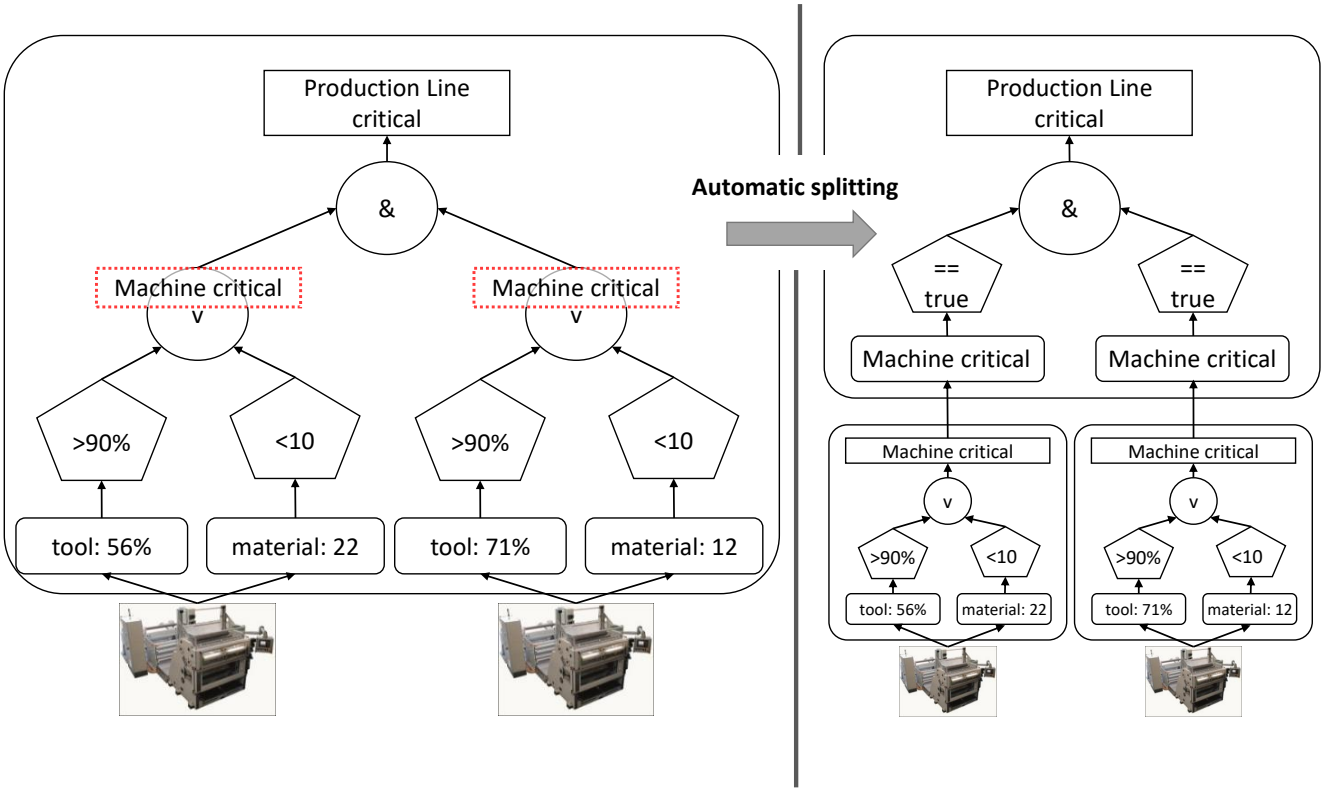

Figure 7. Automatic Splitting of Situation Templates

B. Initiating the Situation Recognition

With the STMT, it is possible to easily model complex Situation Templates. In the following, we present the procedure of initiating the situation recognition based on the modeled Situation Template. Figure 6 depicts the flowchart starting with the selection of a Situation Template. In our previous work, only one thing per Situation Template was allowed, therefore, the next step was the selection of a thing and the process was finished. Now, multiple things as well as situations can be used as input and the process must be changed accordingly. We traverse all context nodes and check their input type. If it is a sensor input, the corresponding thing must be selected that provides the sensor input. After that, the next context node is regarded. If the context node has a situation as input, the corresponding Situation Template is selected and the next context node of the former Situation Template is regarded. This newly selected Situation Template might contain situations as input as well, therefore, we pass through a recursive process. The process is finished when all things that are needed for the situation recognition are selected. This process describes a clean start scenario whereby no situation recognition is running and, therefore, the situation recognition for all Situation Templates has to be initiated.

C. Distributing to Edge and Cloud

In our extended approach, the Situation Templates are modeled using the STMT and the deployment can be initiated from the tool. The last step is the actual distribution to the edge and cloud. To support the user as much as possible, the distribution process can be divided into two steps: 1) automatic splitting, and 2) module distribution, which are introduced in the following.

1) Automatic Splitting: As shown in Figure 7, there are two ways to model a Situation Template. On the left side of the figure, the situation *Production Line critical* is modeled within one Situation Template. On the right side, the same situation is modeled by using the Layered Modeling Approach. By creating separate Situation Templates, we enable the distribution of those Situation Templates to the edge and the cloud. However, in simple scenarios like this one, the modeling on the left side of Figure 7 might be faster, easier and more intuitive. Still, to gain the advantages of a distributed situation recognition even when the user models a single Situation Template, we introduce *Automatic Splitting*. The splitting mechanism detects, if possible, *local* situations (i.e., *Machine critical* in Figure 7; left side) that can be extracted and splits the Situation Template into smaller ones (Figure 7; right side). This method can only be executed after the initialization, i.e., after selecting the things for the Situation Template. Since Situation Templates are modeled in a generic way, only then it is known, which context input belongs to which thing and, therefore, if context inputs originate from the same source, e.g., the same edge node. A prerequisite is that each thing contains meta data about its edge environment that is uniquely identifiable, e.g., by an *edgeID*. As result, each context node can be assigned to the same edge environment as the thing that acts as the context input for this specific context node. Therefore, the context node is annotated with the *edgeID* of the thing. The same applies to the condition nodes (as mentioned in Section V-A, in the implementation context nodes and condition nodes were combined). At operation nodes, multiple context nodes are

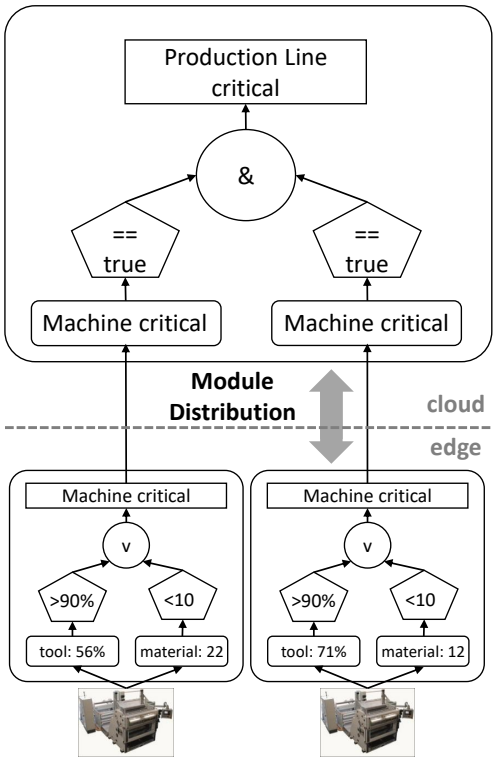

Figure 8. Modular distribution of Situation Templates to edge and cloud

combined and their annotated *edgeIDs* are compared. If they coincide, the operation node is annotated with the *edgeID*. Otherwise, the operation node cannot be distributed to a single edge node, since the different context inputs originate from different sources. This method is executed across all nodes of the Situation Template, starting from all context inputs up to the highest-level operation node. If the highest-level operation node is annotated with an *edgeID*, the whole Situation Template can be executed at the edge node. If only lower-level operation nodes are annotated, the Situation Template is split at this point and that operation node constitutes the highest level operation node in the newly created Situation Template.

2) Module Distribution: The last and final step is the module distribution. Now, we have different Situation Templates, which can be distributed to the edge and the cloud. If the nodes within the Situation Template are all annotated with the same *edgeID*, that Situation Template will be distributed to the corresponding edge environment. If no or multiple *edgeIDs* are present, the Situation Template is distributed to the cloud. Each environment, edge or cloud, contains a system for situation recognition (e.g., a CEP-based system, like Esper [28] and a messaging middleware (e.g., a MQTT-based system, like Mosquitto [29]). At runtime, machines periodically send their data to the message broker running at the edge node. Thereby, not only the situation recognition system can access the data but different applications as well. The situation recognition subscribes to the machine's data and its results, i.e., a situation object, are published to the message broker again, so that applications at the edge can access the situation objects directly. In parallel, all situation objects are mirrored to the message broker in the cloud. Therefore, all applications in the cloud can access the situation object as well and, more importantly, the situation recognition system in the cloud can use them as context inputs for the remaining situation recognition, which as well publishes the resulting situation objects to the message broker. In conclusion, this approach guarantees a flexible and scalable architecture for a distributed situation recognition using widely known and utilized technologies, such as CEP and messaging.

VI. Summary and Future Work

In this paper, we present an approach for distributed situation recognition. To support the distribution, we extend the Situation Template Schema so that multiple things and situations can be used for context input using a layered modeling approach. Furthermore, we present the concept of context stripping to reduce network traffic by removing the associated context of situation objects. We examine three distribution patterns based on execution requirements that are important for a situation recognition in complex environments. In addition, we describe how distributed situation recognition can be realized from the modeling of the situation using Situation Templates to the actual deployment. This is done by the introduced tool-based modeling support and an automated distribution of Situation Templates among the edge and backend cloud. This article is a revised and extended version of the SMART 2018 paper "Layered Modeling Approach for Distributed Situation Recognition in Smart Environments" [1].

In future work, we intend to create a sophisticated cost model, since choosing a suitable distribution pattern is very use-case dependent. Additionally, the management of the situation recognition after splitting, especially in the distribution pattern *Type III: Hybrid*, can become very complex and needs to be considered in future work. This way, users can receive a more detailed decision support based on their specific properties and requirements, which can lead to a faster adoption of new technologies like Edge Computing.

Acknowledgment This work is partially funded by the BMWi project IC4F (01MA17008G).

References

[1] M. Mormul, P. Hirmer, M. Wieland, and B. Mitschang, "Layered Modeling Approach for Distributed Situation Recognition in Smart Environments," in Tagungsband: SMART 2018, The Seventh International Conference on Smart Cities, Systems, Devices and Technologies. Xpert Publishing Services, Juli 2018, Konferenz-Beitrag, pp. 47–53. [Online]. Available: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2018-28&engl=

[2] D. Lucke, C. Constantinescu, and E. Westkämper, Manufacturing Systems and Technologies for the New Frontier: The 41st CIRP Conference on Manufacturing Systems May 26–28, 2008, Tokyo, Japan. London: Springer London, 2008, ch. Smart Factory - A Step towards the Next Generation of Manufacturing, pp. 115–118.

[3] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," Computer Networks, vol. 54, no. 15, 2010, pp. 2787 – 2805.

[4] J. S. He, S. Ji, and P. O. Bobbie, "Internet of things (iot)-based learning framework to facilitate stem undergraduate education," in Proceedings of the SouthEast Conference. ACM, 2017, pp. 88–94.

[5] M. Wieland, H. Schwarz, U. Breitenbücher, and F. Leymann, "Towards situation-aware adaptive workflows: SitOPT – A general purpose situation-aware workflow management system," in Pervasive Computing and Communication Workshops (PerCom Workshops). IEEE, 2015, pp. 32–37.

[6] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, S. G. Sáez, and F. Leymann, "Situation recognition and handling based on executing situation templates and situation-aware workflows," Computing, 10 2016, pp. 1–19.

[7] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet of Things Journal, vol. 3, no. 5, 2016, pp. 637–646.

[8] K. Häussermann, C. Hubig, P. Levi, F. Leymann, O. Simoneit, M. Wieland, and O. Zweigle, "Understanding and designing situation-aware mobile and ubiquitous computing systems," in Proc. of intern. Conf. on Mobile, Ubiquitous and Pervasive Computing. Citeseer, 2010, pp. 329–339.

[9] Q. Fang, Y. Zhao, G. Yang, and W. Zheng, Scalable Distributed Ontology Reasoning Using DHT-Based Partitioning. Springer Berlin Heidelberg, 2008, pp. 91–105.

[10] X. Wang, D. Q. Zhang, T. Gu, and H. Pung, "Ontology based context modeling and reasoning using OWL," in Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on, 2004.

[11] W. Dargie, J. Mendez, C. Mobius, K. Rybina, V. Thost, A.-Y. Turhan et al., "Situation recognition for service management systems using OWL 2 reasoners," in Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on. IEEE, 2013, pp. 31–36.

[12] J. Attard, S. Scerri, I. Rivera, and S. Handschuh, "Ontology-based situation recognition for context-aware systems," in Proceedings of the 9th International Conference on Semantic Systems. ACM, 2013, pp. 113–120.

[13] B. Schilling, B. Koldehofe, U. Pletat, and K. Rothermel, "Distributed heterogeneous event processing: Enhancing scalability and interoperability of cep in an industrial context," in Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems. ACM, 2010, pp. 150–159.

[14] N. P. Schultz-Møller, M. Migliavacca, and P. Pietzuch, "Distributed complex event processing with query rewriting," in Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, ser. DEBS '09. New York, NY, USA: ACM, 2009, pp. 4:1–4:12. [Online]. Available: http://doi.acm.org/10.1145/1619258.1619264

[15] K. V. Laerhoven and H. W. Gellersen, "Spine versus porcupine: a study in distributed wearable activity recognition," in Eighth International Symposium on Wearable Computers, vol. 1, Oct 2004, pp. 142–149.

[16] A. K. Dey, "Understanding and using context," Personal and ubiquitous computing, vol. 5, no. 1, 2001, pp. 4–7.

[17] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, and F. Leymann, "SitRS - A Situation Recognition Service based on Modeling and Executing Situation Templates," in Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing, 2015, Konferenz-Beitrag, pp. 113–127.

[18] A. C. Franco da Silva, P. Hirmer, M. Wieland, and B. Mitschang, "SitRS XT-Towards Near Real Time Situation Recognition," Journal of Information and Data Management, 2016.

[19] M. Mormul, P. Hirmer, M. Wieland, and B. Mitschang, "Situation model as interface between situation recognition and situation-aware applications," Computer Science - Research and Development, November 2016, pp. 1–12.

[20] O. N. Yilmaz, Y.-P. E. Wang, N. A. Johansson, N. Brahmi, S. A. Ashraf, and J. Sachs, "Analysis of ultra-reliable and low-latency 5g communication for a factory automation use case," in Communication Workshop (ICCW), 2015 IEEE International Conference on. IEEE, 2015, pp. 1190–1195.

[21] O. Moll, A. Zalewski, S. Pillai, S. Madden, M. Stonebraker, and V. Gadepally, "Exploring big volume sensor data with vroom," Proceedings of the VLDB Endowment, vol. 10, no. 12, 2017.

[22] J. Budomo, I. Ahmad, D. Habibi, and E. Dines, "4g lte-a systems at vehicular speeds: Performance evaluation," in Information Networking (ICOIN), 2017 International Conference on. IEEE, 2017, pp. 321–326.

[23] E. Directive, "95/46/ec of the european parliament and of the council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data," Official Journal of the EC, vol. 23, no. 6, 1995.

[24] P. Hirmer, M. Wieland, U. Breitenbücher, and B. Mitschang, "Automated Sensor Registration, Binding and Sensor Data Provisioning," in CAiSE Forum, 2016.

[25] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computingthe business perspective," Decision support systems, vol. 51, no. 1, 2011, pp. 176–189.

[26] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on. IEEE, 2015, pp. 73–78.

[27] D. Floyer, "The vital role of edge computing in the internet of things," Oct. 2015. [Online]. Available: https://wikibon.com/the-vital-role-of-edge-computing-in-the-internet-of-things/

[28] "Complex event processing streaming analytics." [Online]. Available: http://www.espertech.com/

[29] "Eclipse mosquitto an open source mqtt broker." [Online]. Available: https://mosquitto.org/

All links were last followed on May 21, 2019.