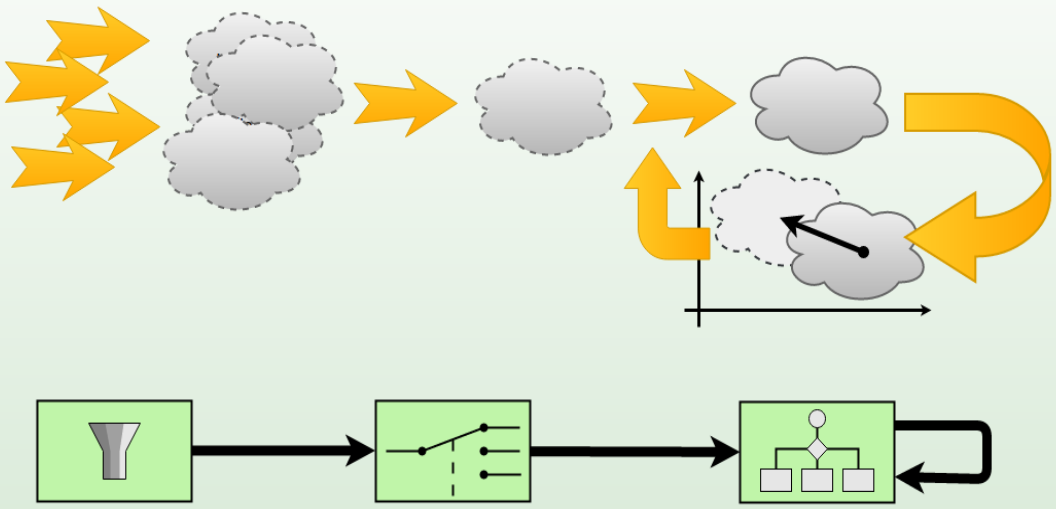
TU Clausthal



Marc Schaaf

Situation-Aware Adaptive Event Stream Processing

A Processing Model and Scenario Definition Language

SSE-Dissertation 15

Software Systems Engineering

Institut für Informatik
Lehrstuhl von Prof. Dr. Andreas Rausch

Situation-Aware Adaptive Event Stream Processing

A Processing Model and Scenario Definition Language

Doctoral Thesis
(Dissertation)

to be awarded the degree
Doctor of Engineering (Dr.-Ing.)

submitted by

Marc Schaaf
from Herford

approved by the Faculty of
Mathematics/Computer Science and Mechanical Engineering,
Clausthal University of Technology,

Date of oral examination
09.06.2017

Dean:

Prof. Dr. rer. nat. Jürgen Dix

Chairperson of the Board of Examiners:

Prof. Dr.-Ing. Michael Prilla

1st Supervisor & Reviewer:

Prof. Dr. rer. nat. Andreas Rausch
Institut für Informatik
Technische Universität Clausthal
Arnold-Sommerfeld-Str. 1
38678 Clausthal-Zellerfeld
Germany

2nd Supervisor & Reviewer:

Prof. Dr.-Ing. Arne Koschel
Fakultät IV - Wirtschaft und Informatik
Hochschule Hannover
Ricklinger Stadtweg 120
30459 Hannover
Germany

3rd Reviewer:

Prof. Dr. rer. nat. Klaus Schmid
Institut für Informatik
Universität Hildesheim
Universitaetsplatz 1
31141 Hildesheim
Germany

Abstract

This work defines a situation aware adaptive event stream processing model and scenario specification language. The processing model and language allow the specification of stream processing tasks, which support an automatic scenario specific adaptation of their processing logic based on detected situations and interim processing results.

The **motivation** for this work lies in the missing support of current state of the art Event Stream Processing (ESP) systems for such a „situation-aware adaptive Event Stream Processing" which leads to the **problem** that for each scenario that requires this kind of processing, a new processing system needs to be designed, implemented and maintained. It is therefore the aim of this work to ease the development of such situation aware adaptive processing systems.

An example for such a **scenario** is the detection and tracing of solar energy production drops caused by clouds shading solar panels as they pass. The scenario requires a processing system to handle large amounts of streaming data to detect a cloud (possible situation). The later verification of the cloud as well as its tracking however only requires a small *situation specific* subset of the overall streaming data, e.g. the measurements from solar panels of the affected area and its surroundings. This subset changes its characteristics (location, shape, etc) dynamically as the cloud traverses the region. The scenario thus requires a *situation-aware adaptation of its processing setup* in order to focus on a detected cloud and to track it.

This work **approach**es the problem by defining *a situation-aware adaptive stream processing model* and a matching *scenario definition language* to allow the definition of such processing scenarios for a *scenario independent processing system.* The **requirements** for the definition of the model and language are the result of an analysis of three distinct scenarios from two application domains. The designed **model** defines situation aware adaptive processing in three main phases:

Phase 1: In the *Possible Situation Indication* phase, possible situations are detected in a large set of streaming data.

Phase 2: The *Focused Situation Processing Initialization* phase determines whether an indicated possible situation needs to be investigated or if it can be ignored, for example because the situation was already under investigation. If a potential situation needs to be investigated, a new situation specific focused processing is started.

Phase 3: In the *Focused Situation Processing* phase, possible situations are verified and

an in depth investigation of the situation including the adaptation of the processing setup based on interim results is possible.

The **evaluation** demonstrates that the language and processing model fulfill the defined requirements by providing an application domain and scenario independent mechanism to define and execute situation aware adaptive processing tasks. For the evaluation, a processing system prototype was created and two scenarios from two different domains realized. The first scenario is the Cloud Tracking scenario introduced above. The second scenario is the detection and tracing of Denial of Service Attacks. Several tests where performed to verify that the scenario definition provides the required information for the processing system and to verify that the designed processing model allows the required situation-aware adaptive processing on a scenario independent processing system.

Acknowledgements

I would like to thank my supervisors Prof. Dr. Rausch and Prof. Dr. Koschel for their support and constructive feedback throughout the numerous meetings and discussions regarding this work. Further I would like to thank Arne Koschel for his continuous support of my studies and especially with regards to this project. Moreover, I would like to thank Prof. Dr. Schmid for his feedback for the final version of this work.

Furthermore I would like to thank the members of the KPE for the interesting workshops which offered me the platform to present my work and the possibility to get in touch with other Ph.D. students.

I would also like to thank the University of Applied Sciences and Arts Northwestern Switzerland for the scholarship which allowed me to develop my thesis in parallel to my normal work obligations.

Moreover I like to thank the whole project team of the Eurostars Project DYNE and in particular Erik Bunn and Topi Mikkola for the various discussions and the chance for such an interesting research project which laid the foundations for this thesis.

Last, but not least, I would like to thank my family and Gwen for their patience and support during this endeavor and in particular Gwen for her reviews and constructive feedback.

Thanks!

Contents

1. Introduction

Contents

Event Stream Processing (ESP) applications play an important role in modern information systems due to their capability to rapidly analyze large amounts of information and to quickly react based on it. They follow the approach to produce notifications based on state changes (e.g. stock value changes) represented by events, which *actively* trigger further processing tasks. They contrast to the typical store and process approaches where data is gathered and processed later in a batch processing fashion which involves a higher latency. Event Stream Processing applications achieve scalability even for large amounts of streaming event data by partitioning incoming data streams and assigning them to multiple machines allowing for parallel processing.

Due to those properties, Event Stream Processing based analytical systems are likely to have a further increasing relevance in future IT systems „as society demands smarter ways for managing electric power, water, health, retail and distribution, traffic, and safety – smarter meaning responding better and *faster* to changing conditions" [CEvA11, p. 6]. Furthermore, it is very likely that future Event Stream Processing applications will have to handle even larger amounts of data while taking up increasingly complex processing tasks to allow for near real-time analytics to take place. In addition, these new tasks require the Event Stream Processing systems to become more flexible with regard to their data processing capabilities, while retaining their scalability and near real-time processing capabilities.

This work defines a processing model and scenario description language for *situation-aware adaptive event stream processing*, specifically suitable for scenarios which require the detection and analysis of situations in large sets of event streams where once a situation was detected, the dynamic situation-specific analysis can take place on a small situation-specific subset of the overall set of event streams.

1.1. Motivating Scenario

This work is motivated by several scenarios arising from two application domains, telecommunications (Telco) network and Smart Grid monitoring (Section 2). For the discussions in this work, the focus lies on the Smart Grid *cloud tracking scenario* (Section 2.1.1) as this scenario resembles the central challenges that can be found in a similar fashion in all the other use cases. The discussion which lead to this conclusion is given in Chapter 2.

The *tracking of cloud movements* based on the monitoring of solar panel installations for their present energy production [WSB+14] is one example of a scenario that requires a situation-aware adaptive processing. The gained information can be used for short term forecasting of the cloud movements and with it the production of the solar panel installations. Such forecasts are required as solar power production tends to fluctuate within a few minutes from a high energy production to nearly no energy production and back due to a passing cloud which momentarily shades the solar panels. While such small production anomalies are not relevant for the stability of the overall energy distribution grid, they are causing local voltage fluctuations in the neighboring energy consumers (e.g. households)[1]. A prediction of such production changes can be used to reduce this effect by introducing compensatory actions like disabling some energy consumers or changing the setting of the feeding transformer.

A prognosis can be generated by tracking individual clouds based on the caused energy production drops as they move across the country, temporarily shading various solar panel installations. The detection of a cloud is based on a sudden and significant drop in the energy production of one of several solar panel installations while the surrounding installations produce energy as expected (Figure 1.1.1). Based on this information the rough size and shape of the cloud can be estimated. Over time, the regain of the energy production of previously shaded installations together with the production drop of previously normal behaving installations can be used to estimate the cloud's movement direction and velocity.

[1] Here neighboring is defined via the energy distribution grids topology, as households that are connected to the same feeding transformer.

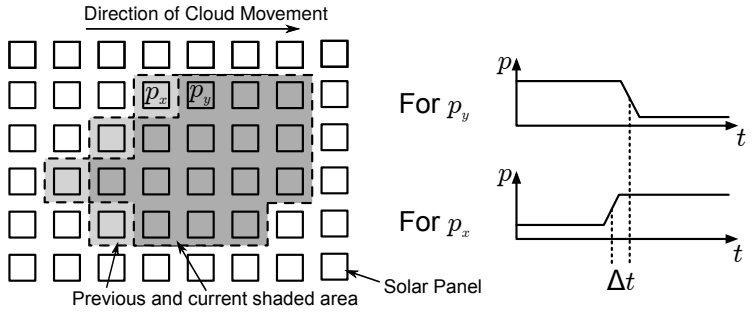


Figure 1.1.1.: A cloud moving across several solar panel installations resulting in an area of reduced energy production.

1.2. Scenario Characteristics with Regard to Event Stream Processing

From the perspective of a stream processing system that implements such a cloud tracking, the scenario has a set of characteristic properties which can also be found in scenarios arising from other domains like for example in the area of telecommunications network monitoring. The following overview of these characteristics is based on the scenario analysis in Chapter 2 (Section 2.2):

SC1: *Possible Situations need to be identified* in a large amount of streaming data, potentially in all available data streams.

SC2: The *Possible Situation Indication needs to be rapid also for large amounts of streaming data* (near real-time).

SC3: Found *Possible Situations require their verification and an in-depth situation-specific analysis* based on *streaming data* and *static background knowledge.*

SC4: The situation-specific analysis only requires access to a small subset of the overall set of event streams.

SC5: The *part of the stream data and background knowledge* needed for the situation-specific analysis *can not be determined before the situation has been detected.*

SC6: The part of the stream and background knowledge needed during the analysis changes based on interim analysis results.

Based on the scenarios and the above properties, a formal specification of the type of data processing needed is developed in Subsection 2.2.2. The formal specification shows that due to the last two properties (SC5 & SC6), a situation-specific analysis of a possible situation i has to be seen as a set of situation-specific analysis functions $F_i := \{sa_{i,1}, ..., sa_{i,n_i}\}$ (Formula 2.3 on page 24) rather than a single situation independent analysis function.

Here all functions $sa_{i,k}$ in the set F_i depend on the interim situation analysis result from the previous function $sa_{i,k-1}$, while the very first function $sa_{i,1}$ depends on the detected

possible situation $i \in I$ itself. Furthermore, not only is the function call dependent on the (possible) situation but also the definition of the function itself (Formula 2.4 on page 25) as each of the functions is focused on a specific (possible) situation, which may change over time.

Due to this dependence on interim results, the use of a statically set up processing system (thus with static processing functions) for the situation-specific analysis can not be used without sacrificing the benefit of a tailored situation-specific analysis functions (SC4) which only looks at the required subset of the available event streams.

1.3. Resulting Problem

Due to the need for situation-specific processing functions which need to be created and adapted during run-time, the actual processing that is executed by an ESP system changes over time:

1. When a possible situation is detected by the processing system, a situation-specific processing function, a *"focused processing function",* needs to be generated and executed.

2. Once the situation-specific processing is running, it needs to be adapted as the analysis detects new information for the investigated situation.

Therefore, in order to support this kind of processing *an ESP system has to support the adaptation of the processing function.* As discussed in Chapter 3, current ESP systems have some general support for the adaptation of processing functions during run-time (e.g. [AAB+05, YKPS07, HSS+14]). Such adaptations can be triggered by an external system or by internal optimizations of the ESP system. Optimization mechanisms typically supported however only adapt the processing within the ESP to increase response time or resource usage. Examples of such optimizations are operator reordering, operator fission / stream partitioning and operator placement on different hosts and cores [HSS+14]. *Such optimizations make no adaptations to the stream processing function itself* as this requires additional knowledge about the domain specific high level scenario that resulted in the stream processing function to be executed by the ESP.

With regard to the scenarios discussed here, the ESP would need to be able to make decisions when and how to adapt the deployed stream processing function based on a processing model that provides the required situation based information and defines a suitable processing semantic. Current ESP systems do not support such a situation-aware processing model that allows to deduce the required processing function adaptations *automatically.* As a result, in current ESP systems, scenarios that require situation-aware stream processing can only be realized by providing custom implementations on top of the stream processing system that realize the required adaptations for the scenario. Typically, such

specialized solutions require much work for each scenario and are hard to maintain and change as not only domain experts are needed but also experts in event stream processing and the used frameworks in order to make changes to these specialized systems.

Problem Statement

In summary, this work addresses the problem that current ESP systems have no support for a situation-aware adaptive stream processing and thus the realization of situation-aware processing tasks requires the design, implementation and maintenance of a custom solution for each distinct application scenario.

1.3.1. Research Question

Based on the discussed problem, the following research question arises:

RQ1: How to allow situation-aware adaptive processing without the need to implement a specialized solution for each scenario that requires a situation-aware adaptive processing?

This general question can be subdivided into the following sub-questions:

RQ1.1: How can a generalized solution provide a processing system suitable to handle large amounts of streaming data for a situation-aware adaptive processing?

RQ1.2: How can a generalized solution define the adaptation steps in a flexible and domain independent way?

RQ1.3: How can a generalized solution define a suitable semantic definition that specifies the behavior during the whole situation-aware adaptive processing, in particular the behavior of automatic adaptations?

RQ1.4: How can situation-aware adaptive processing tasks be specified for a generalized processing system?

1.4. Approach

It is the aim of this work to overcome the discussed problem by defining a *situation-aware adaptive processing model* suitable for the general scenario type defined in Section 1.2. Based on the model a suitable language can be defined which allows the definition of situation-aware adaptive processing tasks. Based on this a run-time system can be designed which implements the designed model and allows the execution of situation-aware adaptive processing tasks.

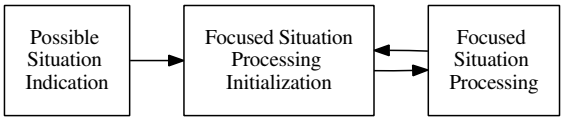


Figure 1.4.1.: High level view of the situation-aware adaptive processing model with the three main processing phases.

1.4.1. Processing Model

The model defines the situation-aware adaptive processing in three main phases, the *Possible Situation Indication*, the *Focused Situation Processing Initialization* and the *Focused Situation Processing* (Figure 1.4.1):

Phase 1: Possible Situation Indication
The Possible Situation Indication phase handles the initial detection of possible situations of interest that may require special attention by a focused situation processing. Such a situation is for example the energy production drop of a monitored solar panel. It is important to note that it is not the aim of this phase to verify if a possible situation concerns an actual situation. Rather, the indication is aimed at a very rapid processing of the incoming data while accepting the generation of false positives.

Phase 2: Focused Situation Processing Initialization
Once a possible situation has been indicated, a Focused Situation Processing may need to be started for its analysis. Here the initialization phase has the responsibility to trigger the set up of such a new processing task but also to decide if a new task should be created or if the indicated possible situation is or was already handled. For this decision it uses scenario-specific rules in combination with defined processing states provided by all Focused Situation Processing instances that are already started (see Phase 3).

Phase 3: Focused Situation Processing
The actual situation-specific processing happens in this third phase and can, in contrast to the first processing phase, be much more time consuming per processed event as the amount of events that need to be processed should already have been reduced dramatically. As a first step, the Phase 3 processing verifies whether the indicated possible situation is a valid situation or whether it is a false indication. Once this verification is complete, the processing can continue with an in-depth analysis of the situation. An important aspect of this analysis process is the possibility to adapt itself based on the current situation's state in order to account for new information on the situation or the change of the situation itself. For example for the cloud tracking scenario, the situation-specific processing has to follow the cloud as it changes its position or size over time. In order to manage the current focus of an ongoing focused situation processing, the processing model defines a focus area and a locked area which mark the set of data streams currently relevant for the analysis process.

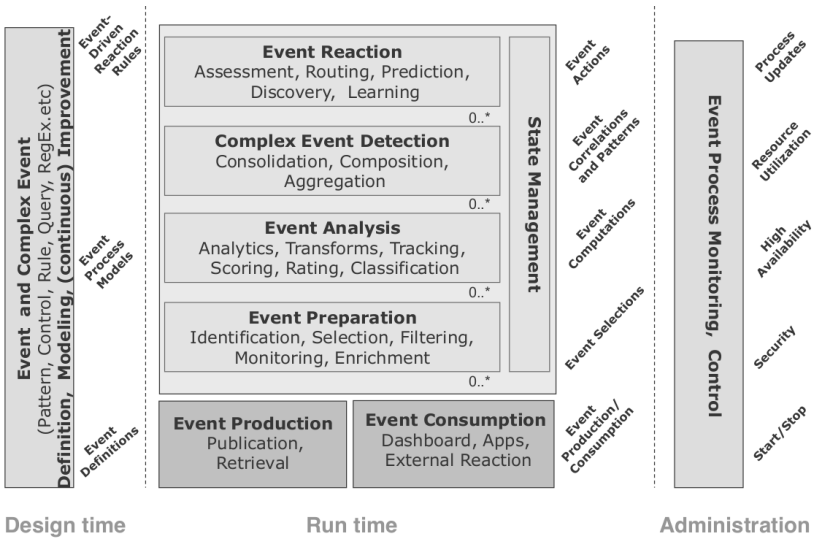


Figure 1.5.1.: Event Processing Reference Architecture [PVAM12]

During the ongoing situation-specific processing, the processing system can produce preliminary processing results in order to inform foreign systems about the current state of the investigated situation like for the cloud tracking scenario, the current position, speed and trajectory of the cloud.

1.4.2. Contributions

Processing Model

A processing model that defines all necessary steps for an adaptive situation-aware processing. The general approach followed by the model is to separate the processing into three major processing phases: Possible Situation Indication, Focused Situation Processing Initialization and Focused Situation Processing. Based on these phases, the model defines the structure for a situation-aware adaptive processing as well as the semantics and the execution process including the adaptation steps needed during processing.

Specification Language

The defined language (Scenario Processing Template Language, SPTL) allows the specification of situation-aware adaptive processing tasks following the defined processing model in order to be executable by a processing system that interprets the processing specifications based on the defined processing model.

1.5. Positioning of the Work

The presented work takes place in the scope of Event Processing with its focus on stream data analytics where the analytical process requires situation-specific adaptations of the deployed processing tasks. With regard to the Event Processing Reference Architecture (Figure 1.5.1) [PVAM12, PVM$^+$12] developed by the Event Processing Technical Society (EPTS), this work can be positioned as follows:

Design time
: From the design perspective, the defined model and specification language extends the typical processing descriptions by *adding the capability to specify situation-aware adaptive processing tasks.* It also introduces a system for the automatic generation of concrete processing tasks based on the specification of processing templates which are evaluated based on separately provided background knowledge as well as run-time information.

Run-time
: *Event Reaction:* From the run-time perspective the designed model extends the Event Reaction with the introduced automatic adaptations of the stream processing statements as a reaction to interim results from the stream processing system.

: *Event Analysis and Complex Event Detection:* The designed model also extends the event tracking as part of the Event Analysis by introducing the capability to classify and track events based on situations and the Complex Event Detection by allowing for situation-specific complex events.

: *State Management:* In order to be able to analyze and track situations, the model extends the State Management by introducing situation-specific states like general situation focused processing or per adaptation step states.

Administration
: *Process Update:* As the model and language allows adaptive situation-aware processing tasks to be defined without the need to implement a custom solution for each scenario, the update of processing tasks changes from the adaptation of a specialized implementation to the mere update of the processing task specification.

: *Resource Utilization:* As a side effect, the defined situation-aware adaptive processing model also impacts the resource utilization as the stream processing can in part take place in a dynamically focused manner, thus requiring the availability of flexible processing resources to cope with increasing and decreasing amounts of analyzed and traced situations.

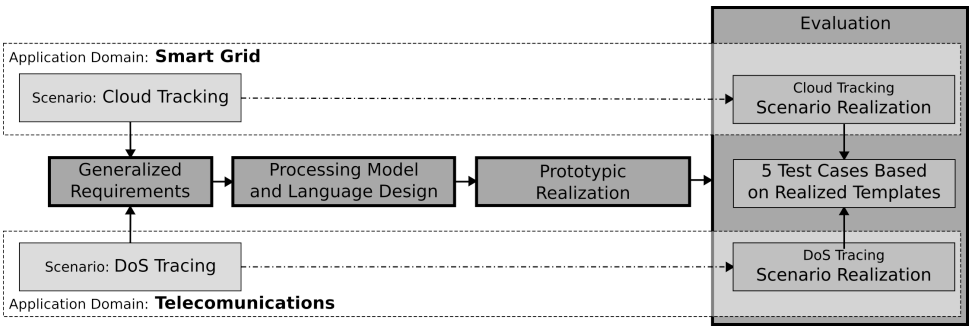


Figure 1.6.1.: Overview of the design and evaluation process of this work.

1.6. Research Design

The overall aim of this work is to design a situation-aware adaptive processing model to overcome current limitations of Event Stream Processing mechanisms with regard to their support for the definition of processing tasks with dynamic situation-specific foci (Section 1.3). The solution is motivated through scenarios from the application areas Smart Grid monitoring and telecommunications network monitoring but is not limited to these areas (Figure 1.6.1).

The research follows the *design research* methodology. Hevner et al. defines design research as "a problem-solving paradigm which seeks to create innovations that defines ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management and use of information systems can be effectively and efficiently accomplished" [HMPR04]. Thus the design research approach suites the overall goal of this work to design a solution for the outlined problems based on an initial assumption. The designed artifact will in turn allow the evaluation of the suitability of the initial approach and the resulting concept to solve the given problems. To allow this evaluation, a prototype is created to test the capability of the approach for a given problem from each of the application domains, Smart Grid monitoring and telecommunications network monitoring.

Based the guidelines defined by Hevner et al. [HMPR04], the following aspects of the application of the design science methodology to this work shall be mentioned:

1.6.1. Design of an Artifact

The central artifact that is designed within this work is a processing model for adaptive situation-aware event stream processing. The design is based on the generalized requirements from three scenarios from two application areas. Therefore, it is the aim of this work to produce a *generalized* solution that is *not only* applicable to one given scenario from one application domain. Instead, the artifact will be usable with *scenarios that follow the characteristics* extracted from the analyzed scenarios.

The design is conducted based on the following steps where the design for each step is con-

ducted based on the defined requirements with a detailed discussion of the design decisions and their alternatives:

D1: Generalized Requirements: Perform a classification of the available scenarios to identify the shared characteristic requirements regarding the definition of the processing model (Chapter 2).

D2: Processing Model: Definition of the processing model that defines the structure and the overall processing flow as a three phased process (Chapter 4).

D3: Domain Specific Language: Based on the processing model, the Domain Specific Language, the "Scenario Processing Template Language" (SPTL), is defined to allow the specification of processing instructions for different scenarios (Chapter 5).

1.6.2. Evaluation of the Designed Artifacts

The suitability of the designed processing model and specification language is evaluated based on a prototypical realization of two scenarios and several test cases based on these scenarios. The evaluation follows the following steps:

E1: Design and Implementation of a Prototype: In order to allow the evaluation of the model and language a prototype is created which implements the processing model and supports the execution of situation-aware adaptive processing tasks based on the defined specification language SPTL. (Chapter 6).

E2: Realization of two Application Scenarios: Based on the specification language SPTL two different scenarios are defined that require a situation-aware adaptive processing from two separate application domains (Smart Grid Monitoring and telecommunications network monitoring) to verify the general applicability of the specification language.

a) **Cloud tracking scenario:** The realization of the investigated Smart Grid scenario by its specification in SPTL for defining a suitable focused processing task (Section 7.3).

b) **Telecommunications Network Monitoring Scenario:** The realization of a scenario from a different application domain than the Smart Grid domain to *demonstrate the generalization* of the designed model and language (Section 7.5).

E3: Test Cases: Based on the two realized scenarios and the designed prototype, five test cases where evaluated to show that the model and language can be used to define and execute situation-aware adaptive processing tasks (Sections 7.4 and 7.6 and Appendix C).

1.7. Dissertation Organization

This work is structured as follows:

Chapter 2 Scenario Requirement Analysis
Three scenarios from two separate application domains are presented. Based on these scenarios, the characteristic properties of a situation-aware adaptive processing are defined which are the foundation for the formal definition of the processing type.

Chapter 3 State of the Art
Based on the generalized processing model from Chapter 2, this chapter discusses the current state of the art with regard to event stream processing systems towards their suitability for a situation-aware adaptive event stream processing.

Chapter 4 Processing Model
Based on the requirements from Chapter 2 and the approach followed by this work (Section 1.4), this chapter defines the situation-aware adaptive processing model based on which situation-aware adaptive processing tasks can be executed.

Chapter 5 Language Definition
Based on the processing model, this chapter defines the Scenario Processing Template Language (SPTL), which can be used to express the scenario-specific aspects of a situation processing task based on the defined processing model.

Chapter 6 Prototype
In order to allow the validation of the processing model and description language, a prototype was created which is discussed in this chapter. The chapter specifically discusses the architecture of the prototype and illustrates how the processing model can be mapped to separate components which can be seen as the starting point for creating a distributed and scalable processing system adhering to the processing model.

Chapter 7 Evaluation
In order to validate the processing model and description language, this chapter discusses the realization of two scenarios from two separate domains as scenario template based on the SPTL. Based on the defined templates several tests where conducted to verify that the processing model provides the required functionality and in turn the SPTL provides the means to express all scenario-specific properties for its processing. Additional test cases are discussed on Appendix C.

Chapter 8 Conclusions and Future Work
Summarizes the work and points out limitations and open questions as a starting point for future work.

Appendix

Provides a complete definition of the SPTL grammar and the complete templates, defined to realize the two scenarios used for the evaluation. Furthermore, it documents three additional test cases as part of the evaluation from Chapter 7.

2. Scenario Requirement Analysis

Contents

This section describes scenarios from each application domain relevant to this work to point out the demand for a situation-aware adaptive processing mechanism. To allow the design of a generalized processing model suitable for all the given scenarios, a comparison is made to extract the characteristics to identify the *problem class* that needs to be solved by the processing model.

2.1. Detailed Description of the Scenarios

This section discusses three application scenarios that motivate this work to lay the foundation for the discussion of their requirements.

2.1.1. Application Area Smart Grid

One of the central aspects of Smart Grids is the integration of information technology with the power-delivery infrastructure. Further the grid will change from unidirectional energy flow (from centralized power plants to the distributed consumers) to a grid that has to support bidirectional energy flows thus allowing distributed energy production like for example from household solar panel installations. As a result, additional information technology will be needed to guarantee for the energy grids stability. The increased information availability will also allow the creation of various sorts of services alongside the physical energy distribution infrastructure to handle the energy production and consumption in a more flexible manner [nis12].

One of the major challenges around the intensified use of information technology within Smart Grids lies in the handling of the data amounts and the rapid analytical processing

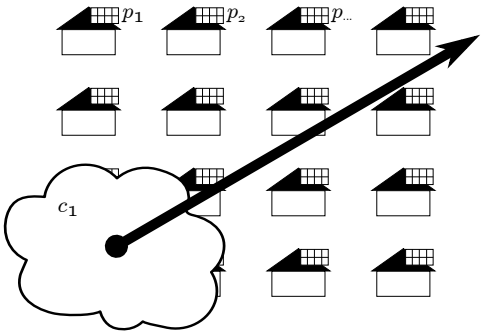


Figure 2.1.1.: A cloud moving across several solar panel installations

of gathered measurement data to identify relevant situations within the power grid and its surroundings. One such scenario lies in the handling of fluctuations of distributed energy production. This can for example be caused by solar panel installations in customer households. The following scenario discusses such a scenario.

Scenario 1 - Cloud Movement Tracking for Dynamic Load Management in Smart Grids
In this scenario, energy production „holes" should be detected and tracked as they are caused by clouds that shade solar panel installations of customer households. A cloud may affect closely related solar panels that have a reduced energy production due to the shading whereas the solar panels outside of the cloud's shadow will still produce their normal amount of energy. This localized drop in the energy production results in the fluctuation of the voltage provided to the households in proximity[1] (Figure 2.1.1). To cope with those fluctuations, switching transformers could installed in the distribution grid. The transformers are able to regulate their output voltage allowing to compensate the effects of fluctuating energy production. To ensure a long lifetime of those transformers, the number of switching operations however should be minimized. Thus, if an IT system would be able to identify and track cloud shadings, this information could be used to optimize the number of switching operations. For example a switching operation could be prevented by reducing the energy consumption of the affected households by temporarily switching of devices while the cloud passes by.

To realize this scenario, a processing system needs to be able to monitor the solar panels to detect such situations. It further needs to be able to deduce all relevant aspects like the position, size and trajectory of the cloud. Further it is important that the information is provided in near real-time so the information can be used to invoke adequate actions.

The scenario requires the *detection* and *tracking* of cloud shadings of solar panel installations. The task can be separated into two parts:

[1] In detail it has an impact on the consumers connected to the same transformer as one or more of the impacted solar panels.

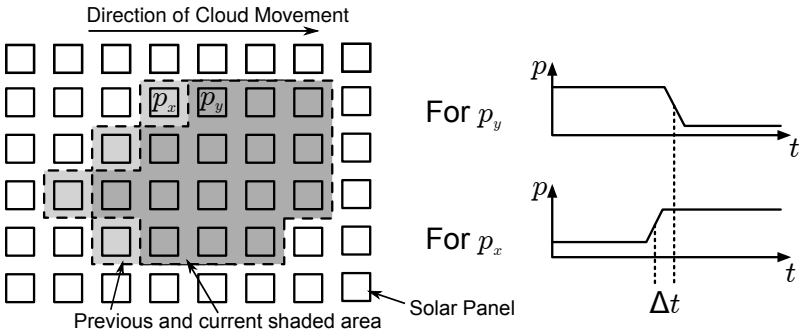


Figure 2.1.2.: Changing energy production over time due to a cloud moving over the panels

Part 1: The *detection* of a cloud which is shading *several* solar panels causing a drop in their energy production and

Part 2: the *tracking* of the cloud's movement based on the changing energy production patterns from several solar panel installations.

For the detection (Part 1), the scenario relies on the analysis of the energy production of solar panel installations. For this it needs to identify sudden drops in the energy production of solar panel installations. If such a drop occurs, a cloud might be the cause. To verify this assumption a comparison with other solar panel installations in close geographical proximity needs to be made. If the production drop is evident in a large enough geographical area of solar panel installations without any installations still producing energy, it can be assumed that the production drop is caused by a cloud. If however only a single solar panel reports a reduced production, the incident should not be considered as being caused by a cloud and should thus not be tracked.

For the tracking of the detected cloud (Part 2) the existence, position and size of the cloud was already determined in the first step. Based on this information, changes in the cloud position and size shall be tracked. As for the detection, the processing is again based on the energy production of the solar panel installations. The tracking of cloud movement is based on the detection of a sudden increase of the energy production by a previously shaded panel in the border area of the shaded area. This event would then be followed by the sudden drop of energy production of a previously unshaded panel adjacent to the previous border of the tracked cloud (Figure 2.1.2):

- With $\pi_{position}(p)$ as the projection to the vector describing the position of the solar panel p and
- $\vec{center}_{c,t} = \sum_{p \in P_{c,t}} \pi_{position}(p)$ with $center_{c,t}$ as the approximate center of the cloud c at time t based on the set $P_{c,t}$ as the set of solar panel installations shaded by the cloud c at time t.

- Based on this, the movement of the cloud c between two points in time t_x and t_y can be approximated as $\vec{movement}_{c,\Delta t_{x,y}} = \vec{center}_{c,t_y} - \vec{center}_{c,t_x}$.

The tracking of size changes can be done in a similar way. For a shrinking cloud, shaded solar panels adjacent to the border of the cloud will become unshaded causing them to produce more energy. For a cloud growing in size, unshaded solar panels adjacent to the previous cloud border will stop to produce energy.

2.1.2. Application Area Large Scale Telecommunications Network Monitoring

Large scale country wide telecommunications networks as they are maintained by telecommunications companies have to handle the traffic from hundreds of thousands or sometimes even millions of customer Internet connections. Such networks consist of a huge number of routers interconnected by various connections with greatly varying bandwidth and reliability. The maintenance of networks of such sizes is a challenging task as simple failures like a single broken connection or router can spread out and cause much larger problems within the overall network due to the automatic traffic re-routing that is done to compensate for one outage of an important connection or router.

The currently available monitoring systems for such systems are well able to provide a near time status information of all the routers and connections in combination with the automatic generation of alerts if failures or drastic changes in the line quality or the utilization occur. These monitoring systems are already heavily used by network operators to oversee the network and to implement manual counter measures if the traffic flow throughout the network is not optimal.

However, these systems currently have the downside that they are raising alerts based on simple local utilization deviation or failures without being able to group together multiple incidents to a larger problem with the actual root cause of the problem. This currently results in a large flood of alerts if a critical connection fails. In such a case, other unrelated problems may not be directly visible to the network operators anymore as it is not easily possible to separate them from the flood of events caused by major failure. This is caused by the lack of current network monitoring systems to dynamically detect problematic situations *and* track related side effects. Based on this shortcoming, two scenarios are discussed in the following sub-sections which both can benefit from a situation-aware adaptive processing as developed in this work.

2.1.2.1. Scenario 2: Telecommunications Network Monitoring for Denial of Service Attacks in Large Scale Telecommunication Networks

The goal of this scenario is to detect and trace a DoS attack through the telecommunication providers network back to its entry point into the network in order to allow the provider to block the traffic from entering its network and to determine whether it has its source

within its own network[2] or is incoming from an upstream provider. Similar to the Cloud Tracking scenario, the task can be divided into two Parts:

Part 1: The detection of a DoS attack at the routers connecting a DoS monitored data center to the providers network and

Part 2: the tracing of the DoS attack back to is origin relative to the provider network.

For the detection (Part 1) of a DoS attack, the monitoring of the routers connecting the data center is required. As a DoS attack against servers in the monitored data center typically results in a significant increase of the packet count and a reduction of the average packet size measured on these routers, based on this change an attack can be detected.

For the tracing (Part 2) of the attack back to the point where it enters the providers network, the correlation of changes from the typical traffic patterns of adjacent routers is required in an incremental way:

1. In the initial step of the tracing of an attack a that was detected at time t_a, the amount of the packet count change $\Delta pc_{a,0}$ and average packet size change $\Delta ps_{a,0}$ can be determined for the border routers that indicated the attack $R_{a,0}$ like for example $R_{a,0} = \{R1, R13\}$ (Figure 2.1.3). The deltas can be deduced by subtraction of the previous measurement values (e.g. from 10 minutes in the past) from the current values

2. In step two, the routers $R_{a,1}$ which are topological adjacent to the routers where the attack was detected are determined. For example $R_{a,1} = \{R2, R5\}$. For each of these adjacent routers, changes in their average packet count and average package size are calculated and compared to the deltas from the indicating routers $R_{a,0}$. If similar changes are detected within the same time t_a on one or a group of the adjacent routers, these routers are assumed to be the origin of the DoS traffic and are thus the basis for the next step of the trace.

3. The process from Step 2 is repeated for each element of $R_{a,1}$ until (a) the trace reaches the border of the providers network or (b) the deltas of the packet count and package size are not distinguishable from normal fluctuations anymore.

2.1.2.2. Scenario 3: Telecommunications Network Monitoring for Link Failures and Correlation of Resulting Link Overloads

The failure of a major communication link in a telecommunication providers network typically causes an automatic re-routing of traffic within the network to compensate for the lost link. The failure of such a link is easily detectable and visualized by common monitoring systems.

[2] A customers network connection could be used by an attacker.

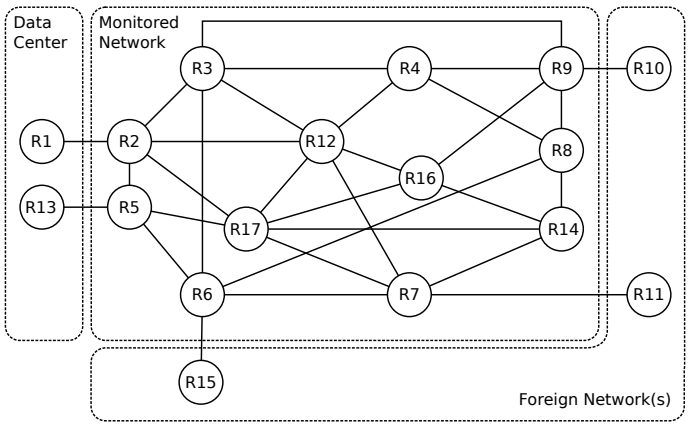


Figure 2.1.3.: Exemplary network structure consisting of several routers of the monitored network, data center routers ($R1, R13$) and edge routers ($R10, R11, R15$) connecting foreign networks.

Considering a nationwide telecommunications network (Figure 2.1.4) with several high capacity outbound and inbound links (er_x). A failure of one of those links would cause various automatic re-routing actions within the monitored network. The re-routing results in a different utilization of the other outbound/inbound links to other networks as well as of the links within the monitored network. Those utilization changes result in various alerts and warnings that can flood the monitoring system making it hard for the network operations personal to identify other, non related problems which require their attention. If the related alerts and warnings would automatically be linked to their root cause, the flooding due to a major link failure could be prevented.

It is the aim of this scenario to detect such major link failures and to determine the relation of other failures and warnings to any detected major link failure in order to allow for the mentioned grouping.

The scenario shares the same general two step processing as the other scenarios, in this case by first detecting a major link failure as indicator for the situation and then to correlate other incidents with the failure:

1. Detect major link failures by specifically monitoring links with high transfer capacity (e.g. er_1 and er_2 in Figure 2.1.4). Based on a detected failure of a major link f, e.g. $f = er_1$, determine if the traffic was re-routed to other high capacity connections by determining if there is a matching traffic increase shortly after the link failed. When such a traffic increase was found, on one or more major links, consider them as probable rerouting destinations RD, e.g. $RD = \{er_2\}$.

2. Once the failing link f and the new destinations for the traffic RD have been determined, trace the traffic changes from both, f and RD back through the network in a similar way as described for the DoS traffic tracing from Scenario 2. For example from the failing link $f = er_1$, the path could be $P_f = r_{23} \rightarrow r_{15} \rightarrow r_{14} \rightarrow r_{11}$ were

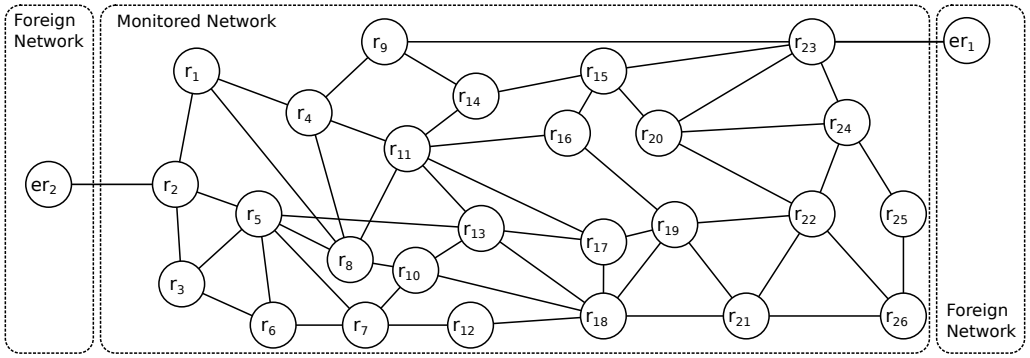


Figure 2.1.4.: Exemplary network structure consisting of several routers (r_x) forming the monitored network together with two interconnects to foreign networks through two edge routers (er_x).

it might get too dispersed to be distinguishable from normal traffic fluctuations. Similarly for the rerouting destination $RD = \{er_2\}$, the path might be traced as $P_{RD_{er_2}} = r_2 \rightarrow r_5 \rightarrow r_8$ until it gets too dispersed. Based on the traced paths P_f and $P_{RD_{er_2}}$, the zones which are likely to be affected by the rerouting are known. This allows to link other alerts and warnings regarding traffic changes with a machining amount of change, to the failing alert for the link f.

2.2. Definition of the General Type of Processing Shared by the Scenarios

Based on the discussed scenarios and the outlined analytical processing required by them, a generalized description of the type of processing that is done for the given cases can be derived. Based on this generalized description, the processing type is defined in a formal way as the foundation for the later design of a suitable processing model.

2.2.1. Characteristics Derived from Scenarios

Based on the scenario descriptions, several characteristics of the general scenario type can be derived.

2.2.1.1. Possible Situation Indication Requirements

For the cloud *detection* discussed in Scenario 1, Part 1, the monitoring for drops in the energy production needs to take place for *all monitored* solar panel installations in parallel and thus parallel access to potentially all measurement event streams has to be possible with the used processing system. For Scenario 3 from the telecommunications area, the possible situation detection also requires several links to be monitored in order to detect a possible failure. Depending on the network size there can be various links that have to be monitored in parallel resulting also in a possibly large number of event streams required for

the Possible Situation Indication. However, the required streams will normally not account for all available streams. Similarly, for Scenario 2, not all links need to be monitored for a DoS attack. However, for a larger network the number of links to continuously monitor can become fairly large. Based on this, the following two general characteristics can be defined:

SC1: *Possible Situations need to be identified* in a huge amount of streaming data, potentially in all available data streams.

SC2: The Possible Situation *Indication needs to be rapid also for large amounts of streaming data* (near real-time).

2.2.1.2. Situation-Specific Analysis Requirements

Based on the available event streams, the *situation-specific analysis* for Scenario 1 needs to correlate energy production information from solar panels in geographical proximity to each other to verify that a cloud was detected and to determine its border. Therefore, the detection requires *flexible access to multiple event streams* specific to the current possible cloud to verify that a cloud was actually detected. Further the processing needs information on the geographical location of the monitored solar panels to select other solar panels that should have been affected to analyze their data streams. Therefore, *meta information* like the geographical location of a data streams source is required.

For the verification of a DoS attack and to begin the tracing back to its entry point into the network, the processing requires access to a set of streams specific to the indicated possible DoS attack. However, the set of needed streams is for the verification of the attack and the first tracing step limited to event streams from routers neighboring the routers which detected the possible attack. In order to select the required routers and their event streams, access to background knowledge on the topology of the monitored network is needed. In a similar way, the verification of a traffic shifting due to a major link failure in Scenario 3 only requires access to a limited set of alternative transit links to which the traffic could have switched in order to verify that a switch occurred. In order to determine this set, access to topological information on the monitored network is needed.

Based on the discussions, the following three general characteristics for the situation-specific analysis can be defined:

SC3: Found *Possible Situations require their verification and an in-depth situation-specific analysis* based on *streaming data* and static *background knowledge.*

SC4: The situation-specific analysis only requires access to a subset of the overall set of event streams.

SC5: The *part of the stream data and background knowledge* needed for the situation-specific analysis *can not be determined before the situation has been detected.*

Aside from the so far discussed verification of a possible situation by the situation-specific analysis, all given scenarios also require some in-depth analysis of a situation in order to determine further properties of the now verified situation.

For the *tracking* of a detected cloud in Scenario 1, Part 2, the following requirements can be derived:

- The monitoring of a cloud's movement or size change is based on the monitoring of the geographically surrounding solar panel installations as well as of the panels that are currently shaded. Thus, again data streams are needed, based on the geographical location of the corresponding solar panels.

- The tracking requires access to the data streams from the panels in the geographical vicinity of the shaded area. However, due to the cloud's movement, the area from which the data streams are needed is not limited to a „small" geographical area. Instead, *the set data streams required for the tracking changes over time* and can over time include a large part of the available data streams but consisted at any given point in time of a much smaller subset required for the current position of the tracked cloud.

In a similar way the traffic tracing of the discussed Scenarios 2 and 3 requires access to a subset of the available event streams which changes over time, as the analysis process traces the traffic in a step by step manner through the network.

Based on the discussions, the following characteristics can be defined in addition for the situation-specific analysis:

SC6: The part of the stream and background knowledge needed during the analysis changes based on interim analysis results.

2.2.2. Formal Definition

Based on the derived characteristics, the required type of processing is formally specified in this section as the foundation for the later processing model design and to clearly point out the challenges addressed by this work.

The following formalization defines the processing of a situation as set of stream processing functions $sa_{...}$ which are generated during runtime based on an indication event for a possible situation (i) or based on interim results from an investigated situation ($ir_{...}$). To allow the generation of these processing functions, a builder function ($Builder$) is declared that is used to define the actual stream processing functions during runtime. Furthermore, in order to detect possible situations, a possible situation indication function (SI) is declared, which generates indication events when it detects potential situations. An overview of the formalization is given in Figure 2.2.1.

The discussed scenarios require the processing of measurement events e which consist of the actual event content, e.g. a measurement value, the time of the event's occurrence as well as the source of the event.

Definition 2.1 (Events and Event Streams)**.** An event can be defined as follows:

$$e := (t, s, c) \text{ with } \begin{cases} t & time \\ s & source \\ c & content \end{cases}$$

As the events need to be processed as an ordered continuous unbounded stream ϕ_s for example originating from an event source s, this stream can be defined as[3]:

$$\phi_s := (e_1, e_2, ...) \quad \text{with } \forall i \in \mathbb{N} : \pi_{time}(e_i) \leq \pi_{time}(e_{i+1}) \tag{2.1}$$

Definition 2.2 (The set of all available Event Streams)**.** Further the set of all available event streams Φ can be defined as:

$$\Phi := \{\phi_{s_x}, ...\}_{x \in \mathbb{N}} \tag{2.2}$$

The definition of the processing given here is focused on its usage of streaming data as this poses the central challenges. Aside from the access to streaming data, the processing also needs access to semi static background knowledge where its variability and exact type is not considered in this definition to reduce the complexity. As such, the available background knowledge is referenced as an element of the set of all possible sets of background knowledge K but is only later defined by Definition 4.1.

$$K := \text{Set of all available background knowledge sets}$$

Based on these basic definitions, the processing type can be defined based on the scenario characteristics SC1 to SC6 as follows (Figure 2.2.1):

Following SC1, possible situations need to be identified by a possible situation indication function SI within possibly all available event streams Φ which can be defined as follows:

For SC1:

The event stream of all possible situation indications:	$I := (i_1, i_2, ...)$
The Possible Situation Indication Function:	$SI : \mathcal{P}(\Phi) \rightarrow I$

For example for the cloud tracking scenario, the possible situation indication function SI would monitor the measurement data streams of all relevant solar panels. Whenever a solar panel would significantly reduce its power production, the indication function would

[3]With π_x as the projection to x.

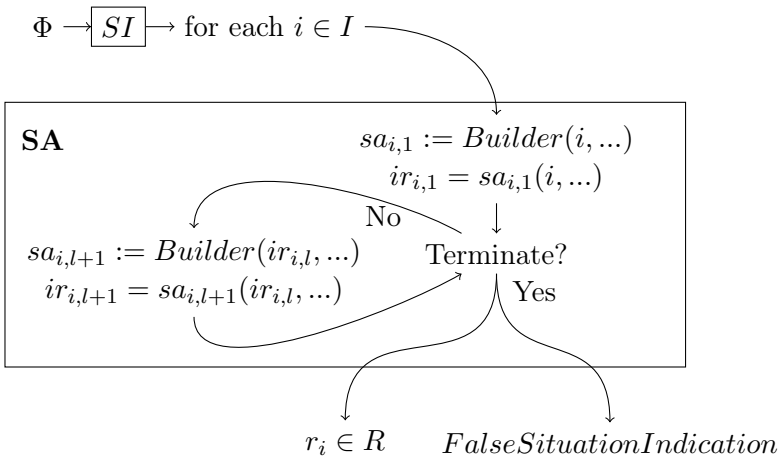


Figure 2.2.1.: Overview over the formalized processing type.

produce an indication event i. Over time this results in the stream of possible situation indications I.

For SC2: The possible situation indication function SI needs to be able to cope with huge amounts of streaming data in order to find possible situation candidates. As such, the function needs to feature the rapid processing of the measurement data in order to provide scalability with regard to growing numbers of event streams $|\Phi|$ that need to be monitored.

Based on a produced possible situation indication $i \in I$, an in-depth situation analysis has to take place by a situation analysis function SA. As the analysis is done for each raised indication $i \in I$ separately, *the following processing definitions take place once for each* $i \in I$*:*

The situation analysis determines if the indicated possible situation i is a valid situation or a false situation. If it is a valid situation, an in-depth analysis is done resulting in a situation analysis result r_i. Further the situation analysis is allowed to produce intermittent results IR_i which are discussed later. In summary SA can be declared as follows:

For SC3:

$$\begin{aligned} &\text{All possible results of the situation analysis:} && R_i := IR_i \cup \{r_i, FalseSituation\} \\ &\text{The situation analysis function:} && SA : (I \cup R_i) \times \mathcal{P}(\Phi) \times K \to R_i \end{aligned}$$

In contrast to the possible situation indication function SI, the situation analysis function SA implements an in-depth analysis process, which will, in most cases, take more resources per processed event then the situation indication processing. However, as the situation analysis is focused on a *single* possible situation i, it only requires access to event data streams which are required for the analysis of this possible situation. For example for an

indicated possible cloud, the situation analysis will only look at a number of data streams originating from the solar panels within a certain geographical proximity to the indicated possible situation.

For SC4:

This subset of the available event streams is provided by a selection function which determines the subset of all the available event streams Φ based on the currently analyzed possible situation $i \in I$ with the information available in the background knowledge $\in K$.

This reduction of the event stream count allows to lower the requirements for scalability of the analysis function that needed to hold for the possible situation indication function SI.

Similar to the event streams, the situation analysis function needs a subset of the available background knowledge on the monitored system which can also be selected by a selection function based on the analyzed possible situation indication $i \in I$ or interim results IR_i. As however the formal definition of the problem is focused on the data stream processing, the selection functions are not defined here but are considered a part of the situation analysis functions discussed in the following paragraphs.

As the situation analysis is an ongoing process which needs to account for changes in the analyzed possible situation, the situation analysis itself can not be considered as one static function but as a *set of n_i situation analysis iteration functions* $\{sa_{i,1}, ..., sa_{i,n_i}\}$ *specific to the possible situation i* which are used to analyze it. With each iteration step, the processing can result in one of the following:

- a false situation: $FalseSituation$
- interim results, one per iteration except for the final iteration: $IR_i = \{ir_{i,1}, ..., ir_{i,n_i-1}\}$
- or the final situation analysis result: r_i

Thus, the initially defined set of possible results R_i of each of the iterations may also contain interim results:

$$R_i := IR_i \cup \{r_i, FalseSituation\}$$

Furthermore, the set of possible inputs to each iteration can be defined as the set $I \cup IR_i$ as a $FalseSituation$ result or the final result r_i terminate the situation processing while the initial iteration has to handle the possible situation indication i as its input. Based on this, the iteration processing functions can be declared as a set of functions F_i:

$$F_i := \{sa_{i,1}, ..., sa_{i,n_i}\}_{l=2...n_i} \text{ where } \begin{cases} sa_{i,1}: & I \times \mathcal{P}(\Phi) \times K \to R_i \\ sa_{i,l}: & IR_i \times \mathcal{P}(\Phi) \times K \to R_i \end{cases} \tag{2.3}$$

Each of the iteration processing functions can generate an interim result or one of the two terminal results (r_i or $FalseSituation$). If for example the first analysis function $sa_{i,1}$ provides the interim result $ir_{i,1}$. This interim result is then used as input for the consequent analysis function $sa_{i,2}$ which could then provide the next interim result $ir_{i,2}$ or one of the terminal results. This process continues until a terminal result was produced.

As each iteration processing function needs to be derived from the initial possible situation indication $i \in I$ or previous processing results $ir_{i,l} \in IR_i$ in combination with the background knowledge, a function $Builder$ needs to be declared, which defines each of the functions in the set F_i based on the indication event or the previous processing results:

$$\begin{aligned} Builder : (I \cup IR_i) \times K &\rightarrow F_i \\ (i \in I, \blacksquare) &\longmapsto sa_{i,1} \\ (ir_{i,l} \in IR_i, \blacksquare) &\longmapsto sa_{i,l+1} \end{aligned} \tag{2.4}$$

The overall situation analysis function can now be defined as a recursive function that uses the $Builder$ to define the current iteration specific function $sa_{i,l}$ which is then used in the iteration l to do the actual analysis. The recursive processing ends when the result of an iteration is either a $FalseSituation$ or the final analysis result r_i. In summary this results in the following definition of the situation analysis function SA :

For SC5: and SC6:

$$SA(x,p,k) = \begin{cases} SA(\ sa_{i,1}(x,p,k)\ ,p,k) & \text{if } x \in I \wedge x = i \ \text{ where } \ sa_{i,1} = Builder(x,k) \\ SA(\ sa_{i,l+1}(x,p,k)\ ,p,k) & \text{if } x \in IR_i \wedge x = ir_{i,l} \ \text{ where } \ sa_{i,l+1} = Builder(x,k) \\ r_i & \text{if } x = r_i \\ FalseSituation & \text{if } x = FalseSituiation \end{cases}$$

Example Situation Analysis

For the example of the cloud tracking scenario, the first iteration function $sa_{i,1}$ would be defined by the $Builder$ so that it compares the energy production by solar panels in the geographical neighborhood to the indication i in order to verify that the indication really concerned a cloud. If a real cloud is detected, this cloud will change its position over time requiring the adaptation of the first processing function $sa_{i,1}$ and thus the definition of a new iteration function $sa_{i,2}$ by the $Builder$ based on the interim results $ir_{i,1}$ produced by $sa_{i,1}$. The new function $sa_{i,2}$ would then be aimed at the processing of the data streams from the new location of the cloud. After possibly many iterations, the cloud will leave the

monitored area. The last iteration will thus result in a final processing result r_i describing this fact and end the processing.

As shown by the formal definition of the situation-aware adaptive processing, a processing system supporting such a processing mode has to provide mechanisms to initiate a processing focused on an indicated possible situation during run-time. Further it needs to provide the means to derive the iteration specific processing functions from previous results in order to adapt the processing according to the needs of the investigated situation.

2.3. Requirements Towards an Event Stream Processing System

In order to evaluate the suitability of event stream processing systems towards the motivating scenarios for this work, this section defines three high level requirements towards event stream processing systems based on the scenario requirements and their formalization. The three requirements are as follows:

RQ1: *Support to set up a situation indication processing that can handle large amounts of streaming data.*
A processing system needs to be capable of deducing stream processing statements based on a provided processing description in combination with system specific background information in such a way that situations can be detected in a possibly huge set of event streams.

RQ2: *Support to deduce and initiate an analysis processing for a detected situation, where the analysis processing is specific for the detected situation.*
A processing system needs to be able to deduce and deploy *situation-specific* processing statements *during run-time* from a provided processing description in combination with the

a) current processing state, especially the detected situation

b) a situation-aware high level processing model

c) background information on the monitored system.

RQ3: *Support to handle changes of a currently investigated situation that require the adaptation of the processing of an ongoing situation-specific analysis based on interim results.*
A processing system needs to be capable of *deducing the need to adapt a situation-specific processing* during run-time based on the same information as RQ2. It further needs the capability to deduce the required changes and the capability to apply the changes to the running processing system.

Based on the requirements defined in this chapter, the next chapter gives an overview of the current state of the art in the area of this work and discusses the shortcomings of current approaches and systems regarding the requirements defined here.

3. State of the Art

Contents

In order to point out the gap within the current state of the art of event processing systems, this section first gives an overview over Event Processing and its origins and then continues with a discussion of general classes of event stream processing systems. Then, further approaches which are directly related to the problem considered here, are discussed and rated with regards to the requirements. The section concludes with a summary of the discussed suitability of the discussed classes and systems to point out the gap in the current state of the art.

3.1. Overview Event Processing

Event processing consists of methods and tools to filter, transform, and detect patterns in events, in order to react to changing conditions, typically under some time constraints [CEvA11]. In event processing, an *Event* is defined as "Anything that happens, or is contemplated as happening" [DL11, p.5]. As events occur over time, they form a linearly ordered unbound sequence which is called an *Event Stream* [DL11] (Definition 2.1 on page 22). The processing of such streams is called *Event Stream Processing* (*ESP*).

From a conceptual point of view, event processing applications are formed by one or more *Event Processing Agent*s (EPA) sometimes also called event processing components. EPAs are entities that process event objects. As such they act as event consumers and event producers. Several EPAs are typically interlinked by communication channels to form

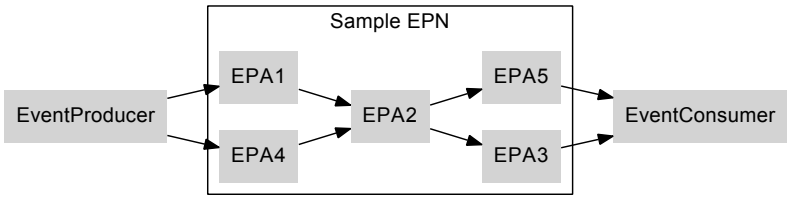


Figure 3.1.1.: Sample Event Processing Network

an *Event Processing Network* (EPN) [DL11, Luc01] (Figure 3.1.1). The communication between the EPAs is provided by an Event Notification Service.

As discussed by Mühl et al. [MFP06], event-based computing follows a contrasting approach to the conventional request/reply mode of interaction and inherently decouples the components from each other. In event based systems, the components communicate by generating or receiving event notifications. An interested component subscribes to the event notifications that it is interested in. In turn, components that generate event notifications publish them so that they can be received by the subscribers. The Event Notification Service is used to mediate the communication among the components (Section 3.2.1).

Event processing applications can be found in various areas such as in monitoring systems, ranging from network monitoring [CJ09] to business activity monitoring [Luc01], in the processing of sensor network information like for example in RFID based logistics applications [WS09a] or traffic management systems like outlined in [Dun09]. Furthermore, event processing systems are used in Enterprise Application Integration for a flexible and scalable integration of the various enterprise systems [BD10]. Moreover, event processing is used for various analytical applications such as the classical stock trading use case or for the detection of customer behavior in web shops as shown in [WSGL11].

The following sections introduce some general concepts in the area of event processing as the foundation for the later discussions of system classes. Further discussions regarding the history and the different variations of event processing can be found in [Luc07b, MFP06].

3.1.1. Active Database Systems: ECA-Rules

One of the origins of event processing lies in the development of Active Database Management Systems (ADBMS) in the early 1990's, which extend the classical database management systems with the capability to allow the active reaction to certain changes within the database. As such, ADBMS systems introduced the support for triggers which allow the definition of such reactions. The definition is based on *E*vent *C*ondition *A*ction (ECA) Rules, which are triggered based on the occurrence of the specified event and execute the specified action if the specified condition is fulfilled [Con96].

To overcome the limitation of using ECA rules only in connection with a particular

database management system the paradigm of event processing has been proposed in the late 1990's to separate the rule processing from the database system in order to allow the usage of ECA-Rules across several databases as well as other kinds of information sources [GKBF98]. One of the first systems following this unbundling approach was C^2offein [KL98, Kos99, KK98], which provided ECA-Rule Processing in a separated Activity Service using CORBA as communication layer.

3.1.2. Event Driven Architectures (EDA)

Event Driven Architectures (EDA) have been proposed in the last years as an architectural paradigm for event-based applications [BD10, Luc01]. In an EDA the central control flow is realized based on event based communication between components. Thus, the processing of events is the central architectural concept. As a result, a very flexible control flow is possible. Furthermore, the application's components are loosely coupled with each other, easing the extension of the application and the reuse of existing components.

A special form of an EDA is the Staged Event Driven Architecture (SEDA) as „an architecture for handling the massive concurrency and load conditioning demands of busy Internet services" [Wel02, p. 1]. SEDA was developed by Matt Welsh from Harvard University in 2002 [Wel02], and since its publication it received a great deal of attention. It has been adopted by several well established server applications like Apache Camel [Apad], Apache ActiveMQ [Apab], Mule ESB [Mul14] or Apache Service Mix [Apag]. The central idea behind SEDA lies in the combination of event based programming combined with thread handling to construct applications in multiple stages. The stages are interconnected by event queues and each stage fulfills a distinct task in the processing of a request of the service. The explicit interconnection between the layers via queues allows for a flexible load management by thresholding or filtering of the event queues (cp. [Wel02] p.4), thus allowing the construction of scalable server systems that need to handle huge amounts of client connections in parallel.

Furthermore, the combination of Service Oriented Architectures (SOA) with event processing to create an Event Driven SOA (ED-SOA), sometimes called SOA 2.0, gained some attention [Luc, Mar06, LC08]. The aim to extend the concepts of a SOA with the capability to react dynamically to occurring events is expected to open a new set of application areas like, for example, the realization of dynamic business processes.

3.1.3. Complex Event Processing

Complex Event Processing (CEP) was introduced by David Luckham in his book *The Power of Events* [Luc01]. Luckham considers "CEP as the logical and obvious next step in the development of event processing" [Luc07a]. CEP has its origin in the Discrete Event Simulation and is focused on the event processing itself in order to generate new higher level events. CEP therefore allows for a step wise abstraction and reduction of the event

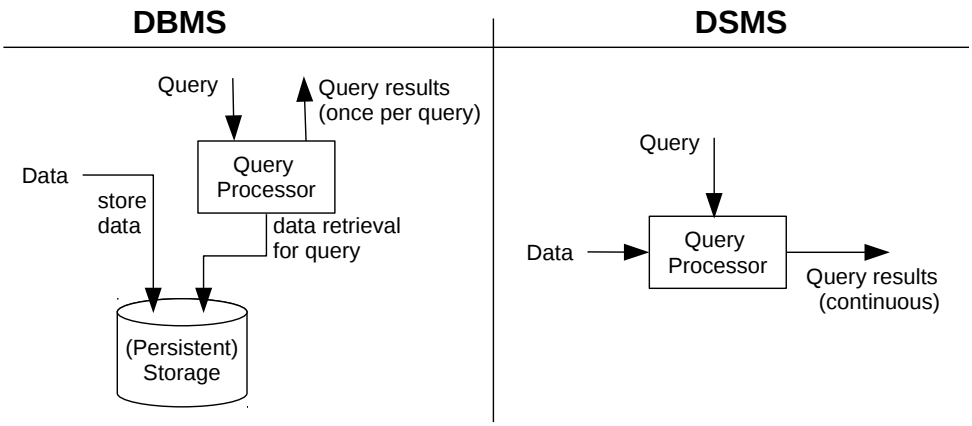


Figure 3.1.2.: Information Processing in a Database Management System and a Data Stream Management System

load. The central idea behind CEP is to provide the means to handle the increasing flood of events that modern information systems are faced with.

The CEP concept builds on the aggregation and abstraction of numerous low level events into a new high level event based on their temporal, spatial or causal relation, thus providing an abstraction from the underlying events. For example, several credit card transactions, having their origin in different countries, could be correlated to a new complex event that represents a possible credit card fraught. In a CEP system such abstraction can take place in many stages further reducing the event amounts after each step while increasing the level of abstraction with each step. As such, the abstraction from the low level events does not only allow the reduction of the event counts, it also allows the event processing on a higher level than the incoming event stream has. Aside from the step wise abstraction, the capability to handle vast quantities of events in a timely fashion can be considered as an essential aspect of CEP (cp. [CEvA11, CM12, EN11]).

Based on this, CEP shares several properties with Event Stream Processing. In fact, the Event Processing Technical Society considers CEP and ESP only as *conceptual classifications* which "can be useful in delineating philosophies of event processing and intended applications, but do not specify precisely the underlying capabilities of event processing engines." [DL11, p.14].

3.1.4. Event Stream Processing

Based on the need to handle continuous streams of event data, the concepts of ADBMS where extended towards better handling of streaming data. These efforts resulted in the first *Data Stream Management Systems* (DSMS), like for example Aurora and Borealis [ACC+03b, AAB+05], STREAM [ABB+04] or TelegraphCQ [CCD+03b].

In a DSMS, the data processing is built for the handling of continuous unbounded streams

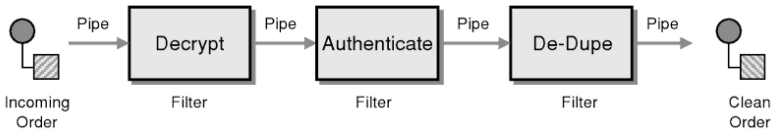


Figure 3.1.3.: Pipes and Filters Architecture example (Source: [HW12, p. 71]).

of data. A DSMS typically processes incoming data in its main memory as the data arrives, without storing it to some persistent second level storage (Figure 3.1.2). For this kind of processing, it *continuously* evaluates the incoming data against the specified continuous queries. This stands in contrast to the processing concept known from a typical DBMS where the data is stored on a persistent second level storage when it arrives. If a client enters a query into the DBMS, the required data for the query is retrieved from the second level storage in order to evaluate the query. Once the evaluation of the retrieved data snapshot is done, the results are returned to the query issuer and no further processing of the query is done, even if additional data, relevant for the query arrives at the DSMS.

Due to the different information processing approach of a DSMS, such systems only have a limited view on the available event data, as they at best only have access to the parts of the event stream that was already received. As such, new mechanisms to query such data streams needed to be derived. Several methods to deal with the limited view on the data have been developed in the form of window based queries. Here a sliding window of the event stream, based on event counts or time frames, is taken into account for the processing, including statements on the non-existence of certain events within the current time window. These efforts resulted in several new query languages which are discussed in Section 3.2.3 and 3.3.

Even though not directly related to the Event Stream Processing discussed here, it should be noted, that approaches exist to adapt the general MapReduce processing concept to support stream data processing. Map Reduce is a common mechanism for processing big amounts of data introduced by Google in 2004 [DG04]. However, it is focused on a batch operating mode, splitting a huge amount of *stored data* into chunks suitable for processing in parallel by a number of machines, and gathering the results from each of the processing nodes to provide the aggregated overall result. Due to this store and process approach it is not particularly suited for stream data processing. However, first approaches to combine the split and merge processing semantic to data stream processing exist like for example [ABM10, LY08]. The approaches are very similar to the Distributed Event Stream Processing Middlewares discussed in Section 3.2.2 and thus suffer from the same limitations.

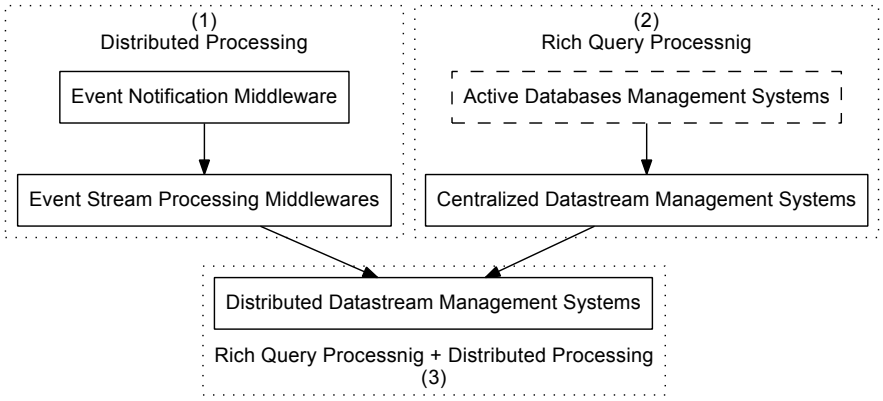


Figure 3.2.1.: Hierarchy of the discussed Event Stream Processing classes.

3.1.5. Pipes and Filters

Another related architectural concept is the Pipes and Filters architectural pattern. The pattern "provides a structure for systems that process a stream of data" [BMR+96, p. 53]. It defines the data processing steps as filters which are interconnected by pipes which transport the data stream from one filter to the next (Figure 3.1.3). It thereby divides "larger processing tasks into a sequence of smaller independent processing steps" [HW12, p. 71] .

This structure is also referred to as a processing pipeline and can be found in various application areas ranging from video or image processing for computer vision applications to the Unix pipeline mechanism as well as enterprise integration tasks. In the context of enterprise application integration, systems like for example Apache Camel [Apac] support the definition of such processing pipelines where the pipes can be provided by messaging systems like Apache ActiveMQ [Apaa]. For Unix/Linux systems, the Pipes and Filters principle is also a very common processing mechanism. Here multiple programs (filters) can be connected via pipes, based on their input and output streams to form a processing pipeline. This functionality can also be combined with shell scripts thus allowing for example for the automation of system administration tasks.

3.2. Classes of Event Stream Processing Systems

The following sub-sections give a brief overview over different classes of Event Stream Processing systems and discuss their suitability with regard to the requirements defined in the previous section. Further some systems and their capabilities, as relevant for this work are presented for each class. The discussion follows the structure shown in Figure 3.2.1, starting from the distributed processing perspective (1), it discusses Event Notification Middlewares as the underlying communication mechanism and continues with Event Stream Processing Middlewares which introduce distributed basic event stream processing.

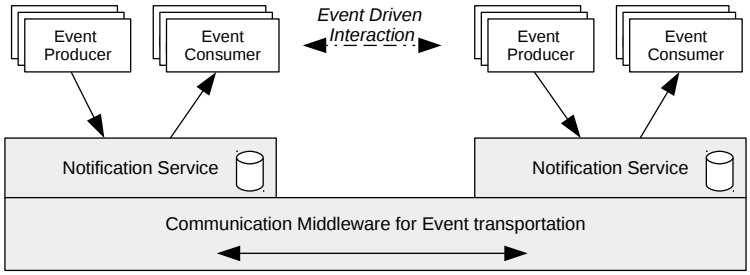


Figure 3.2.2.: Event Notification Service (based on [MFP06, Fig. 2.1])

The discussion then moves on to the other perspective, systems that provide rich query processing (2) in the form of centralized DSMS but do not consider distributed processing. The discussion then concludes in the combination of both perspectives (3) by looking at distributed DSMS that feature-rich query processing together with distributed processing.

3.2.1. Event Notification Middlewares

Distributed Event Processing requires a suitable communication middleware in order to convey events among processing nodes and external event sources and sinks. Within the scope of event processing systems such a middleware is typically called Notification Service [MFP06] (Figure 3.2.2). A Notification Service realizes a publish/subscribe-pattern thus providing the facilities to subscribe to relevant events, to consume events based on such a subscription and to publish new events. It is further responsible for the transfer of the events among distributed system components. Depending on the concrete incarnation the Notification Service also may take care of some form of Quality of Service guarantees like guaranteed delivery [EFGK03].

There are various communication Middlewares available for building distributed event processing systems which range from standards-based systems like the CORBA Event and Notification service or the Java Messaging Service API (JMS) over commercial and non-commercial Message Oriented Middlewares like for example TIBCO Rendezvous, IBM WebSpereMQ or Apache ActiveMQ to a variety of research prototypes like Gryphon [SBC+98], SIENA [CRW01], JEDI [CDNF01], REBECA [PGS+10], the event routing system proposed by Wishnie et. al. [WS09b] or the OM4SPACE Activity Service [SAKG14].

Aside from those notification systems and API's there are several cloud-based communication services that allow to easily convey messages or events without taking care about aspects like dynamic scalability of the communication middleware based on current utilization. An example for such services is the Amazon Simple Notification Service[Amab].

Event Notification Middlewares have no own support for the event stream processing. Therefore, they are on their own not suitable for the problem considered in this work.

3.2.2. Event Stream Processing Middlewares

Distributed Event Stream Processing Middlewares extend the concept of Event Notification Middlewares by adding the capability to handle the scheduling of given event stream processing tasks in addition to the basic event communication. As such they allow for the *automatic* distribution of an event stream processing application if it obeys the programming model defined by the middleware. Typically, Event Stream Processing Middlewares however do not provide the query or rule processing engines necessary to realize the actual event processing within each processing task nor do they provide a query language to specify the required processing. Thus, Event Stream Processing Middlewares leave the actual stream data processing to the application developer to realize.

Examples of such systems are Apache Storm [sto], Apache S4 [Apaj], Apache Spark Streaming [Apah], Apache Samza [Apaf], Google MillWeel [ABB+13] or Muppet [LLP+12].

The Apache Storm processing platform [sto] was originally developed by BackType which was later acquired by Twitter. Storm provides a framework for the creation of distributed stream processing applications and claims to hide the complexities that come with aspects like guaranteed message processing, robust process management, fault detection and automatic reassignment, efficient message passing, local mode and distributed mode [Mar]. It defines a concept of worker nodes and master nodes. Each worker node runs a supervisor that can start and stop worker processes based on assigned work. The system uses Apache Zookeeper [Apai] for the coordination among the nodes.

In order to use Storm to create an ESP application, the processing logic needs to be defined as a *directed acyclic graph* of processing functions which in Storm is called a *topology*:

```
TopologyBuilder builder = new TopologyBuilder();                                    1
builder.setSpout("measurements", new MeasurementSourceSpout(), 10);                2
builder.setBolt("average", new MyBolt(), 3).shuffleGrouping("measurements");        3
```

The processing functions, in Storm called Bolts, need to utilize a storm-specific API to consume streaming data and to produce their processing results as a new stream:

```
public class MyBolt extends BaseRichBolt {                                          1
  @Override public void prepare(Map conf, TopologyContext context,                  2
      OutputCollectorBase collector) {...}
                                                                                    3
  @Override public void declareOutputFields(OutputFieldsDeclarer declarer) {...}    4
                                                                                    5
  @Override public void execute(Tuple input) {                                      6
    /* manually implement stream processing for the given input tupel */            7
  }                                                                                 8
}                                                                                   9
```

Storm is then responsible for deploying the functions on multiple worker nodes, possibly as a parallel processing system and to provide the required communication among the worker nodes. For this task Storm uses a scheduler to assign the work to the various processing nodes. Extensions of this mechanism towards a dynamic scheduling in Storm exist like for

example [ABQ13] but they are focused on a generic assignment of processing loads based on currently available capacity not on a processing model supporting situation awareness as required in this work.

Apache S4 [NRNK10, Apaj], initially released by Yahoo in 2010, provides similar functionality as Storm. It also provides a framework for the development of distributed stream processing applications. S4 defines its processing elements based on an actors model, which provides the semantic definition of the encapsulation and allows for concurrent deployment [NRNK10].

Apache Spark is a general processing engine for large-scale data. Spark Streaming extends this platform to support a streaming mode in order to build stream processing applications. Similar to the other two platforms, processing tasks need to be implemented based on a Spark specific API. Both systems share the same limitation as Storm regarding the requirements of this work.

With regard to a dynamic task assignment and redistribution, some research in the direction of using cloud resources for the realization of such flexible stream processing exists, like [GJPPMV10, SAG+09, KKP11]. Also, from the commercial area, approaches exist like for example Amazon Kinesis [Amaa] which provides a communication middleware for event streams and a static stream partitioning in order to distribute the processing load to several parallel processing nodes. Even though these systems feature a dynamic task assignment and scalability based on current load situations, they are not aimed at the required situation-aware adaptiveness and have no own support for it.

In summary Distributed Event Stream Processing Middlewares support the automatic distribution of event processing components but do not support any query or rule languages for the actual event processing in those components. Instead, the processing components need to be implemented specifically for the given application based on a programming interface provided by and specific to the used middleware. In conclusion, systems within this class are on their own not suitable for a situation-aware adaptive processing as they only partly fulfill the given requirements discussed in Table 3.2.1.

3.2.3. Centralized Data Stream Management Systems

The system classes discussed in the previous sections focus on the distributed processing and the required communication and task distribution, they do not provide specification languages for the actual stream processing logic. This section will thus cover the class of systems which support such languages. This section will first cover *centralized systems* which *do not support distributed operation* on their own while the next section will extend this discussion towards systems that support query languages in combination with distributed operations. Both classes of systems are typically called Data Stream Management Systems (DSMS) in relation to the term Database Management System (DBMS),

RQ1 Support to set up a situation indication processing that can handle large amounts of streaming data	**Partial Support** If a suitable set of processing functions are implemented, systems of this class can distribute the functions across several processing nodes in order to handle the load. However, they do not feature mechanisms to specify the processing task in a suitable high level rule or query language thus requiring a manual implementation of the actual stream processing.
RQ2 Support to deduce and initiate an analysis processing for a detected situation, where the analysis processing is specific for the detected situation	**Partial Support** Event Stream Processing Middlewares have no direct support to initiate situation-specific processing tasks based on indicated situation candidates. However, they typically have the capability to deploy new / additional processing tasks if provided by a third party system which could realize a situation-aware adaptive processing on top of such a middleware.
RQ3 Support to handle changes of a currently investigated situation that require the adaptation of the processing of an ongoing situation-specific analysis based on interim results	**Partial Support** Event Stream Processing Middlewares have no support to automatically adapt an ongoing situation-specific processing based on interim processing results. However, a third party system could request the necessary adaptations similar to RQ2.

Table 3.2.1.: Suitability of Event Stream Processing Middlewares towards the defined requirements.

however the term DSMS does not clearly distinguish between centralized and distributed systems.

The following paragraphs discuss several centralized DSMS without focusing on their used processing languages. A more detailed discussion on the languages is given separately in section 3.3 followed by a more detailed discussion of adaptive DSMS optimization mechanisms.

The STanford stREam datA Manager (STREAM) [ABB+04] is a centralized DSMS developed by the Stanford University. STREAM supports a declarative SQL based Continuous Query Language (CQL) [ABW06] which has a special focus on a clear semantic. TelegaphCQ [CCD+03b, KCC+03, CCD+03a, Tel] is a prototype from UC Berkeley, based on PostgreSQL which extends PostgreSQL's SQL dialect for the handling of data streams. Furthermore, a well-established system is Esper [Esp, BV07], a centralized open-source Event Stream Processing Engine. Esper supports the Event Query Language (EQL), an SQL-like query language to specify continuous Queries. In contrast to the STREAM and TelegraphCQ, Esper is not based on a DBMS but is instead intended to be directly integrated into Java or .Net applications.

Similar to the static query optimizations known from relational database management systems, such query optimizations are also possible for stream query languages. However, for stream query languages with limitations due to the limited information on the data streams as their properties are, in contrast to typical database relations, not known in

RQ1 Support to set up a situation indication processing that can handle large amounts of streaming data	**No Support** As centralized Data Stream Management Systems do not feature any form of distributed parallel processing, they can on their own not be used to realize a possible situation indication processing for large amounts of streaming data.
RQ2 Support to deduce and initiate an analysis processing for a detected situation, where the analysis processing is specific for the detected situation	**No Support** Centralized Data Stream Management Systems have no direct support to initiate situation-specific processing tasks based on indicated situation candidates as they do not feature a higher level processing model above the deployed queries or rules.
RQ3 Support to handle changes of a currently investigated situation that require the adaptation of the processing of an ongoing situation-specific analysis based on interim results	**Partial Support** Centralized Data Stream Management Systems can support the adaptation of the deployed rules during run-time. However, the engines do not support a mechanism to trigger such changes on their own based on a high level processing model.

Table 3.2.2.: Suitability of Centralized Data Stream Management Systems towards the defined requirements.

advance. A discussion of such optimization approaches is given in Section 3.4.1.

Furthermore, rule processing engines have been extended to support streaming data like for example JBoss Drools Fusion [dro]. Drools Fusion is a centralized DSMS which extends the JBoss Drools rule engine with functionality specific for streaming time series data like the support for window-based operations. In contrast to the SQL-like query languages, Drools Fusion uses an Event Condition Action (ECA) based processing specification as known from Active Database Systems.

Another approach is used by Aurora. Aurora [ACC+03a, ACC+03b] is a Data Stream Management System developed by the Brandeis University, Brown University and M.I.T to overcome the limited applicability of DBMS for monitoring applications. Aurora supports the specification of continuous queries as loop-free directed data flow graphs of stream processing operators based on Aurora's query algebra SQuAl (Stream Query Algebra).

Aurora supports dynamic optimizations of the processing graphs in order to meet definable quality of service requirements. The general optimization operations used are load shedding and query network optimization which is based on the commutative property of the operators. Aurora starts with an un-optimized processing network and optimizes it during run-time based on gathered statistics like the average operation execution cost or an operation's selectivity [ACC+03b].

Even though centralized DSMS provide a broad set of functionality for the realization of a *centralized* event stream processing system, they are not capable of providing a situation-aware adaptive processing as required by this work as discussed in Table 3.2.2.

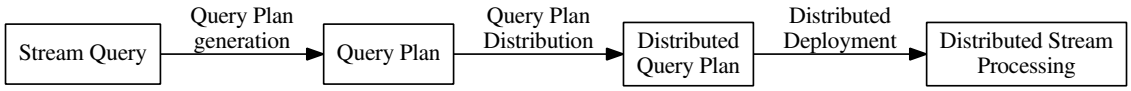


Figure 3.2.3.: Outline of the Stream Query distribution process implemented by a distributed DSMS.

3.2.4. Distributed Data Stream Management Systems

While the previous section focused on centralized DSMS, this section discusses *distributed* DSMS to determine their suitability towards the defined requirements.

There are two general goals in the distribution of event processing applications that can be distinguished:

1. Distribution *as a result* of a distributed environment that requires the integration of distributed and possibly heterogeneous event sources and sinks into an overall event processing system.

2. Distribution of an event processing system *as the enabler* to achieve scalability or reliability through the usage of multiple machines.

The following discussion is focused on the second aim of distributed event processing as one of the central goals is to achieve the scalability required by the scenarios for situation-specific adaptive processing tasks.

Distributed DSMS combine Event Stream Processing Middlewares (Section 3.2.2) with a query language and query processing engine as discussed for centralized DSMS (Section 3.2.3). Thus, distributed stream processing applications can be created by specifying the relevant queries and the DSMS takes care of deriving a suitable *distributed* query plan and setting up the distributed processing system that implements this plan (Figure 3.2.3).

This combination also allows for a query aware optimization of the distributed deployment which is not possible for normal Event Stream Processing Middlewares due to their limited knowledge of the actual event processing tasks. In addition to the query optimizations used in centralized DSMS, the distribution decisions can be optimized based on the expected load of the different processing operators in order to provide a distributed setup that is capable of handling the required load while minimizing over provisioning of processing resources. Such mechanisms are discussed in Section 3.4.1.

Furthermore, distributed DSMS typically employ mechanisms to handle load fluctuations by adapting their resource provisioning or by employing load shedding mechanisms.

Early systems in this area are the extensions of centralized DSMS like Aurora*, an approach towards the distribution of Aurora. In turn, Borealis [AAB+05, Bor] is based on Aurora* [XZH05] and Medusa, a federated stream processing system [CBB+03, SZS+03]. Another example is the distribution of the SASE processing system by Wang et al. [WY10]

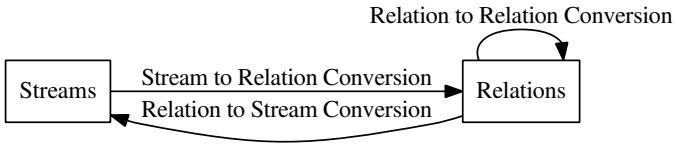


Figure 3.3.1.: Stream relation conversions in CQL (based on [ABW06])

or NiagaraCQ [CDTW00]. As a recent development from 2012, StreamCloud is "An Elastic Parallel-Distributed Stream Processing Engine" [Gul12, GJPPM+12, GJPPMV10] based on Borealis, which aims to address the limited scalability of state of the art DSMS. The approach of the system is to split continuous queries into sub-queries which are assigned to different servers in order to distribute the processing load. It also supports dynamic load balancing mechanisms to avoid over- and under-provisioning of the available processing resources while handling fluctuations in the data streams. The dynamic load balancing of StreamCloud is based on the average CPU utilization of the processing nodes and does not take into account changes of the continuous query itself.

On the other hand PIPES, the Public Infrastructure for Processing and Exploring Streams [Krä07, KS04], from the University of Marburg is aimed at providing fundamental building blocks required to implement a distributed data stream management system but is on its own not a ready-made system.

There are also several systems which focus on event processing in sensor networks like for example Cougar [YG02] or HiFi [CEF+04]. As these systems try to solve a separate class of problems, they are not considered in this section.

In general the systems discussed here are capable of setting up distributed stream processing based on given queries and to optimize the system to provide the required processing capacity and response times. However, the systems have no mechanisms to adapt deployed stream queries based on detected situations and situation changes as they have no knowledge of the overall analytical task that deployed a given stream query. As such the systems are not capable of realizing a situation-aware adaptive processing as required for this work. Table 3.2.3 discusses the suitability of the system class with regard to the defined requirements.

3.3. Event Processing Languages

The two event stream processing classes, Centralized DSMS and Distributed DSMS allow the specification of the event stream processing function based on Event (Stream) Processing Languages. Section 3.2.3 already mentioned several languages while discussing the class of Centralized DSMS. This section extends this by discussing three language categories based on the way the processing functions are specified:

RQ1 Support to set up a situation indication processing that can handle large amounts of streaming data	**Full Support** The properties of Stream Processing Middlewares for RQ1 apply. Further support to specify the processing in a high level query language and to optimize the processing based on query graph information provides the capability to set up a distributed processing system capable of handling large amount of streaming data on multiple machines.
RQ2 Support to deduce and initiate an analysis processing for a detected situation, where the analysis processing is specific for the detected situation	**No Support** The Properties of Centralized DSMS for RQ2 apply. Therefore, no support for a situation-aware model or a similar mechanism is given which allows for the automatic generation of situation-aware processing statements.
RQ3 Support to handle changes of a currently investigated situation that require the adaptation of the processing of an ongoing situation-specific analysis based on interim results	**Partial Support** Systems of this class can support dynamic query plan optimization but the adaptations are not backed by a high level situation-aware adaptive processing model, thus also preventing automatic adaptations of the queries itself. Further the properties of centralized DSMS for RQ3 apply.

Table 3.2.3.: Suitability of distributed Data Stream Management Systems towards the defined requirements.

Query Based

In query based languages, the event streams are processed in a similar form as relational data in a relational data based by specifying a for example an SQL like query. For example the centralized DSMS STREAM supports a declarative SQL based Continuous Query Language (CQL) [ABW06]. CQL follows the approach to map windows of streaming data to relations in order to process them together with other relations and to map the result back to a data stream (Figure 3.3.1). For this purpose CQL extends SQL with a *windowing operator* which allows the conversion of the data stream into a relation. Further it provides operators to convert back from a relation to a data stream. The following query shows these two operators:

```
SELECT ISTREAM(AVG(PackageCount) FROM Traffic [Range 1 Hour])
```

The windowing is specified at the end of the query as a duration based window of one hour. Based on this value, the query calculates an average value and provides the results as a stream by using the `ISTREAM` operator.

Flow Graph Based

In flow based languages, the stream processing functions are specified as an acyclic directed graph where the nodes are represented by processing operators and the edges resemble the flow of events from one operator to the next. The resulting structure is similar to the processing structure defined by Event Processing Networks (EPN) (Section 3.1). An example of a system that is based on this approach is the Stream

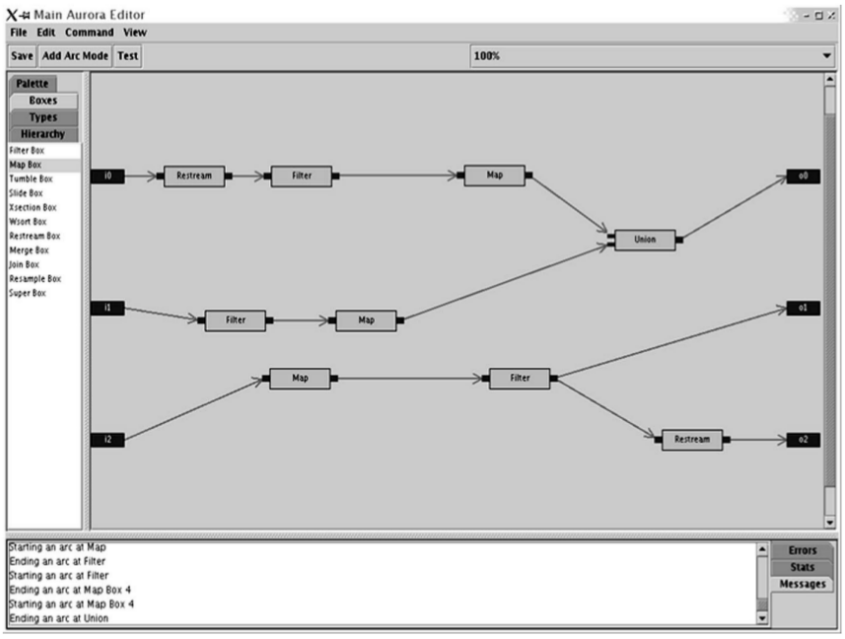


Figure 3.3.2.: Aurora System Model showing the event stream processing logic as a directed acyclic graph (Source [ACC+03b, Fig. 18])

Query Algebra (SQuAL) defined by the centralized DSMS Aurora (Figure 3.3.2). Aurora supports operations like for example "Filter", "Map" or "Join" [CBB+03]. Based on these operators the processing graph is defined and executed by Aurora.

Rule Based

In rule based languages, the processing function is defined by processing rules, which are triggered by the appearance of certain events where a rule defines a certain reaction to the event. A typical rule type are Event Condition Action (ECA) rules as known from Active Database Systems (Section 3.1.1) where the action is to be executed if the specified event occurred and the given condition holds. An example of such an ECA-Rule processing language is the rule language of JBoss Drools Fusion. The following listing gives an example for a rule in the Drools rule language:

```
rule "Sample"                                                    1
when                                                             2
  $value : SingleMeasurement($value:doubleValue)                 3
  eval ( $value > 10 )                                           4
then                                                             5
  System.out.println("Event detected")                          6
end                                                              7
```

In this example, rule is triggered by a "SingleMeasurement" event where the contained value needs to be 10 or more. In this case the specified action is executed which prints the message "Event detected".

There are also several rule languages for ontologies like for example SWRL [W3C04],

SPIN Rules [KHI11] or Jena Rules [Apae]. Even though these rule languages are not designed for usage with streaming data, several approaches exist to allow the processing of streaming data in formats like RDF with the typical query languages used for ontologies like SPAQL to combine the stream processing with for example RDF based background knowledge like [AFRS11, BGJ08, BBC+09, CCG10].

Aside from the examples discussed here, several other languages exist like Stanford Rapide [Luc96, Luc01], SASE [WDR06], NEEL [LRD+11], IBM's Stream Processing Language (SPL) [HAG+09], which is the successor of SPADE [GAW+08] in IBM's stream processing systems, Siddhi [SGLN+11] or StreamSQL [Sof].

Even though the presented languages provide a rich set of functionality for the specification of the event stream processing logic, all these **languages lack support for an automatic domain-specific adaptation** of their queries, flow graphs or rules as they don't support a higher level processing model such as the adaptive situation-aware processing model presented here.

In addition to the discussed languages, approaches for the generation of event processing functions exist like the generation framework presented by Magrid et al. [MOB+08]**.** Other approaches like for example iCEP [MCT14] aim at the generation of event processing functions based on historic event data. These approaches are, however, aimed at easing the general process of developing event-processing applications and don't consider the situation-aware adaptive behavior needed here.

3.4. Approaches and Systems related to Situation-Aware Adaptive Processing

Aside from the discussed event stream processing classes, some approaches exist that have some similarity with or a closer relation to the situation-aware adaptive processing and its challenges addressed by this work. These approaches are discussed in the following subsections.

3.4.1. Adaptive DSMS Optimization Mechanisms

As the discussion of DSMS revealed, the central missing capability with regard to the defined requirements lies in the support of suitable adaptation mechanisms. In order to shed some more light on this limitation, this section is focused on adaptive optimization mechanisms for DSMS and the general limitations of these approaches with regard to the defined requirements.

In a distributed DSMS, the processing functions are distributed across a set of machines in order to distribute the processing load. To define a suitable distribution, DSMS usually

utilize some form of heuristic based on statistical information on the streams. However, the typically available statistical information on the stream (e.g. number of events or the selectivity of an operator) is not directly available for data streams as only a possibly small portion of the data stream is known. To cope with this, approaches, which gather statistical information over time, exist alongside with systems that allow the specification of the expected stream behavior in addition to the continuous query itself.

For continuous queries deployed over a long time, the initially specified statistical properties can also become incorrect over time. As a result, a DSMS needs to provide means for the adaptation of a deployed distributed query plan in order to handle wrong initial assumptions or over time changing conditions.

A typical approach to detect the need for an adaptation is to monitor the continuous query execution in order to gather statistics about its resource usage and processing delays. Based on the gathered statistics, an adaptation need can be determined and the query plan and its distribution be adapted. For the adaptation, distributed DSMS implement several adaptation operations. Among others such operations are the reordering of operators in the query graph, operator splitting in order to allow for parallel execution (Fission / Partitioning), joining of two separate operators (Fusion) to avoid overhead caused by remote communication or the migration of the operators to other machines [HSS+14].

Examples of systems that employ statistics-based optimizations of their query graphs are StreaMon [BW04, Bab05], Aurora* and Borealis [AAB+05, XZH05]. Aurora* for example starts with a very crude data stream partitioning in the beginning and tries to optimize its processing system over time based on the gathered resource usage statistics [CBB+03]. Furthermore, various approaches have been proposed which employ adaptive optimizations to handle load fluctuations by utilizing the dynamic resource availability of cloud computing offerings like [GJPPMV10, SAG+09, KKP11] in order to scale on demand.

Other approaches introduce new operators which allow for an adaptive partitioning or query plan execution. For example the Flux operator [SHCF03] allows for a dynamic partitioning of state-full operators during run-time in order to flexibly scale a stream processing system to handle varying processing loads. An even more flexible version is the Eddy operator which was proposed by Avnur et al. [AH00]. The Eddy operator allows for a continuous reordering of the operators in a query plan during run-time. The approach considers the query plan as a task where tuples need to be routed through the operators. Within this model, the Eddy operator allows a per-tuple routing decision thus allowing for a fine-grained control of the actual query graph at run-time. An application of the Eddy operator to continuous queries exists with the Continuous Adaptive Continuous Queries over Streams (CACQ) [MSHR02].

Further approaches towards an efficient migration from one query to an new query exist like the Constraint-exploiting Adaptive Processing Engine (CAPE) [RDS+04] and its distributed version D-CAPE [SLJR05] or [ZRH04]. An overview of such adaptive query

RQ1 Support to set up a situation indication processing that can handle large amounts of streaming data	**Full Support** When considering adaptive optimization mechanisms for distributed DSMS, such mechanisms can be used to correctly setup and maintain a distributed Possible Situation Indication processing for large amounts of streaming data.
RQ2 Support to deduce and initiate an analysis processing for a detected situation, where the analysis processing is specific for the detected situation	**No Support** The presented mechanisms for adaptive DSMS query optimizations have no support for the initiation of situation-specific processing tasks as part of their adaptation process.
RQ3 Support to handle changes of a currently investigated situation that require the adaptation of the processing of an ongoing situation-specific analysis based on interim results	**Partial Support** As the discussed adaptive optimization mechanisms do not support a higher level situation-aware adaptive processing model, they are not able to provide adaptations based on such a model. However, they provide the means to adapt an ongoing processing which could be used in combination with an external system which triggers the situation based adaptations.

Table 3.4.1.: Suitability of adaptive DSMS optimization mechanisms with regard to the defined requirements.

processing mechanisms and approaches is given in [HFC+00, BB05].

As discussed, approaches for the adaptive optimization of DSMS follow the general mechanism to gather statistics of the actual processing load during run-time and allow shifting load among different operators to correct previous assumptions. As such they can be used in the context of the Possible Situation Indication. However, the approaches have no support to adapt a processing system based on a higher level situation-aware model. Thus, the mechanisms do not support the creation of situation-specific processing tasks as part of their adaptation process. Therefore such approaches are on their own not suitable for the given problem (Table 3.4.1).

3.4.2. Process-oriented Event Model

The Process-oriented Event Model (PoEM) [PSPP14] is an approach focused on industry applications and allows the modeling of possible states of a monitored entity. Based on the state it further allows the definition of state specific actions. To ensure that a failure of a triggered action does not go unnoticed it supports a special escalation mechanism as part of the event reaction work-flow. If an actor was notified based on an event but fails to respond, the event can be escalated to a higher level system to be handled there.

The PoEM approach supports the specification of a higher level processing work-flow which also considers the modeling of actions based on the current (monitored entity) state. However, the approach has no special support for the specification of automatic adaptations of the processing based on a detected situation in order to provide a targeted processing for the situation (Table 3.4.2). Further, the approach is not aimed at the detection of possible

RQ1 Support to set up a situation indication processing that can handle large amounts of streaming data	**No Support** The approach is not aimed at handling large amounts of streaming data for a possible situation indication.
RQ2 Support to deduce and initiate an analysis processing for a detected situation, where the analysis processing is specific for the detected situation	**Partial Support** PoEM supports the modeling of system states in order to define state specific reactions. This mechanism can be considered as a form of situation-specific processing. However, no in-depth processing based on a situation is intended.
RQ3 Support to handle changes of a currently investigated situation that require the adaptation of the processing of an ongoing situation-specific analysis based on interim results	**No Support** The approach has no support for the adaptation of a running analysis task based on intermittent results.

Table 3.4.2.: Suitability of the Process-oriented Event Model with regard to the defined requirements.

situations in huge amounts of streaming data originating from various monitored entities.

3.4.3. Hybrid Static and Dynamic Optimization

Soulé et al. propose a hybrid optimization approach [SGA+13] that combines dynamic optimization with static optimizations. Their approach assumes that several parts of a stream processing application can be optimized statically and only few links in between the static parts can benefit from dynamic optimization as only their output varies. In order to achieve this, they subdivide a stream application into coarse-grained sub-graphs. The sub-graphs are interlinked by so called dynamic rate boundaries. Within a sub-graph the system applies static optimizations.

This approach could in parts be applied to the given problem of a situation-aware adaptive processing (Table 3.4.3) by considering the possible situation indication processing as an initial static sub-graph which is followed by some dynamic processing part. However, in contrast to the situation-aware adaptive processing approach presented in this work, the dynamic optimization steps are not capable of acting in a situation-aware fashion.

3.4.4. Data Stream Processing for Moving Range Queries

Specialized approaches for handling *time varying queries* in event processing systems also exist for mobile applications under the term Moving Range Queries. One approach from this area, that is particularly relevant for this work, is the Mobility-Aware Complex Event Processing (MCEP) [OKR+14a, KORR12, OKR+14b], a CEP system optimized for moving range queries in mobile application areas.

RQ1 Support to set up a situation indication processing that can handle large amounts of streaming data	**Supported** Assuming that the mechanism is applied to a distributed DSMS, the possible situation processing could be separated into a sub graph which could be optimized separately from the remaining processing to gain the required scalability.
RQ2 Support to deduce and initiate an analysis processing for a detected situation, where the analysis processing is specific for the detected situation	**No Support** The discussed approach is not aimed at providing mechanisms to derive a situation-specific processing based on raised possible situation indications.
RQ3 Support to handle changes of a currently investigated situation that require the adaptation of the processing of an ongoing situation-specific analysis based on interim results	**No Support** The discussed approach is not aimed at adapting a continuous query during run-time or to deduce the need to do so, based on interim processing results.

Table 3.4.3.: Suitability of the "Hybrid Static & Dynamic Scheduling" approach with regard to the discussed requirements.

MCEP supports the *automatic adaptation of the event processing* based on the geographical location of the processing result consumer. The adaptation is based on a special *dynamic interest query* which considers a *focal point* of a consumer and a *spatial interest*. To adapt the query, the consumer has to provide discrete location updates. Based on the updates, old queries are stopped and new queries started based on the change in the spatial interest of the consumer.

The approach has some similarities (Table 3.4.4) to the situation-aware adaptive processing designed here, with regard to the definition of a changeable focal point and a surrounding area of interest. However, their approach is focused on spatial relationships only whereas the problem set addressed here requires a more generic support for the definition of the focal *area* and the related area of interest. Further MCEP has no specific support for adapting the dynamic interest queries based on previous results of the query. Instead, external location updates are required by the MCEP process. Furthermore, it does not have specific support for separating the initial parts of a processing from the dynamically focused part in order to support the required scalability for the indication.

3.5. Situation-Aware Processing Outside of the Event Processing Scope

The general concept of a situation-aware processing can be found in various areas like for example in robotics [WG96] or cyber-security [GKS14] where it is approached from the corresponding problem domain in order to develop solutions *specific for the given domain or detail problem*. These approaches may or may not use Event Stream Processing (ESP) as part of their solution. However, for these approaches ESP is only the means to achieve

RQ1 Support to set up a situation indication processing that can handle large amounts of streaming data	**No Support** The MCEP system has no support for the detection of situations in large amounts of streaming data as the trigger for further situation-specific processing.
RQ2 Support to deduce and initiate an analysis processing for a detected situation, where the analysis processing is specific for the detected situation	**Partial Support** The MCEP system can derive and set up a location specific moving range queries which could be considered as a very specialized form of a situation-specific processing. However, it has no general support to set up a situation-specific processing based on a possible situation trigger. Further it has no support for a general situation-aware adaptive processing model but instead features a specialized model for location based queries.
RQ3 Support to handle changes of a currently investigated situation that require the adaptation of the processing of an ongoing situation-specific analysis based on interim results	**Partial Support** MCEP allows for the run-time adaptation of deployed queries based on a geographic area of interest which is assumed to change over time. The approach has however no support to derive adaptations in a generalized form in combination with background knowledge. Further the limitations from RQ2 regarding the absence of a general situation-aware processing model apply.

Table 3.4.4.: Suitability of Moving Range Queries with regard to the defined Requirements.

their overall goal but they do not aim at extending Event Stream Processing with the generalized capability of situation-aware adaptive processing. As such the model designed here can be used to ease the development of such domain specific applications by reducing the effort of utilizing an ESP system for situation-aware adaptive processing.

3.6. Summary and Conclusions

An overview of the evaluation results of the discussions from Sections 3.2 and 3.4 is given in Table 3.6.1. The discussions revealed that the *existing classes of event-stream processing systems and approaches are not capable of a situation-aware adaptive event stream processing.*

Even though distributed DSMSs provide a lot of the functionality that is needed, they lack support for the automatic deduction of query adaptations based on a higher level model as they have no own support for such a model. On the other hand, Moving Range Query systems like MCEP provide such an adaptation mechanism. However, their focus lies on the area of geographical information systems making the solutions specific to this area without support for a more general processing model. Furthermore, they are not aimed at supporting the detection of possible situations in huge amounts of streaming data. Approaches that allow for a certain adaptiveness of DSMS are focused on the optimization of the deployed queries to handle fluctuations in the incoming data stream sizes or incorrect assumptions during the initial query plan optimization. They, however, also do not feature a higher level model that allows the adaptation of the queries themselves based on the

	Requirement RQ1	**Requirement RQ2**	**Requirement RQ3**
	Support to set up a situation indication processing that can handle large amounts of streaming data	Support to initiate an analysis process for a detected situation where the analysis processing is specific for the detected situation	Support to handle changes in an investigated situation that require the adaptation of the processing of an ongoing situation-specific analysis based on interim results
General Classes of ESP Systems			
Event Stream Processing Middlewares	Partial Support	Partial Support	Partial Support
Centralized Data Stream Management Systems	No Support	No Support	Partial Support
Distributed Data Stream Management Systems	Full Support	No Support	Partial Support
Comparable Approaches			
Adaptive DSMS Optimizations	Full Support	No Support	Partial Support
Process-oriented Event Model	No Support	Partial Support	No Support
Hybrid Optimization	Supported	No Support	No Support
Moving Range Queries	No Support	Partial Support	Partial Support

Table 3.6.1.: Overview of the suitability of the considered event stream processing classes regarding the requirements from Section 2.3.

detection of a possible situation or a changing situation.

4. Processing Model

Contents

This work follows the approach of separating the detection of a possible situation from its verification and situation-specific analysis. In order to allow for such a situation-aware processing, this chapter defines a generalized processing model which consists of several well defined phases which follow the abstract processing approach proposed by this work (Section 1.4). This chapter will discuss each processing phase in detail and specify what the purpose and context of each single processing phase is in order to define it. Furthermore, the design decisions and the reasoning behind the process will be presented.

As an introduction for these discussions, the following section first gives a quick overview of the resulting processing model in high level terms before continuing with the design discussions in Section 4.2 ff.

4.1. Overview of the Processing Model

Following the general approach proposed by this work, the processing model is divided into three main phases (Figure 4.1.1):

Phase 1: Possible Situation Indication *(Defined in Section 4.3)*

In this phase, the detection of *possible* Situations within the possibly big amounts of streaming data that is received from a monitored system is realized. The focus here lies on the *rapid* detection of possible situations. To

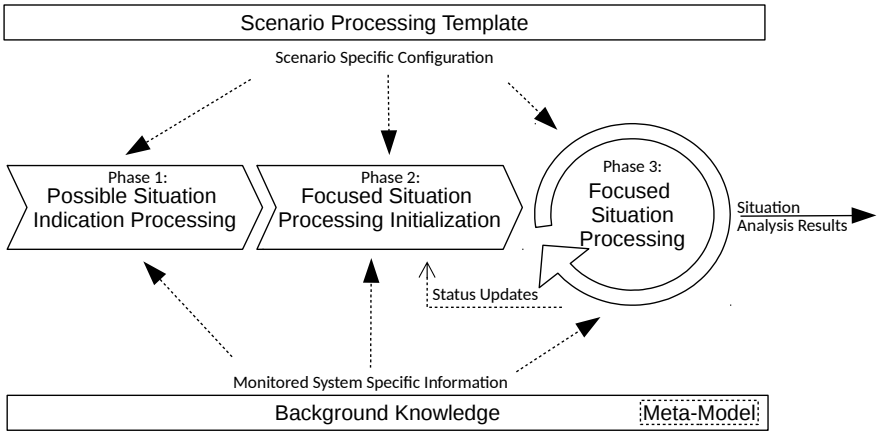


Figure 4.1.1.: Simplified view of the processing model.

provide the required scalability in this phase, the processing may produce a possibly great number of false positives when an additional verification of possible situations would require additional resources.

Phase 2: Focused Situation Processing Initialization *(Defined in Section 4.5)*
In this phase, previously indicated possible situations are classified in order to determine if a specially focused situation processing is required or if the indicated possible situation is or was already investigated. If the situation is not yet analyzed, a specialized processing task is initialized. In order to allow for this classification to take place, the Phase 2 processing uses status information provided by the Phase 3 processing which allows the correlation of the received indication with finished and ongoing Focused Situation Processing instances.

Phase 3: Focused Situation Processing *(Defined in Section 4.6)*
In the third phase, an indicated possible situation is investigated in further detail to (a) determine whether the indication regards a real situation or is a false positive and (b) to investigate the situation in detail as required for a given application scenario. Therefore, during this phase, several Focused Situation Processing *Instances* are started where each instance follows one (possible) situation. The processing within such an instance results either in the notification of a false situation or situation-related analyses results which can be used by third party systems.

The three phases resemble the general processing flow while monitoring a system for possible situations. Aside from these three phases, an initialization phase is defined:

Phase 0: Possible Situation Indication Processing Initialization *(Defined in Section 4.4)*

The processing initialization phase is required in order to initiate the possible situation indication processing. This step produces the specific processing function for the system that is to be monitored. As part of this initialization, background knowledge is retrieved from a background knowledge repository and combined with the situation indication processing description from a Scenario Processing Template.

Further the processing model defines the following knowledge sources which exist outside of the defined phases:

Situation Processing Template *(Defined in Section 4.2.1)*

A Scenario Processing Template *describes* the necessary steps to detect, verify and analyze a situation. The description is given as a template that needs to be combined with the background knowledge on the actual system that is being monitored thus allowing to reuse templates for many systems in the same application domain.

Background Knowledge *(Defined in Section 4.2.2)*

The background knowledge provides information on the monitored system which can be used together with the Scenario Processing Templates to initialize and maintain the processing. Further the information is used during the situation-specific processing in Phase 3 to allow the use of additional knowledge that is not part of the event streams or the general Scenario Processing Template.

4.2. General Elements of the Processing Model

4.2.1. Scenario Processing Template

The Scenario Processing Template describes a relevant situation for the processing model by specifying configuration settings that parameterize the processing model for the detection and analysis of the specified kind of situation. In order to refer to these templates, let T be the set of all available Scenario Processing Templates and $\tau \in T$ one such template.

One major part of this specification is the definition of how the actual event stream processing has to take place. The stream processing is specified in the templates by the definition of two Stream Processing Builder functions. The builder functions are used together with additional background knowledge and possibly interim processing results to define an actual stream processing that shall take place to detect, verify or track a situation (see Section 4.4 and 4.6.2.5).

Aside from the builder functions, the template also specifies several other scenario-specific aspects for the processing model like the collision-handling between conflicting (possible) situations. The various configurations that are part of the template are specified

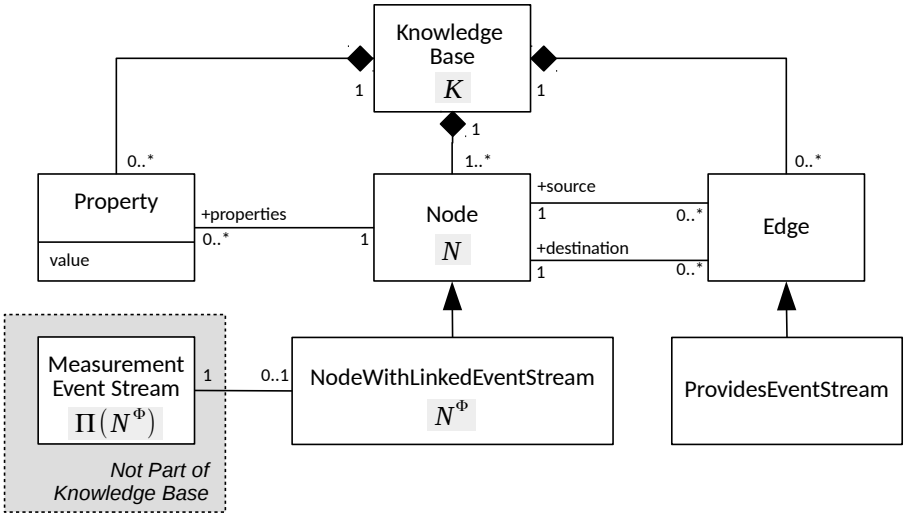


Figure 4.2.1.: Meta model of the background knowledge defining the basic concepts for the processing system including special nodes which have associated event streams.

in detail in the remainder of this chapter. The following chapter will then use these definitions in order to create a description language that is used to specify Scenario Processing Templates.

4.2.2. Background Knowledge

The processing model relies on the availability of suitable background knowledge on the monitored system which is provided by a background knowledge base. The knowledge base is defined as a directed graph of nodes and edges where properties can be assigned to a node as defined by the meta-model (M2) shown in Figure 4.2.1. Further the meta-model defines a specific type of node which has an event stream associated with it, the event stream itself is however not part of the knowledge base.

Based on this meta-model, a domain specific model (M1) can be defined for each application domain and based on it the concrete application specific instances (M1), as illustrated by the examples in Sub-Sections 4.2.2.1 and 4.2.2.2.

The background knowledge is considered as mostly static as it remains unchanged for large periods of time and is then updated in a bulk update outside of the scope of the model presented here.

For later definitions, the set N is defined as the set of all nodes contained in the current knowledge base K. Further the set N_Φ is defined as a subset of N which contains all nodes with associated event streams:

Definition 4.1 (Background Knowledge Base)**.**

K	as the set of all knowledge bases
N	as the set of nodes
N^{Φ}	as the subset of nodes ($N^{\Phi} \subseteq N$) with an associated event stream

Furthermore, a function Π needs to be defined, which provides a mapping from any given subset of nodes from the set N^{Φ} to the corresponding subset of event streams from the set Φ:

Definition 4.2 (Event Stream Selection Function)**.**

$$\Pi : \mathcal{P}(N^{\Phi}) \rightarrow \mathcal{P}(\Phi)$$

4.2.2.1. Example: Smart Grid Background Knowledge

Example 1

Figure 4.2.2 shows an example background knowledge base content from the Smart Grid domain suitable for the Cloud Tracking scenario. The depicted model contains instances on level M0 for one solar panel (*SG_Holten_1*) which provides a single measurement stream on its power production (*holten_1_pv_power*). Further the figure shows the model for the Smart Grid domain (Level M1) which defines elements like a *device*, a geographical location as well as a *pvPowerProduced* node type with associated event stream.

4.2.2.2. Example: Telecommunications Network Background Knowledge

Example 2

In a similar way as the Smart Grid example, Figure 4.2.3 gives an example from the telecommunications network monitoring as used by the corresponding scenarios. The example defines instances for two routers (*router1* and *router2*) each with one network interface. Both interfaces are interconnected by a communications link. Further, for each interface two nodes with event streams are defined for inbound and outbound traffic measurements. As in the previous example, the figure also contains the needed model on level M1 for the telecommunications domain which defines elements like a router, interfaces and links.

4.2.3. Focus Area and Locked Area

An essential task of the processing model is to manage the occurrence of potential and verified situations. Part of this process is the correlation of newly detected indications with already identified situations as well as the correlation of situations that, after some

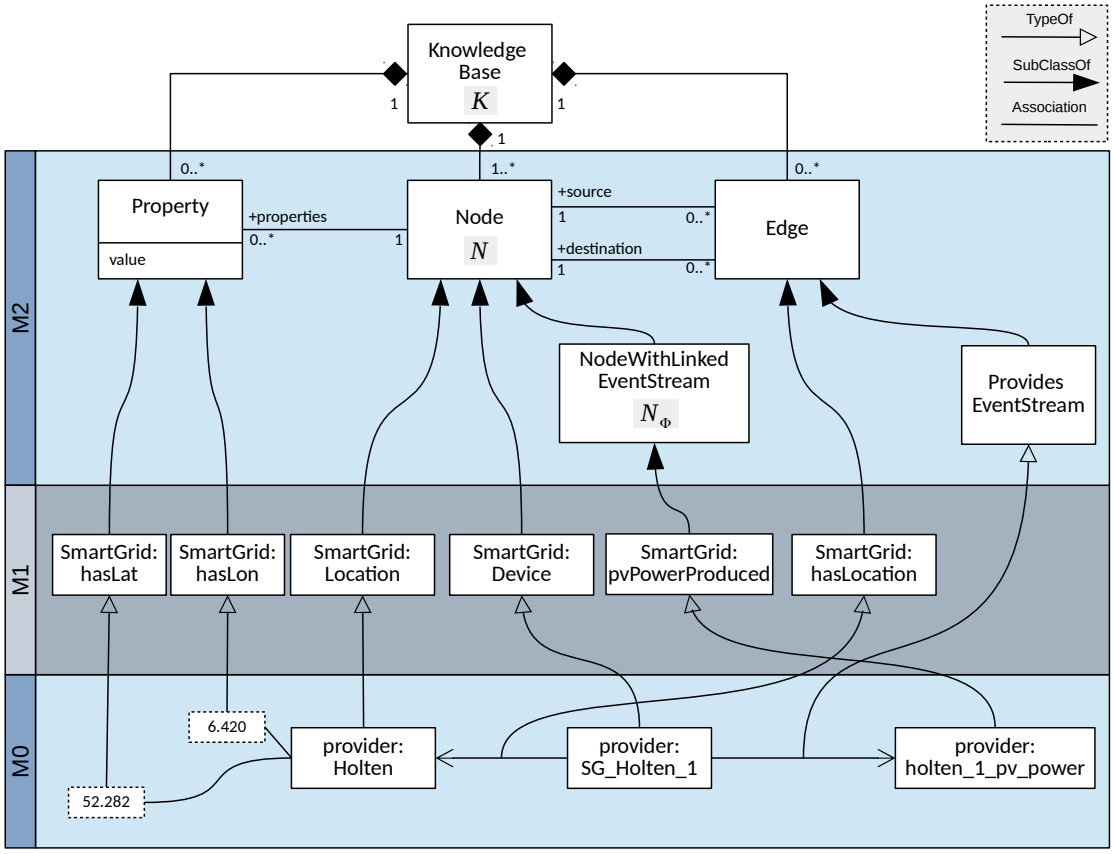


Figure 4.2.2.: Example Smart Grid background knowledge base contents.

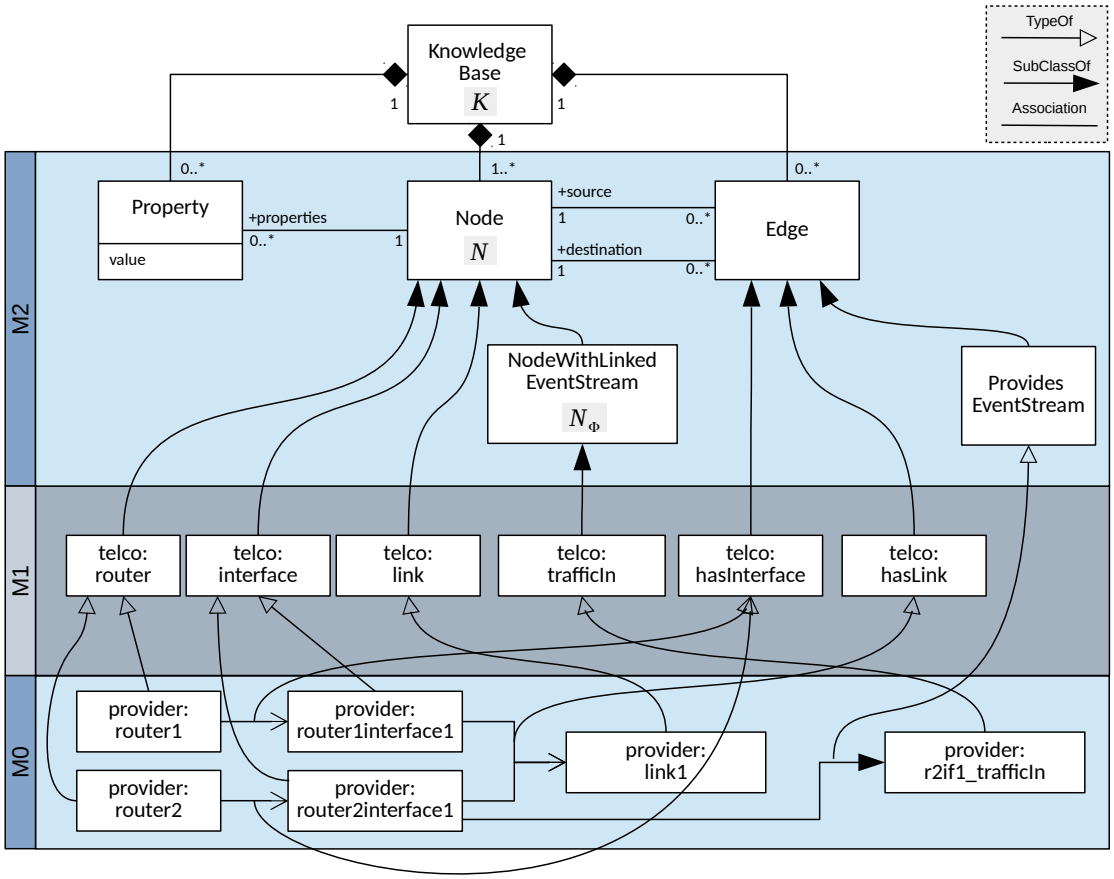


Figure 4.2.3.: Example Telecommunications Network background knowledge base contents.

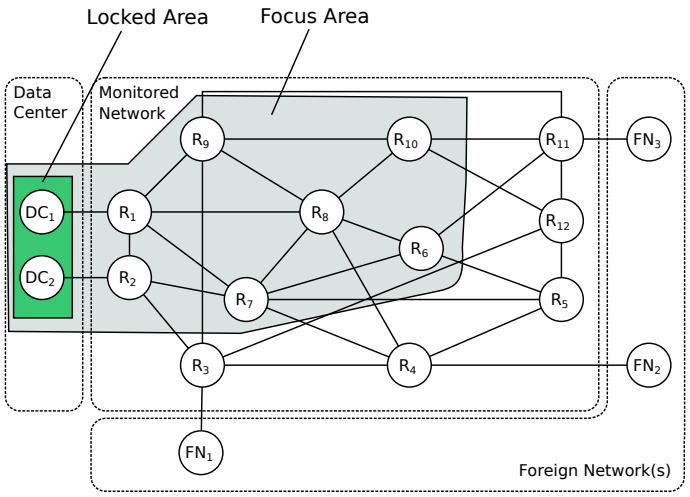


Figure 4.2.4.: Exemplary Locked Area and Focus Area for the DoS tracking scenario.

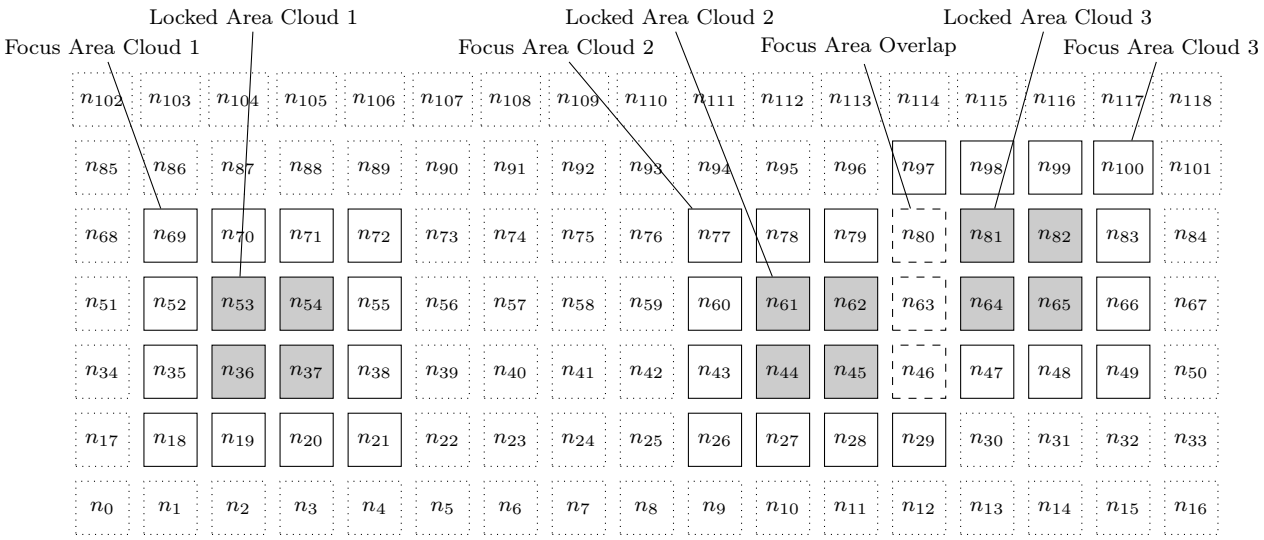


Figure 4.2.5.: Exemplary Locked Areas and Focus Areas for the cloud tracking scenario based on three clouds were the two clouds on the right hand side share parts of their Focus Area due to their close proximity.

analysis, turned out to be the same. To allow for this process to happen, the model needs a mechanism to keep track of the identity and scope of the handled (possible) situations. The mechanism used by the processing model is based on two different sets of nodes, the ***Focus Area*** and the ***Locked Area***, which are kept per (possible) situation. Both, the Focus Area and the Locked Area are determined in the beginning of a new Focused Situation Processing Instance based on a raised possible situation indication as discussed in Section 4.5.5 and can change over time once the Focused Situation Processing Instance is active in order to accommodate for changes in the situation as discussed in Section 4.6.2.8. The following paragraphs define the Locked Area and Focus Areas and explain their use based on the two example scenarios, cloud tracking and DoS tracing.

Definition 4.3 (Locked Area)**.** The **Locked Area** is a set of nodes that are uniquely affected by the investigated situation within a certain time frame and can thus be used to represent this situation's identity (Figure 4.2.6). As such any node from the set N can only be assigned to at most one Locked Area in a certain time frame and thus one Focused Situation Processing Instance at the time.
Later functions that determine a Locked Area, must thus adhere to the following conditions:

1. The Locked Area has to represent the investigated possible situation in order to allow the correlation of other possible situation indications originating from the investigated possible situation.

2. The Locked Area must only contain elements which are known to be part of *only* this (possible) situation.

Example 3

For example for the cloud tracking scenario, the Focused Situation Processing Instance locks all nodes that are verified to be affected by the tracked cloud (Figure 4.2.5) in a certain time frame. If any other Focused Situation Processing Instance would require the acquisition of this area in the same time frame, the second processing task would be tracing the same cloud and thus the same situation.

For the cloud tracking scenario, the Locked Area will change over time as it has to follow the tracked cloud. A fixed Locked Area per situation is also possible and is for example needed for the DoS scenario. Here the situation occurs on a set of routers which report the DoS attack and thus caused the initial Possible Situation Indication (Figure 4.2.4). These routers represent the DoS situation and are thus locked by the responsible Focused Situation Processing Instance.

Definition 4.4 (Focus Area)**.** The **Focus Area** is a set of nodes that is required by a running Focused Situation Processing Instance within a specific time frame as part of its current stream processing (Figure 4.2.6). As such the nodes within a Focus Area are not exclusively related to the corresponding situation and can also be part of other Focus Areas at the same time. The handling of the possible collisions is specific to the scenario (Section 4.5.4).
Later functions that determine a Focus Area, must adhere to the following conditions:

1. A Focus Area must contain any node that is required for the processing in the upcoming focused processing iteration.

2. A Focus Area may contain additional nodes which may or may not be relevant in order to guarantee that Condition 1 is kept.

Example 4

For example for the cloud tracking scenario, the Focus Area contains the nodes which are currently shaded by the cloud plus a number of nodes that geographically surround the affected area to allow the Focused Situation Processing Instance to determine the clouds borders and over time the movement of the cloud (Figure 4.2.5). As the Focus Area is not exclusive to one Focused Situation Processing Instance, two Focus Areas can overlap as shown for Cloud 2 and 3 in Figure 4.2.5.

The Focus Area used by the cloud tracking scenario will move over time from one set of nodes to another while the number of nodes contained in the Focus Area will roughly stay the same[a]. For other scenarios the Focus Area can however also grow as for example for the DoS tracing scenario. Here the Focus Area starts with the topologically surrounding nodes of the attacked routers. From there the Focus Area grows while the Focused Situation Processing Instance follows the path of the traffic

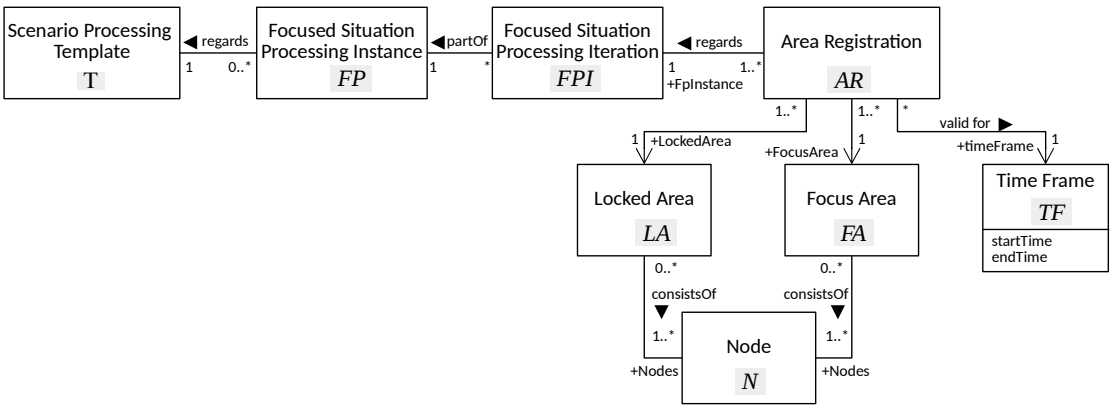


Figure 4.2.6.: An Area Registration, holds a Locked and Focused Area as well as the corresponding Focused Situation Processing Instance for a specified Time Frame.

through the network to its origin (Figure 4.2.4).

[a] Assuming that the cloud is not significantly growing or shrinking over time.

4.2.3.1. Area Registration

In order to keep track of the Locked Area and Focus Area usage, the processing model defines a set of *Area Registrations AR* which are generated during run-time for the Focused Situation Processing. An area registration exists within the scope of one specific Scenario Processing Template $\tau \in T$ and is therefore shared among all (possible) situations from one template:

$$AR_\tau := \{ar_{\tau,l}, ...\}_{l=1...n}$$

for n Area Registrations $ar_{\tau,l}$ where each Area Registration is a tuple of one Focus Situation Processing Instance, one Locked Area, one Focus Area and one Time Frame for which it is valid, as defined in Figure 4.2.6.

Following the definitions of Allen [All83], a time frame is defined as a start time and an end time where they are closed in their lower end (start time) and open on their upper end (end time) (Figure 4.2.6). Allen defines various relations between two time frames like before, after, during or overlaps. In order to test for an overlap, the following function is defined:

$$TimeFrameOverlap : TF \times TF \to \mathbb{N}$$

which takes two time frames and determines if they overlap as specified by [All83] and returns 0 if the two time frames do not overlap or the size of the overlap otherwise.

Based on the Area Registration, the unique constraint regarding the Locked Area usage can be formalized as follows:

Let

$$ar_{\tau,l}, ar_{\tau,k} \in AR_\tau \quad \text{with} \quad l,k \in 1...|AR_\tau| \quad \text{and } l \neq k$$

then

$$\begin{aligned}\neg\exists \quad & \pi_{LockedArea}(ar_{\tau,l}) \cap \pi_{LockedArea}(ar_{\tau,k}) \neq \emptyset \ \wedge \\ & TimeFrameOverlap(\pi_{TimeFrame}(ar_{\tau,l}), \pi_{TimeFrame}(ar_{\tau,k})) \neq 0 \wedge \\ & \pi_{FpInstance}(\pi_{FpIteration}(ar_{\tau,l})) \neq \pi_{FpInstance}(\pi_{FpIteration}(ar_{\tau,k}))\end{aligned}$$

where π_x is the projection to x

(4.1)

In order to manage the Area Registrations, two functions for the registration and release of the areas are needed. The function's definition however has to be scenario-specific as they need to implement a scenario-specific collision-handling as discussed in Section 4.5.4. Thus, in the following, the functions are only declared together with a definition of their general, scenario independent, behavior:

Definition 4.5 (Focus Area and Locked Area Registration Life-Cycle Functions)**.**

Registration Function (Create Area Registration)

The registration function tries to create a new Area Registration for the given Focus Area, Locked Area and for the given new Focused Situation Processing Iteration $fpi \in FPI_\tau$ in the given time frame $tf \in TF$. According to (4.1), the registration must fail when the requested Locked Area $la \in LA$ overlaps with another already assigned Locked Area registration in the same time frame. If the registration of the requested area is not possible, the function does *not* create a new Area Registration but provides the set of Focused Situation Processing Iteration $col \in \mathcal{P}(FPI_\tau)$ whose Locked Areas overlap with the requested Locked Area for the requested time frame. If the registration is successful, the function results in the creation of a new Area Registration which contains the provided parameters.

$$\begin{aligned}RegisterArea_\tau : FPI_\tau \times FA \times LA \times TF \times AR_\tau &\longrightarrow \mathcal{P}(FPI_\tau) \times AR_\tau \\ (fpi, fa, la, tf, AR_{\tau,0}) &\longmapsto (col, AR_{\tau,1})\end{aligned}$$

where for a successful registration:

$$col = \emptyset \wedge AR_{\tau,0} \cup (fpi, fa, la, tf) = AR_{\tau,1}$$

and for a failed registration attempt:

$$col \neq \emptyset \wedge fpi \notin col \wedge AR_{\tau,0} = AR_{\tau,1}$$

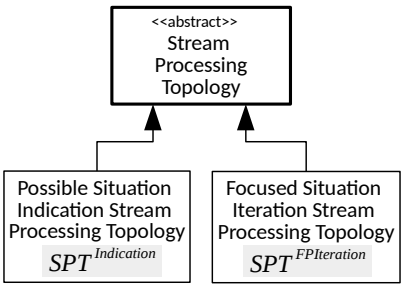


Figure 4.2.7.: A Stream Processing Topology consists of several Stream Processing Functions which form the processing topology.

Registration Release Function (Delete Area Registration)

The release function deletes a previously created Area Registration $(fpi, fa, la, tf) \in AR_\tau$ of a given Focused Situation Processing Iteration fpi of a Locked Area la and Focus Area fa for a given time frame tf:

$$ReleaseAreaRegistration : FPI_\tau \times FA \times LA \times TF \times AR_\tau \longrightarrow AR_\tau$$
$$(fpi, fa, la, tf, AR_{\tau,0}) \longmapsto (AR_{\tau,1})$$

where the result complies with:

$$AR_{\tau,0} \backslash (fpi, fa, la, tf) = AR_{\tau,1}$$

4.2.4. Stream Processing Topology

The processing Phases 1 and 3 execute the actual event stream processing to detect possible situations (Phase 1) and to verify and follow them (Phase 3)[1]. Within the processing model the used stream processing set ups are defined as Stream Processing Topologies. A Stream Processing Topology is a set of Stream Processing Functions (Nodes) where each function takes a number of event streams (Edges) as its input and may[2] produce one new event stream as output. The generated stream can then again function as input to other processing functions, thereby forming a directed acyclic graph.

The processing model differentiates two sets of Stream Processing Topologies based on their use and capabilities. For the Phase 1 processing, the model defines the set of *Possible Situation Indication Stream Processing Topologies* $SPT^{Indication}$ where for each Scenario Processing Template τ exists only one such topology $SPT_\tau^{Indication}$. For the Phase 3 processing, the model defines the set of *Focused Situation Iteration Stream Processing Topologies* $SPT^{FPIteration}$ where for each Focused Situation Processing Instance, the set

[1] As Phase 2 only decides if a new Focused Situation Processing Instance shall be created, it does not do any event stream processing.

[2] The later Phase 3 processing defines stream processing functions which may manipulate a processing context which can be their only result (See Section 4.6.2.5).

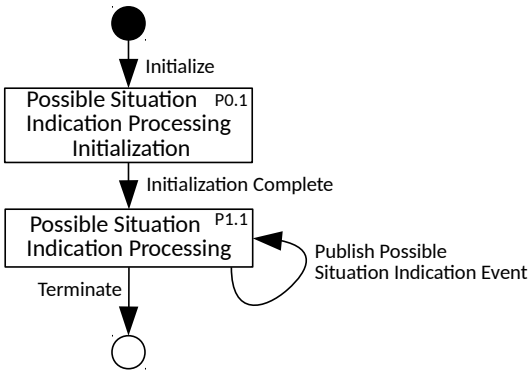


Figure 4.3.1.: Overview of Processing Phase 0 and 1

contains several topologies as for each iteration of the Focused Situation Processing a separate stream processing topology is needed (see Section 4.6).

The detailed definition of the topologies and the stream processing functions takes place for the Phase 1 processing in Section 4.3 and in Section 4.6.2.5 for the Phase 3 processing.

4.3. Phase 1: Possible Situation Indication Processing State P1.1

Phase 1 executes the stream processing required to find possible situation indications. The processing is based on the execution of the Stream Processing Topology $SPT_{\tau}^{Indication}$ (Figure 4.3.1) that is defined during the execution of Phase 0. The stream processing takes a number of event streams as its input to screen them for situation indications. The result of the Phase 1 processing is a single stream of Possible Situation Indication Events $\sigma_{\tau}^{Indication} \in \Phi^{Indication}$ specific for the current Scenario Processing Template which is then handled by processing Phase 2.

As the situation indication processing design is the basis for the definition of Phase 0, the Situation Indication Initialization, this section first defines the Phase 1 processing and is then followed by the definition of Phase 0.

4.3.1. General Design Considerations for Phase 1

The situation indication phase screens the possible great number of event streams produced by a measured system to *discover indications for possibly relevant situations*. The indication process is intended to *identify sections of the overall event streams* for the processing system to focus on in more detail. With this, its central goal is to reduce the amount of event data that needs to be processed in further detail by the Phase 3 processing to analyze relevant situations.

As specified in Section 2.2.2 by requirement SC2, the possible situation indication processing must not prevent scalability with growing numbers of events that need to be monitored. In order to cope with this requirement, the processing done in this phase has to

be fairly simple. As a trade-off for this simplicity, the processing is assumed to result in low quality results by producing a high number of false possible situation indications as a detailed verification is not possible within the given boundaries.

In general the event volume that has to be coped with in this step is the result of two factors:

1. The amount of *separate event streams* generated by *separate measurement sources* (e.g. the separate streams generated by each of the monitored solar panels) and
2. the amount of *data per event stream* as it might be generated by some high speed monitoring system like power quality related frequency measurement equipment in a Smart Grid scenario.

These two factors result in the following considerations for the design of the Phase 1 possible situation indication stream processing: In order to handle big amounts of measurement data within single data streams, the situation indication function should only invest a limited amount of processing time per event. To cope with big numbers of parallel event streams, the situation indication should only require the correlation between a limited subset of the available streams in order to allow for a parallelization of the situation indication processing.

Based on these general considerations, an initial approach for the Possible Situation Indication Stream Processing Topology would only be one processing step, realized by a *Single Event Stream Processing Function* which implemented a scenario-specific mechanism to detect Possible Situation Indication Events. For this approach, the function is instantiated once for each event stream that is to be monitored for possible situations. In order to guarantee for horizontal scalability, the indication functions are only allowed to each look at one single event stream with no correlation to other processing function instances. However, such an initial approach needs to extended by the capability to incorporate pre-queried background knowledge as parameters to the stream processing functions. Querying the background knowledge before the stream processing starts protects the stream processing from negative performance impacts due to the queries.

Even though such an approach guarantees for horizontal scalability due to the separate indication detection, it is not capable to handle more complex scenarios which require the correlation between *fixed* sets of event streams in order to detect indications. This limitation leads to the actual processing mechanism which is defined in the remainder of this sub-section.

4.3.2. Definition of the Situation Indication Stream Processing

The final approach for the Possible Situation Indication Processing can be described as an acyclic graph of stream processing functions which represent the nodes with the interconnecting event streams as the edges.

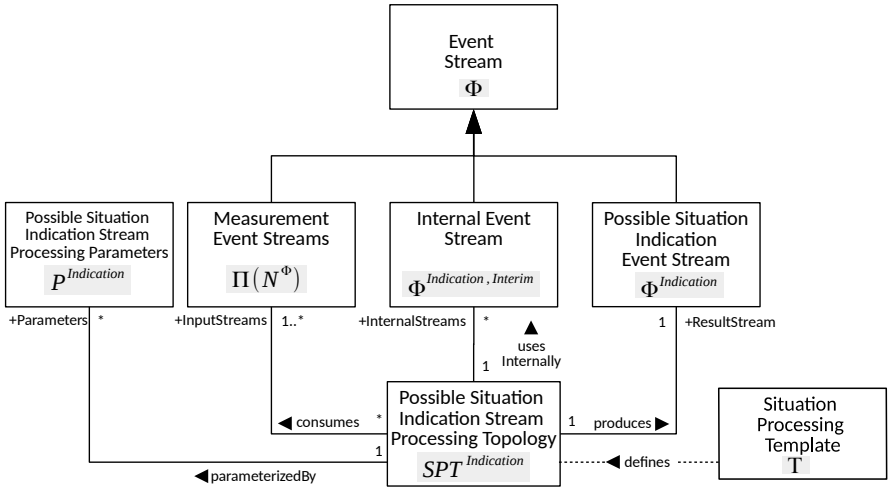


Figure 4.3.2.: A Possible Situation Indication Stream Processing Topology $SPT^{Indication}$ consumes several event streams from the set $\Pi(N^{\Phi})$ in which it is searching for possible situations and produces a single event stream $\sigma_{\tau}^{Indication} \in \Phi^{Indication}$ of Possible Situation Indication Events. Internally it may create several event streams from the set $\Phi^{Indication,Internal}$ in order to create its processing graph.

Within the graph, the following three kinds of event streams can be defined (Figure 4.3.2):

$\Pi(N^{\Phi})$
: as the set of event streams which are the input of the Situation Indication Processing Topology.

$\Phi_{\tau}^{Indication,Interim}$
: as the set of event streams that carry internal interim results from one stream processing function to another *within* the scope of the Situation Indication Processing Topology of a given Scenario Processing Template τ.

$\phi_{\tau}^{Indication} \in \Phi^{Indication}$
: as the resulting event stream of the one Situation Indication Processing Topology of a given Scenario Processing Template τ.

In order to allow the usage of background knowledge within the event stream processing functions, the required knowledge needs to be retrieved *before* the actual stream processing starts in order to prevent the negative impacts on the stream processing performance caused by the knowledge base access. In order to provide the knowledge to the processing functions, a set of parameters can be defined:

$$P_{\tau}^{Indication} := \{p_{\tau,l}^{Indication}, ...\}_{l=1...n}$$

for n Stream Processing Functions in $SPT_{\tau}^{Indication}$

The parameters are retrieved from the background knowledge in a scenario-specific way prior to the execution of the stream processing function. The resulting parameters are then used by the stream processing functions, where each function takes one of the parameters.

Based on these definitions, a stream processing function itself can be declared as a function that takes multiple event streams as its input from the two sets $\Pi(N^{\Phi})$ and $\Phi_{\tau}^{Interim}$, together with one of parameters from the parameter set $P_{\tau}^{Indication}$ and produces one new event stream as its result which is either an interim result stream from the set $\Phi_{\tau}^{Interim}$ or the stream of situation indications $\phi_{\tau}^{Indication} \in \Phi^{Indication}$:

Definition 4.6 (Indication Stream Processing Function)**.**

$$sp_{\tau}^{Indication} : \mathcal{P}(\Pi(N^{\Phi}) \cup \Phi_{\tau}^{Indication,Interim}) \times P_{\tau}^{Indication} \rightarrow \Phi_{\tau}^{Indication,Interim} \cup \Phi^{Indication}$$

Further the set of all Stream Processing Functions $SP_{\tau}^{Indication}$ within the Indication Processing Topology of one Scenario Processing Template can be defined as

$$SP_{\tau}^{Indication} := \{sp_{\tau,l}^{Indication}, ...\}_{l=1...n}$$
for n Stream Processing Functions in $SPT_{\tau}^{Indication}$

As a result, the Stream Processing Topology $SPT_{\tau}^{Indication}$ for the situation indication phase can be defined as a Tupel of the set of stream processing functions $SP_{\tau}^{Indication}$ with the set of parameters $P_{\tau}^{Indication}$ and the three sets of event streams $\Pi(N^{\Phi})$, $\Phi_{\tau}^{Indication,Interim}$ and $\phi_{\tau}^{Indication}$:

Definition 4.7 (Indication Stream Processing Topology)**.**

$$SPT_{\tau}^{Indication} := (P_{\tau}^{Indication}, SP_{\tau}^{Indication}, \Pi(N^{\Phi}), \Phi_{\tau}^{Indication,Interim}, \phi_{\tau}^{Indication})$$

4.3.2.1. Stream Duplication and Merging

As the stream processing topology allows the assignment of a single stream as input to more than one stream processing function as well as the publication of one event stream by more than one function, definitions of the handling of these cases need to be made.

When an event stream is assigned as input to more than one stream processing function, the stream is duplicated so that all functions get all events from the event stream. When two or more stream processing functions in the topology publish to the same event stream, the individual output streams are merged honoring the time ordering of the events.

4.3.3. Result of the Situation Indication Phase

The defined situation indication Stream Processing Topology $SPT_{\tau}^{Indication}$ produces a stream of situation indication events $\phi_{\tau}^{Indication}$. Each single event contained in the stream

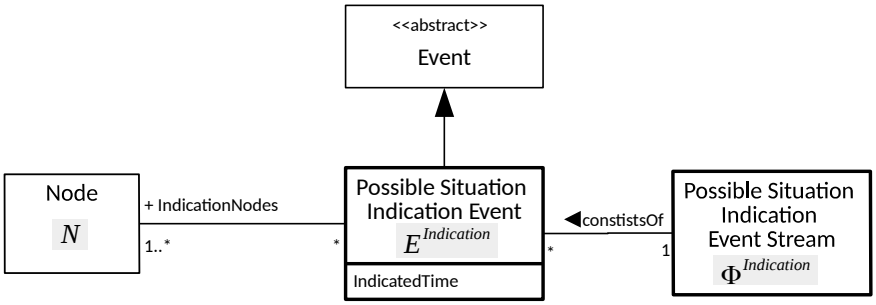


Figure 4.3.3.: Possible Situation Indication Event

represents the indication of a possible situation, where each event contains a set of nodes $\in N$ for which the possible situation is indicated and a point in time for which the possible situation was detected (Figure 4.3.3).

It is not required or assumed that a single situation is only reported once by the possible situation indication processing. Instead, it is the task of the Phase 2 processing to group together related indications and to drop duplicates. As such the generated event stream $\phi_{\tau}^{Indication}$ is the input for the processing in Phase 2 which is discussed in Section 4.5.

4.4. Phase 0: Possible Situation Indication Processing Initialization State P0.1

Based on the definition of the processing Phase 1, the initial Phase 0 can now be defined.

During the Possible Situation Indication Processing Initialization phase, the information from the Scenario Processing Template $\tau \in T$ is combined with the background knowledge $k \in K$ to create the actual Indication Processing Topology $SPT_{\tau}^{Indication}$ suitable for the monitored system. As such, this setup phase prepares the situation indication processing that takes place in the following Phase 1 (Figure 4.3.1). The initialization of the processing instances of the other phases (2 & 3) is however not part of the initial setup done by Phase 0 as the required information for their setup is based on the situation-specific results of the preceding processing phases.

The topology $SPT_{\tau}^{Indication}$ is created by a builder function $Build_{\tau}^{Indication}$ which takes as input, a set of nodes with linked event stream that shall be monitored for possible situations $\mathcal{P}(N^{\Phi})$ together with the available background knowledge $k \in K$ and defines the Indication Stream Processing Topology for the current scenario template $SPT_{\tau}^{Indication} \in SPT^{Indication}$. The function $Build_{\tau}^{Indication}$ itself is therefore specific for each scenario τ and also defined by the Scenario Processing Template (Figure 4.3.2).

Definition 4.8 (Indication Stream Processing Builder)**.**

$$\begin{aligned} Build_{\tau}^{Indication} : \mathcal{P}(N^{\Phi}) \times K &\longrightarrow SPT^{Indication} \\ (n, k) &\longmapsto spt \end{aligned}$$

where the result complies with

$$\pi_{StreamProcessingFunctions}(spt) \neq \emptyset \wedge \pi_{InputStreams}(spt) \neq \emptyset \wedge \pi_{ResultStream}(spt) \in \Phi^{Indication}$$

The builder creates a new Stream Processing Topology for a specific set of relevant nodes $\mathcal{P}(N^{\Phi})$ which is retrieved from the background knowledge k by a scenario-specific *Indication Nodes Query Function* $Q_{\tau}^{Indication}$ which is defined by the Scenario Processing Template:

Definition 4.9 (Indication Nodes Query Function)**.**

$$\begin{aligned} Q_{\tau}^{Indication} : K &\longrightarrow \mathcal{P}(N^{\Phi}) \\ k &\longmapsto n \end{aligned}$$

where the result complies with

$$n \neq \emptyset$$

Both functions $Q_\tau^{Indication}$ and $Build_\tau^{Indication}$ are executed *only* in Phase 0 *before* the actual event stream processing starts. Therefore, these functions have no access to the information contained in the event streams.

Based on the two declared functions, the Situation Processing Initialization can be defined as shown in Algorithm 1. The result of the algorithm and the processing Phase 0 is the defined Situation Indication Stream Processing Topology $SPT_\tau^{Indication}$ specifically generated for the system that is to be monitored.

Data: $k \in K$
Result: $indicationTopology \in SPT^{Indication}$

1 $indicationNodes \leftarrow Q_\tau^{Indication}(k)$
2 $indicationTopology \leftarrow Build_\tau^{Indication}(indicationNodes, k)$
3 **return** $indicationTopology$

Algorithm 1: Situation Indication Processing Initialization

4.5. Phase 2: Focused Processing Initialization

The Phase 2 processing categorizes the raised situation indications from Phase 1. The responsibility of this phase lies in two areas, (A) the triggering of a new Focused Situation Processing Instance if the situation can not be assigned to an already existing Focused Situation Processing Instance and (B) the classification and filtering of duplicate situation indications for situations that are already processed in a current Focused Situation Processing Instance. The categorization itself is largely based on scenario-specific rules which are specified by the corresponding Scenario Processing Template τ. The Phase 2 processing will thus classify each incoming indication in one or more of the following categories based on the actions that are executed for handling the indication.

A complete example of the process defined in this section based on the Cloud Tracking scenario is given at the end of this section in Subsection 4.5.9.

Definition 4.10 (Indication Classification Results)**.**

New Possible Situation
: The indication regards a potential new situation that is not yet investigated and a new Focused Situation Processing Instance needs to be started.

Additional Indication
: The indication is related to one ore more currently ongoing Focused Situation Processing Instances and needs to be assigned to them to be incorporated into their analysis process.

Duplicate Indication
: The indication is a duplication of a previous indication and can be discarded without the need to correlate it with ongoing or past Focused Situation Processing Instances.

Ignored Indication
: No analysis of the indication needs to be initialized even though it could be matched to one or more ongoing Focused Situation Processing Instances as scenario-specific rules decided that no assignment to the existing instances is needed for the current scenario.

Figure 4.5.1 gives an overview of the processing flow within this processing phase and the corresponding classification results which are discussed in the following subsections.

The following subsection discusses the pre-classification which is used to filter out duplicate indications. The later subsections will then cover the classification of the remaining events and the action execution based on the classification results.

Figure 4.5.1.: Overview of the handling of a received possible situation indication event $e^{Indication}_{\tau,j} \in \phi^{Indication}_{\tau}$ by processing Phase 2

4.5.1. Indication Pre-Classification State P2.1

Every indication $e_{\tau,i}^{Indication} \in \phi_{\tau}^{Indication}$ received regarding a possible situation related to a set of nodes $\pi_{IndicatedNodes}(e_{\tau,i}^{Indication})$, is checked for being a *duplicate* with regard to previously received indications for the same set of nodes. If the indication is considered a duplicate, it is dropped without further processing (State P2.8) otherwise the classification of the indication continues with the Locked Area and Focus Area determination (State P2.2) and based on it, the collision detection with other Focus Processing Instances based on *their* Locked and Focus Areas in State P2.3.

The processing model considers an indication $e_{\tau,i}^{Indication}$ as a duplicate if a *previous* indication $e_{\tau,j}^{Indication}$ for the same set of nodes was already raised *recently* where the time frame for this check is specific to the given scenario and needs to be expressed as a scenario-specific parameter $p_{\tau}^{Indication,Dup}$. The duplication detection is then done by the following pre-classification function $PreClass^{Indication}$:

Definition 4.11 (Indication Pre-Classification Function)**.**

$$PreClass^{Indication} : \phi_{\tau}^{Indication} \times \Phi_{\tau}^{Indication} \times \mathbb{N} \longrightarrow \{true, false\}$$
$$(e_{\tau,i}^{Indication}, \phi_{\tau}^{Indication}, p_{\tau}^{Indication,Dup}) \longmapsto res$$

where $res = true$ indicates a duplicate indication when

$$\begin{aligned} \exists \quad & e_{\tau,j}^{Indication} \in \phi_{\tau}^{Indication} \wedge \\ & \pi_{IndicationNodes}(e_{\tau,j}^{Indication}) = \pi_{IndicationNodes}(e_{\tau,i}^{Indication}) \wedge \\ & \pi_{IndicatedTime}(e_{\tau,j}^{Indication}) \geq \pi_{IndicatedTime}(e_{\tau,i}^{Indication}) - p_{\tau}^{Indication,Dup} \wedge \\ & \pi_{IndicatedTime}(e_{\tau,j}^{Indication}) \leq \pi_{IndicatedTime}(e_{\tau,i}^{Indication}) \wedge \\ & e_{\tau,j}^{Indication} \neq e_{\tau,i}^{Indication} \end{aligned}$$

and $res = false$ otherwise.

It is important to note, that this check can be done without considering currently running or past Focused Situation Processing Instances and without querying any additional background knowledge. It is thus used to reduce the number of events that require a more detailed classification in the following steps P2.2 to P2.4.

Example 5

For example for the cloud tracking scenario, a very simple situation indication processing could check for solar panels that have very little to no energy production as it would be the case if a cloud currently shades these panels. This condition is however not only met when the cloud first starts to shade the panel but also the whole time while the panel is shaded. Therefore, the indication processing will continue to raise new indication events for the given panel until the cloud is not shading it anymore. The pre-classification of such recurring indications as duplicates can be used

to suppress these additional indications.

4.5.2. Potential Locked Area and Focus Area and Time Frame Determination

State P2.2

The following collision detection is done based on the potential Locked Area of the indicated possible situation. The potential Locked Area therefore has to mark the area of the potential situation in order to allow the detection of other situations based on an overlap with their Locked and Focus Area. In order to be usable for this process, the potential Locked Area needs to adhere to the conditions specified by Definition 4.3.

In addition to the determination of the potential Locked Area, also the potential Focus Area is determined during this phase. The potential Focus Area, as discussed in Section 4.1, is a set of nodes which share a common possible relevance for the indicated possible situation. If the received indication results in the creation of a new Focused Situation Processing Instance, the potential Focus Area is required in order to initialize the new processing task. As for the potential Locked Area, the potential Focus Area also has to adhere to the previously defined conditions in Definition 4.4.

In addition, to the potential Locked Area and potential Focus Area, the initial Time Frame $tf_{e_{\tau,i}^{Indication}}$ for the later registration needs to be determined to allow the collision detection to take place. The initial Time Frame for the registration equals the Time Frame that will be used for a first iteration of a new Focused Situation Processing Instance.

In order to determine the potential Locked Area and potential Focus Area as well as the initial Time Frame based on the raised possible situation indication $e_{\tau,i}^{Indication}$, a scenario-specific query function is needed that combines the indication event with available background knowledge $k \in K$ to determine the two sets of nodes that represent the potential Locked Area and the potential Focus Area together with the Time Frame:

Definition 4.12 (Potential Locked, Focus Area and initial Time Frame Query Function)**.**

$$Q_{\tau}^{PotentialLAFA} : \phi_{\tau}^{Indication} \times K \longrightarrow \mathcal{P}(N) \times \mathcal{P}(N) \times TF$$
$$(e_{\tau,i}^{Indication}, k) \longmapsto (LA_{e_{\tau,i}^{Indication}}, FA_{e_{\tau,i}^{Indication}}, tf_{e_{\tau,i}^{Indication}})$$

where the result complies with $LA_{e_{\tau,i}^{Indication}} \neq \emptyset \wedge FA_{e_{\tau,i}^{Indication}} \neq \emptyset$ and the scenario-specific definition of $Q_{\tau}^{PotentialLAFA}$ needs to adhere to the Locked Area (Def. 4.3) and Focus Area (Def. 4.4) conditions. Further the duration of the time frame $tf_{e_{\tau,i}^{Indication}}$ must be greater than zero.

Example 6

For example for a received possible situation indication in the Cloud Tracking scenario $e^{Indication}_{SmartGrid,i} \in \phi^{Indication}_{SmartGrid}$ with one indicated node $n = \pi_{IndicatedNodes}(e^{Indication}_{SmartGrid,i})$, and the indicated time $t = \pi_{IndicatiedTime}(e^{Indication}_{SmartGrid,i})$ the potential Locked Area, the potential Focus Area and the initial Time Frame are determined as follows:

The potential Locked Area would consist of exactly the one indicated node n (solar panel) as so far this represents the indicated possible situation. The potential Focus Area contains the solar panel node n and in addition any solar panel node within a certain geographical distance from n as the new Focused Situation Processing Instance needs to investigate if they are also affected by the cloud (Figure 4.2.5). The initial Time frame starts from the indicated time t and ends for example 5 minutes later at $t + 5min$.

As such, the query function can be defined as follows:
$Q^{InitialLaFa}_{SmartGrid}$:= A query function that produces the

- potential Locked Area equal to $\pi_{IndicatedNodes}(e^{Indication}_{SmartGrid,i})$.
- potential Focus Area equal to $\pi_{IndicatedNodes}(e^{Indication}_{SmartGrid,i})$ plus solar panel nodes in geographical proximity to $\pi_{IndicatedNodes}(e^{Indication}_{SmartGrid,i})$.
- initial Time Frame as $(\pi_{IndicatiedTime}(e^{Indication}_{SmartGrid,i}), \pi_{IndicatiedTime}(e^{Indication}_{SmartGrid,i}) + 5min)$.

were the area covered by the generated initial Focus Area needs to be large enough to contain typical clouds including a small area around such a cloud in order to allow the first iteration of the Focused Processing Instance to determine if a suitable area is affected by finding the borders of the affected area and thus allowing to determine the size of the affected area.

Based on the results of this processing step (the potential Locked Area $LA_{e^{Indication}_{\tau,i}}$ and potential Focus Area $FA_{e^{Indication}_{\tau,i}}$ as well as the initial Time Frame $tf_{e^{Indication}_{\tau,i}}$), the collision detection can be executed in the following state.

4.5.3. Collision Detection — State P2.3

In order to determine if a received possible situation indication $e^{Indication}_{\tau,i}$ is related to one or more current Focused Situation Processing Instances, a collision detection between the potential Locked Area $LA_{e^{Indication}_{\tau,i}}$ of the indicated possible situation and all registered Locked Areas and Focus Areas in the matching time frame is done. The collision detection results in 0 or n *Collision Tuples* $\in CT_\tau$. As defined in Figure 4.5.2, each Collision Tuple contains the indication event $e^{Indication}_{\tau,i}$ together with the colliding Area Registration $\in AR_\tau$.

In order to specify the grade of the collision, the tuple contains three more properties:

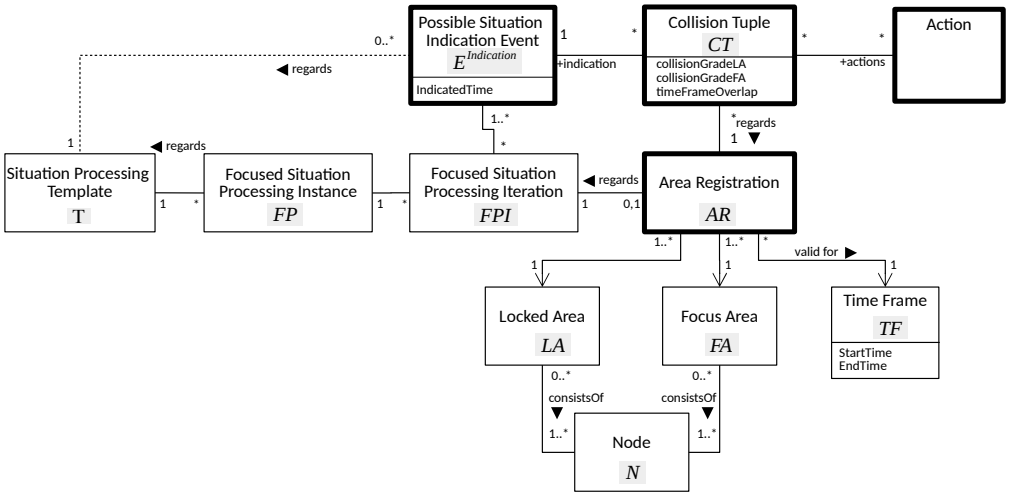


Figure 4.5.2.: The collision detection results in a number of collisions. Each collision is represented by a Collision Tuple which references the Indication Event as well as the conflicting Area Registration.

1. The grade of the collision of the potential Locked Area with the Locked Area of the Area Registration ($collisionGradeLa \in \mathbb{N}^+$) and

2. in the same way the grade of the collision of the potential Locked Area with the Focus Area of the Area Registration ($collisionGradeFa \in \mathbb{N}^+$).

3. the duration of the overlap between the initial time frame for the indication with the time frame of the Area Registration ($timeOverlapDuration \in \mathbb{N}^+$).

The two collision grades are elements of $\mathbb{N}^+$ where the number represents the count of nodes from the potential Locked Area $LA_{e_{\tau,i}^{Indication}}$ which overlaps with the corresponding area in the Area Registration. Thus, the grade can range from 0 (no overlap) to $|LA_{e_{\tau,i}^{Indication}}|$ which resembles a complete overlap with the corresponding area. In the same way, the time overlap duration represents the duration for which the two time frames overlap. Furthermore, the Collision Tuple can contain zero or more actions that are to be taken to handle the collision. The actions are however not yet set as they are determined later during the Step P2.4.

The collision detection function $ColDetect$ can thus be defined as a function that takes a given indication $e_{\tau,i}^{Indication} \in \phi_\tau^{Indication}$ together with the corresponding potential Locked Area $LA_{e_{\tau,i}^{Indication}} \in \mathcal{P}(N)$ and the corresponding initial Time Frame $tf_{e_{\tau,i}^{Indication}} \in TF$ from Step P2.2 together with one of the Area Registrations from the scope of the current processing template $AR_\tau \in AR$ and produces a Collision Tuple $ct \in CT_\tau$. As the collision detection mechanism is independent of the actual scenario, the function can be defined independently of the current processing template.

Definition 4.13 (Collision Detection Function)**.**

$$ColDetect : \phi_{\tau}^{Indication} \times TF \times \mathcal{P}(N) \times AR_{\tau} \longrightarrow CT_{\tau}$$
$$(e_{\tau,i}^{Indication}, tf_{e_{\tau,i}^{Indication}}, LA_{e_{\tau,i}^{Indication}}, ar) \longmapsto ct$$

where

$$ct = (e_{\tau,i}^{Indication}, ar, |\pi_{LockedArea}(ar) \cap LA_{e_{\tau,i}^{Indication}}|, |\pi_{FocusArea}(ar) \cap LA_{e_{\tau,i}^{Indication}}|,$$
$$TimeFrameOverlap(\pi_{TimeFrame}(ar), tf_{e_{\tau,i}^{Indication}}))$$

Based on this function, the set of all collision Tuples $CT_{e_{\tau,i}^{Indication}}$ for a given indication $e_{\tau,i}^{Indication}$ can be defined as:

$$CT_{e_{\tau,i}^{Indication}} := ColDetect(e_{\tau,i}^{Indication}, tf_{e_{\tau,i}^{Indication}}, LA_{e_{\tau,i}^{Indication}}, AR_{\tau})$$

4.5.4. Collision Action Assignment — State P2.4

Based on the results of the collision detection, a decision can be made, which actions are to be taken to handle the possibly detected collisions for a received possible situation indication. Based on the actions the overall classification in the initially defined categories (Definition 4.10) is done.

An example on how the indication collision classification and later action execution works is given in Subsection 4.5.9 for the Cloud Tracking Scenario.

In order to handle a possible situation indication together with possibly occurred collisions, the following actions are possible:

$$Actions := \{StartNew, AddToExisting, NoAction\}$$

StartNew
: Request the start of a new Focused Situation Processing Instance for the received indication.

While this first action can only be executed once for a possible situation indication independent of the number of collisions, the action *AddToExisting* can be executed multiple times for the indication in order to assign the indication to multiple Focused Situation Processing Instances that collided with the initial Locked Area.

AddToExisting
: Assign the received indication to an already existing Focused Situation Processing Instance that is considered related to this particular (possible) situation

based on the detected collision. Thus, the action can be assigned for multiple collision tuples in order to assign the indication event to multiple Focused situation Processing Instances.

NoAction

Do not take any action for a particular collision. This can also be assigned to multiple collision tuples in order to mark them as not relevant for the handling of the current indication event.

Based on the kind of collisions that occurred for a given indication, these actions can be used to handle the indication event.

The collisions that occur for an indication can be divided into four distinct categories. With regard to these four categories, only for the first two, a scenario independent handling of the indication is possible. The later two actions however contain some ambiguousness regarding their collision with existing Focused Situation Processing Instances and thus require a scenario-specific handling. All four cases are discussed in detail in the following sections:

❶ *No Overlap with any Area of any Focused Situation Processing Instance*

Condition:	$\forall ct \in CT_{e^{Indication}_{\tau,i}}:$	$\pi_{TimeFrameOverlap}(ct) = 0 \vee$
		$(\pi_{CollisionGradeLA}(ct) = 0 \wedge$
		$\pi_{CollisionGradeFA}(ct) = 0)$
Action:	$StartNew$ (Fixed)	
Classification:	*New Possible Situation*	

In this simplest case, a received possible situation indication has *no overlap* between its potential Locked Area and the Locked Area or Focus Area of *any* Area Registration in the scope of the current Scenario Template in the same or overlapping time frame. In this case the received indication is considered as a *new* possible situation and should thus lead to the creation of a new Focused Situation Processing Instance to investigate the possible situation. Therefore, this case results in the fixed action $StartNew$.

If the defined condition holds, the indication is classified as *New Possible Situation.*

❷ *Complete Overlap with the Locked Area of a Focused Situation Processing Instance*

Condition:	$\exists ct \in CT_{e_{\tau,i}^{Indication}}:$	$\pi_{TimeFrameOverlap}(ct) \neq 0 \wedge$
		$\pi_{CollisionGradeLA}(ct) = \lvert LA_{e_{\tau,i}^{Indication}} \rvert$
Action:	*AddtoExisting* (Fixed)	
Classification:	*Additional Indication*	

For this case also an indication can be mapped to an existing Focused Situation Processing Instance due the complete overlap of the potential Locked Area of the indication with the Locked Area of an existing Focused Situation Processing Instance as this area exclusively links the hereby marked nodes to the corresponding situation investigated by the Focused Situation Processing Instance. Thus, no new Focus Processing Instance is started but the indication is forwarded to the existing Focused Situation Processing Instance. Therefore, the *fixed* Action for this case is $AddToExisting$.

If the defined condition holds, the indication is classified as an *Additional Indication.*

❸ *Partial Overlap with the Locked Area of a Focused Situation Processing Instance*

Condition:	$\exists ct \in CT_{e_{\tau,i}^{Indication}}:$	$\pi_{TimeFrameOverlap}(ct) \neq 0 \wedge$
		$(0 < \pi_{CollisionGradeLA}(ct) < \lvert LA_{e_{\tau,i}^{Indication}} \rvert)$
	$\neg\exists ct \in CT_{e_{\tau,i}^{Indication}}:$	$\pi_{TimeFrameOverlap}(ct) \neq 0 \wedge$
		$\pi_{CollisionGradeLA}(ct) = \lvert LA_{e_{\tau,i}^{Indication}} \rvert$
Action(s):	$AddToExisting, NoAction$ (Scenario Specific)	
Classification:	*Additional Indication* or *Ignored Indication*	

As the partial collision with at least one Locked Area prevents the creation of a new Focused Situation Processing Instance, only two possible actions remain ($AddToExisting$ and $NoAction$) to handle the indication[3]. The decision, if the indication event shall be assigned to a colliding Focused Situation Processing Instance or if no action is to be taken is based on a scenario-specific function which has to assign one of the two actions to each collision tuple (Definition 4.14). If the defined condition holds, the indication is classified as either an *Additional Indication* if it was assigned to at least one Focused Situation Processing Instance or as *Ignored Indication* if it was *not* assigned to any Focused Situation Processing Instance.

[3]Starting a new Focused Situation Processing Instance is not possible anymore as the Area Registration that needs to be created for the new instance would not comply with the Locked Area uniqueness constraint (4.1).

Definition 4.14 (Partial Locked Area Collision Action Assignment Function)**.**

$$ActionAssignment_{\tau}^{PartialLaOverlap} : \mathcal{P}(CT_{\tau}) \longrightarrow \mathcal{P}(CT_{\tau})$$

❹ *Complete or Partial overlap with the Focus Area of a Focused Situation Processing Instance*

Condition:	$\exists ct \in CT_{e_{\tau,i}^{Indication}}$:	$\pi_{TimeFrameOverlap}(ct) \neq 0 \wedge$
		$\pi_{CollisionGradeFA}(ct) > 0$
	$\neg\exists ct \in CT_{e_{\tau,i}^{Indication}}$:	$\pi_{TimeFrameOverlap}(ct) \neq 0 \wedge$
		$\pi_{CollisionGradeLA}(ct) > 0$
Action(s):	$StartNew, AddToExisting, NoAction$ (Scenario Specific)	
Classification:	*New Possible Situation* and/or *Additional Indication* or *Ignored Indication*	

As only collisions with one or more Focus Areas of Focused Situation Processing Instances where detected, the indication may be handled by all three kinds of actions ($StartNew$, $AddToExisting$ and $NoAction$). As for the previous case ❸, the decision how the indication shall be handled is based on a scenario-specific function (Definition 4.15). The function assigns to each collision tuple either $AddToExisting$ or $NoAction$. Further it may request that the action $StartNew$ is executed. If the defined condition holds, the indication can be classified as a *New Possible Situation* if a new Focused Situation Processing Instance was started based on the indication. In addition, it can be classified as *Additional Indication* if it was assigned to one or more already existing Focused Situation Processing Instances. If neither of these cases occurred, the indication is classified as *Ignored Indication.*

Definition 4.15 (Focus Area Collision Action Assignment Function)**.**

$$ActionAssignment_{\tau}^{FaOverlap} : \mathcal{P}(CT_{\tau}) \longrightarrow \mathcal{P}(CT_{\tau}) \times \{StartNew, false\}$$

Based on these four cases, actions have been selected for handling the received indications which are executed in the two following states (P2.6 and P2.7).

4.5.5. New Focused Situation Processing Instance — State P2.6

If a received indication was classified as an indication for a *new* possible situation, a new Focused Situation Processing Instance $fp_{e_{\tau,i}^{Indication}}$ is initiated in this state. As a preparation for the start-up of this new Instance, a new Area Registration needs to be created for the potential Locked Area and potential Focus Area in order to mark the situation as being investigated. The Area Registration is created by calling the $RegisterArea_{\tau}$ function (Definition 4.5) with the potential Locked Area, potential Focus Area and initial Time

Frame which where determined during P2.2. Once the Area Registration was created, the new Focused Situation Processing Instance $fp_{e_{\tau,i}^{Indication}}$ is started. Due to the Area Registration, all later indications for the same possible situation can be assigned to the newly instantiated $fp_{e_{\tau,i}^{Indication}}$.

The actual start-up of the Focused Situation Processing Instance $fp_{e_{\tau,i}^{Indication}}$ is part of the processing Phase 3 and is thus discussed in Section 4.6.

4.5.6. Assignment to Active Focused Situation Processing Instances — State P2.7

During this state the indication is assigned to all Focused Situation Processing Instances contained in the collision set where the *AddToExisting* action was assigned by the collision classification during State P2.4. To hand over the indication event to the Focused Situation Processing Instance, the indication is assigned to the currently active Focused Situation Processing Iteration $fpi_{fp_x,it}$ of this processing instance by adding the indication to the set of additional indications $E_{fp_x,it}^{AdditionalIndications}$. It is then the responsibility of this iteration to incorporate the indication in a suitable manner in its ongoing processing. This handling of additional indications is thus discussed in the processing Phase 3 descriptions in Subsection 4.6.3.

4.5.7. Drop Possible Situation Indication — State P2.8

Based on a given indication $e_{\tau,i}^{Indication}$ that was classified as *DuplicateIndication*, the Phase 2 processing will drop this indication without any further consideration.

4.5.8. Resulting Focused Processing Initialization Algorithm

Based on the definitions in the previous sections, the complete Focused Situation Processing Algorithm can be defined as shown in Algorithm 2.

4.5.9. Phase 2 Indication Classification Example

Example 7

For example for the Cloud Tracking Scenario, the Phase 2 classification could be configured by defining the two scenario-specific collision-handling functions as follows:

- $ActionAssignment_{\tau}^{PartialLaOverlap}$ as a function that assigns *noAction* to all collision tuples.
- $ActionAssignment_{\tau}^{FaOverlap}$ as a function that assigns *noAction* to all collision tuples if at least one collision has a complete overlap with the Focus Area of an already instantiated Focused Situation Processing and request the start of a new Focused Situation processing instance if there are only one or more partial overlaps with Focus Areas of already instantiated Focused Situation Processing

Data: $e_{\tau,i}^{Indication}$, $\phi_{\tau}^{Indication}$, $p_{\tau}^{Indication,Dup}$, AR_{τ} ,k
Result: Possibly a new Focused Situation Processing Instance fp

if $PreClass^{Indication}(e_{\tau,i}^{Indication}, \phi_{\tau}^{Indication}, p_{\tau}^{Indication,Dup}) = true$ **then**
  // P2.8: Drop Possible Situation Indication, no further handling of $e_{\tau,i}^{Indication}$ needed
else
  // P2.2
  $(LA_{e_{\tau,i}^{Indication}}, FA_{e_{\tau,i}^{Indication}}, tf_{e_{\tau,i}^{Indication}}) \leftarrow Q_{\tau}^{PotentialLAFA}(e_{\tau,i}^{Indication}, k)$
  // P2.3
  $CT_{e_{\tau,i}^{Indication}} \leftarrow ColDetect(e_{\tau,i}^{Indication}, tf_{e_{\tau,i}^{Indication}}, LA_{e_{\tau,i}^{Indication}}, AR_{\tau})$
  // P2.4: Determine and assign the actions that shall be taken
  $CollisionCase1 \leftarrow true$, $CollisionCase2 \leftarrow false$, $CollisionCase3 \leftarrow false$
  forall $ct \in CT_{e_{\tau,i}^{Indication}}$ **do**
    if $\pi_{TimeFrameOverlap}(ct) \neq 0 \wedge (\pi_{CollisionGradeLA}(ct) \neq 0 \vee \pi_{CollisionGradeFA}(ct) \neq 0)$ **then**
      $CollisionCase1 \leftarrow false$
    end
    if $\pi_{TimeFrameOverlap}(ct) \neq 0 \wedge \pi_{CollisionGradeLA}(ct) = |\pi_{LockedArea}(e_{\tau,i}^{Indication})|$ **then**
      $CollisionCase2 \leftarrow true$
      // add fixed action
      $CT_{e_{\tau,i}^{Indication}}^{Actions} \leftarrow CT_{e_{\tau,i}^{Indication}}^{Actions} \cup \{(ct, addToExisting)\}$
    end
    if $\pi_{TimeFrameOverlap}(ct) \neq 0 \wedge \pi_{CollisionGradeLA}(ct) < |\pi_{LockedArea}(e_{\tau,i}^{Indication})|$ **then**
      $CollisionCase3 \leftarrow true$
    end
  end
  $startNewFocusedProcessing \leftarrow false$
  if $CollisionCase1 = true$ **then**
    $startNewFocusedProcessing \leftarrow true$
  else
    if $CollisionCase2 = true$ **then**
      // actions where already assigned
    else
      if $CollisionCase3 = true$ **then**
        // Assign Scenario specific actions
        $CT_{e_{\tau,i}^{Indication}}^{Actions} \leftarrow ActionAssignment_{\tau}^{PartialLaOverlap}(CT_{e_{\tau,i}^{Indication}})$
      else
        // Assign Scenario specific actions
        $(CT_{e_{\tau,i}^{Indication}}^{Actions}, startNewFocusedProcessing) \leftarrow$
          $ActionAssignment_{\tau}^{FaOverlap}(CT_{e_{\tau,i}^{Indication}})$
      end
    end
  end
  // P2.5: Execute the assigned actions
  forall $ct \in CT_{e_{\tau,i}^{Indication}}^{Actions}$ **do**
    if $AddToExisting = \pi_{action}(ct)$ **then**
      // P2.7
      $ar \leftarrow \pi_{AreaRegistration}(t)$
      $fpi \leftarrow \pi_{FpIteration}(ar)$
      $E_{fpi}^{AdditionalIndications} \leftarrow E_{fpi}^{AdditionalIndications} \cup \{e_{\tau,i}^{Indication}\}$
    end
  end
  if $startNewFocusedProcessing = true$ **then**
    // P2.6
    $fp_{e_{\tau,i}^{Indication}} \leftarrow createNewFocusedProcessing(e_{\tau,i}^{Indication})$
    $(collions, AR_{\tau}) \leftarrow$
      $RegisterArea_{\tau}(fpi_{fp_{e_{\tau,i}^{Indication}},1}, LA_{e_{\tau,i}^{Indication}}, FA_{e_{\tau,i}^{Indication}}, tf_{e_{\tau,i}^{Indication}}, AR_{\tau})$
    return $fp_{e_{\tau,i}^{Indication}}$
  end
end

Algorithm 2: Focused Situation Processing Initialization

Figure 4.5.3.: Example Collision Classification for two raised indications e_{P1} and e_{P2}.

Tasks.

Where the potential Locked and Focus Area is determined as defined in the Example 4.5.2.

Figure 4.5.3 Part 1 shows a small field of solar panels, where on the right hand side a cloud is shading four panels. This cloud is already tracked by the Focused Situation Processing Instance FP#1. Now a new cloud entered the field of solar panels from the north where it first shades the solar panel $P2$ and a little later the solar panel $P1$. The shading of the two panels results in two new Possible Situation Indication events e_{P2} and e_{P1} which need to be handled by the Phase 2 Processing.

As the event e_{P2} appeared earlier than e_{P1}, it is handled first by the Phase 2 Processing. The First step is the pre-classification which it passes as it is the first indication raised for this particular node $P2$. In the next step, the potential Locked and Focus Areas are determined for the indication as shown in 4.5.3 Part 2. As the potential Locked Area of e_{P2} does not overlap with any other registered area, no collision is detected by the collision detection function (State P2.3). Therefore, a new Focused Situation Processing Instance $FP\#2$ needs to be created. The Area Registration for $FP\#2$ then contains the potential Locked and Focus Area of e_{P2} thereby marking these areas for the new Focused Situation Processing Instance $FP\#2$ as shown in Figure 4.5.3 Part 3. Therefore, the event e_{P2} is classified as a *New Potential Situation*.

Now the second indication event e_{P1} is processed by Phase 2. First the pre-classification is again passed as it is the first event for the solar panel $P1$. In the next step, the potential Locked Area and potential Focus Area for e_{P1} are determined as shown in 4.5.3 Part 4. for this event, the potential Locked Area has a complete overlap with the Focus Area of $FP\#2$. Based on the initially defined rules, the start of a new Focused Situation Processing instance is prevented (Action *PreventNew*) and all further action execution for this event is stopped (Action *StopActionExecution*). Thus, the event e_{P1} is classified as an *Ignored Indication* and is dropped without triggering any further processing.

4.5.10. Synchronization Considerations

In order to ensure the correct function of the collision detection mechanism and with it the correct assignment of indications to already instantiated focused situation processing instances, the processing model defines two synchronization mechanisms in the scope of the Phase 2 processing which are discussed in the following two subsections.

4.5.10.1. Synchronized Collision Detection and Action Execution for parallel Indications

The Possible Situation Indication Processing (Phase 1) may raise possible situation indication events *in parallel*. The Phase 2 processing of those events should thus also be parallelized wherever possible. However, *the Processing Steps P2.3 to P2.7 need to be synchronized* between multiple parallel indication classification processes in order to prevent concurrent operations to invalidate the collision detection results (Step P2.3) before a decision on the collision-handling has been made (P2.4) *and* executed (P2.5 - P2.7).

4.5.10.2. Synchronization of Collision detection with Focused Processing Instances

Aside from handling parallel indications correctly, also a basic synchronization with the instantiated Focused Situation Processing Instances is needed in order to prevent the Possible Situation Indication Processing from outrunning the possibly slower Focused Situation Processing Instances as illustrated by the following example:

Example 8

For example for the cloud tracking scenario consider the following process for a single cloud that shades the solar panel n and does not change its position in the considered time frame:

1. The Possible Situation Indication Processing raises a Possible Situation Indication Event for the node n for the indication time 0. The Focused Situation Processing Initialization (Phase 2) determines an initial Time Frame as $t_1 := (0, 10)$ and a potential Locked Area including the node n. As no Focused Situation Processing Instance was started yet, no collisions are detected, and the Locked Area is registered for the Time Frame t_1 and a new Focused Situation Processing Instance fp_1 is created.

2. While the Focused Situation Processing Instance is active, additional indications arrive for the node n with an indication time within t_1. These indications can be assigned to the already running instance fp_1 due to the overlap of their initial Time Frame and potential Locked Area with the Area Registration of fp_1.

3. Some time later, the Focused Situation Processing Instance fp_1 is finished with its processing of time frame t_1 and is now in the process of determining the time frame and Locked Area for the next iteration but has not yet acquired this next

locked are yet.

4. Meanwhile, another Possible Situation Indication Event arrives for Node n with the indication time 11. The initial Time Frame is determined as $t_2 := (11, 21)$ with a potential Locked Area including n. As no collision is detected as there is no time overlap between t_1 and t_2, the Locked Area can be acquired and another new Focused Situation Processing Instance fp_2 is started.

5. The original Focused Situation Processing Instance fp_1 has by now determined that it will process the time frame $t_3 := (10, 20)$ next with a Locked Area that includes n. When it tries to acquire this Locked Area, the acquisition fails as the second Focused Situation Processing Instance fp_2 already occupies it. This results in the need of a merge between the two Focused Situation Processing Instances fp_1 and fp_2.

If however the Possible Situation Processing had been prevented from outrunning the Focused Situation Processing Instance fp_1, the processing would have proceeded normally without starting redundant Focused Situation Processing Instances.

In order to prevent such cases, the processing model requires the Focused Situation Processing Initialization to delay the collision detection (P2.3) for Possible Situation Indications if the determined initial time frame would outrun the current Focused Situation Processing Instances. Thus, to ensure this mechanism, the collision detection (P2.3) for any given indication event $e_{\tau,i}^{Indication}$ needs to be delayed till the following condition holds:

$$
\begin{array}{rcl}
\neg\exists fp_x \in FP_\tau & \text{with} & \\
tfLast_{fp_x} & := & \text{The time frame of the Area Registration} \\
 & & \text{of the most recent Iteration of } fp_x \\
 & \text{and} & \\
tf_{e_{\tau,i}^{Indication}} & := & \text{The initial time frame determined for } e_{\tau,i}^{Indication} \\
 & \text{where} & \\
\pi_{endTime}(tf_{e_{\tau,i}^{Indication}}) & > & \pi_{endTime}(tfLast_{fp_x}) \\
 & \wedge & \neg TimeFrameOverlap(tf_{e_{\tau,i}^{Indication}}, tfLast_{fp_x})
\end{array}
$$

4.6. Phase 3: Focused Situation Processing

The Phase 3 Focused Situation Processing resembles a specialized processing implemented by multiple Focused Situation Processing Instances, where each instance is focused on the investigation of one particular (possible) situation. The aim of a Focused Situation Processing Instance $fp_{e^{Indication}_{\tau,i}} \in FP$ which was created for an indication event $e^{Indication}_{\tau,i} \in E^{Indication}_{\tau}$ is to allow an *in-depth analysis* of the indicated possible situation for which the indication event $e^{Indication}_{\tau,i}$ was raised. Here the Focused Situation Processing Task has the following goals:

G1: verify that the indicated possible situation $e^{Indication}_{\tau,i}$ is an actual situation,

G2: analyze the situation in further detail and if necessary,

G3: follow a situation as it changes over time.

Example 9

For example for the cloud tracking scenario, the focused situation processing first has to verify that the triggering indication concerns an actual cloud and not only a malfunctioning solar panel installation. In this scenario, the verification is realized by identifying more than one solar panel in the same geographical area as affected. Then once the verification is done, the analysis of the situation determines the size of the cloud by identifying the border between the affected panels and non affected panels. Over time the focused situation processing then has to follow the movement of the cloud as the panels affected by the cloud change requiring a new analysis to detect the new cloud border.

To simplify the definitions in this section, let

$$x := e^{Indication}_{\tau,i}.$$

The *final result* $r_{fp_x} \in R^{FP}$ (Section 4.6.2.7) of every Focused Situation Processing is either

1. a *false situation*, if the indication turns out to be invalid or
2. a specific *situation analysis result*.

Further a Focused Situation Processing can publish *interim results* (Section 4.6.2.7) based on the current state of its analysis. For example for the cloud tracking scenario, the focused situation processing can publish the current position and size of the followed cloud every time a change in the clouds position or size is detected.

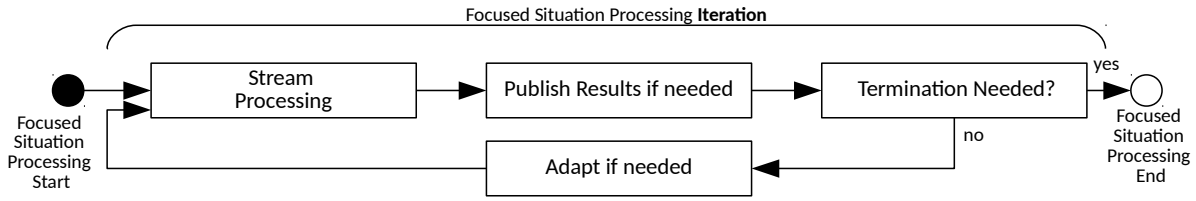


Figure 4.6.1.: Simplified view of the focused situation processing flow.

In contrast to the situation indication processing in Phase 1, the Focused Situation Processing is designed to support more complex processing tasks in order to implement the three aforementioned goals (G1-G3). To allow for such processing tasks, the Focused Situation Processing is allowed to use more resources per processed event than the Phase 1 processing as only a limited subset of the overall event load needs to be handled by a single Focused Situation Processing Instance due to its use of a limited Focus Area (Section 4.2.3). Further, the Focused Situation Processing can access the available background knowledge during the ongoing focused situation processing, something that is prohibited for the Phase 1 processing.

Moreover, the Focused Situation Processing is designed to support the automatic adaptation of its current processing setup to handle changes in the investigated situation or to adapt the processing based on new insights into the investigated situation provided by interim processing results. This adaptive capability is a central functionality of the Phase 3 processing and the overall processing model. The following Subsection 4.6.1 discusses this adaptive functionality in detail and is followed by the design of the internal structure of the Phase 3 processing in Section 4.6.2 ff.

The function itself follows a defined process which is shown in Figure 4.6.1 in its high level form (See Section 4.6.2 for the complete version). The process incorporates scenario-specific elements from the Scenario Processing Template τ which in turn define the actual analysis of the possible situation in order to verify the indication and to further investigate the situation as required for the given scenario. As such the processing function will be defined based on the process given in Figure 4.6.2 in the following sections concluded by Section 4.6.5 providing a complete definition of the focused situation processing function.

4.6.1. Adaptive Processing

The capability of the processing model to adapt itself to the current (possible) situation is one of the central functionalities of the whole processing model and is thus discussed in this section separately from the following discussions of the different Phase 3 processing steps.

Two general types of adaptation can be distinguished within the scope of the whole focused processing model which are both part of the focused situation processing:

1. An adaptation in terms of the overall processing system to set up the focused situation processing based on a raised indication *before the actual situation-specific processing started* in order to provide the environment to verify the situation's existence and to start with the situation-specific processing.

2. Adaptation of the already running focused situation processing in order to follow changes in the focused situation or based on new information on the situation. In contrast to 1, this adaptation has to take place *during an ongoing situation-specific processing*.

The initial adaptation (1) of the overall processing system is realized by instantiating a new separate processing task that is not interlinked with the existing stream processing used for the Phase 1 processing. The second kind of adaptation (2) needed for the focused situation processing however bears a higher complexity as the situation-specific analysis is already running.

In order to implement discrete adaptation steps, the Focused Situation Processing is executed as a number of iterations, where an adaptation is only possible after the iteration processing was done. The applied adaptation is then valid for the next iteration. The exact process used for the iterations and their adaptations is discussed in the following sections.

4.6.2. Focused Situation Iteration Processing

Based on the chosen iteration-based adaptation mechanism, the high level processing flow presented in Figure 4.6.1 can be refined with the following steps which are defined in detail in Subsections 4.6.2.1 to 4.6.3 (Figure 4.6.2):

1. Iteration pre-processing to set up the environment for the current iteration.

2. Generation of the iteration stream processing task based on the current situation-specific processing state.

3. Execution of the iteration stream processing.

4. Post-Iteration processing to gather results from the stream processing step and prepare for the following steps.

5. Interim result publication if any notable results where generated during this iteration.

6. Locked area and Focus Area derivation and registration update if the results of this iterations processing require a new adapted setup for the next iteration.

Further some special cases need to be considered like the termination of a Focused Situation Processing Instance or the collision of two instances based on their Locked Areas.

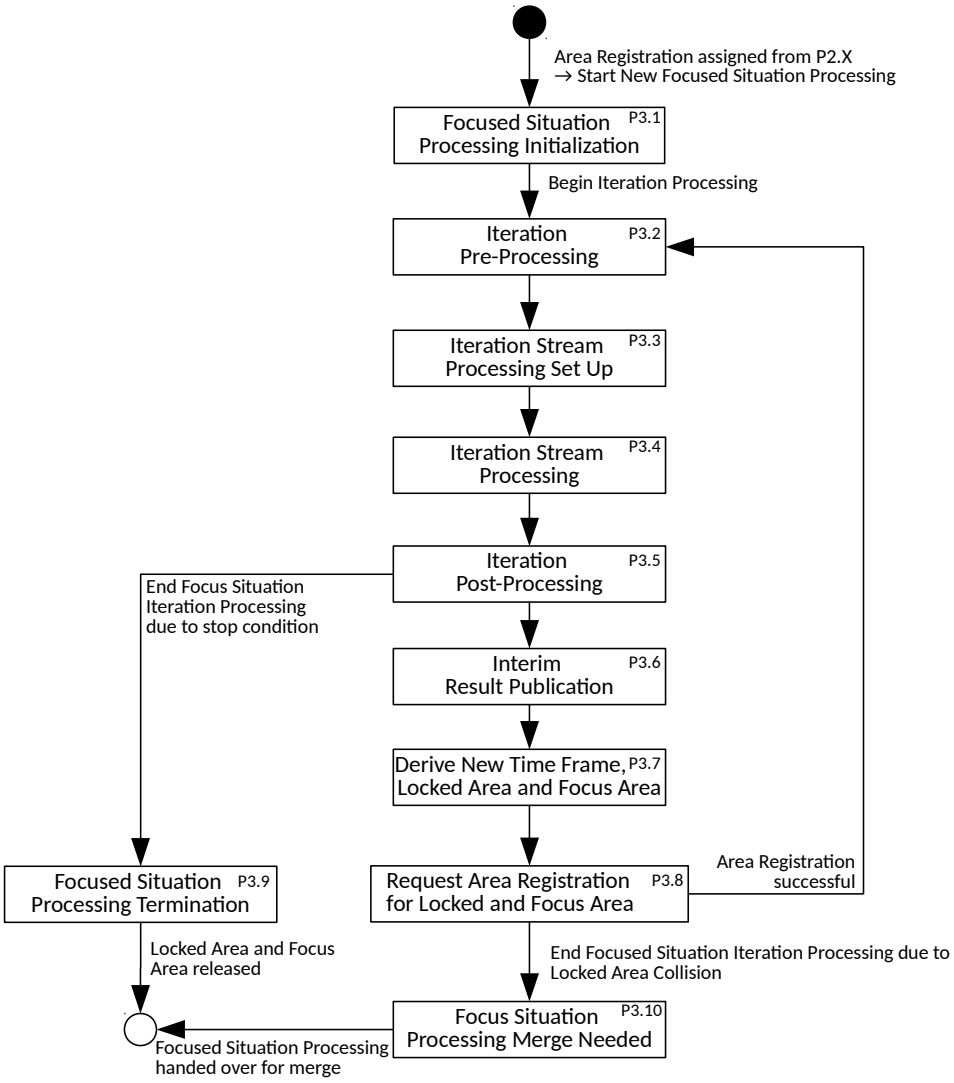


Figure 4.6.2.: Overview of the Phase 3 processing states

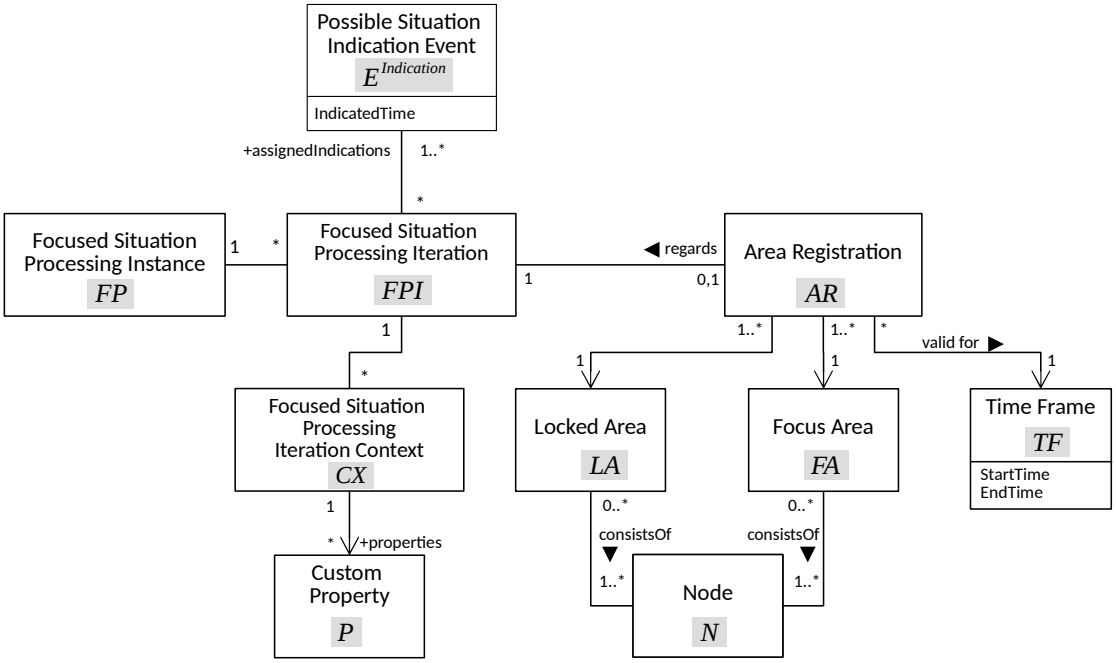


Figure 4.6.3.: Each Focused Situation Processing Instance FP has multiple Focused Situation Processing Iterations FPI where each iteration relates to zero or one Area Registration $\in AR$, a set of assigned indication events $E^{AssignedIndications}_{fpi_x,it} \subseteq E^{Indication}$ and has zero or more Context tuples $\in CX$. Each Context can contain a number of scenario-specific properties $\in P$.

4.6.2.1. Focused Situation Processing Iteration and its Environment

For each Focused Situation Processing *Iteration*, the processing model defines several iteration specific elements (Figure 4.6.3):

Focused Situation Processing Iteration Context
: In order to allow for the different steps of the iteration processing to keep the processing state, a Focused Situation Processing Context $CX_{fp_x,it}$ is defined for each iteration it of a Focused Situation Processing Instance fp_x. The context is specific to one iteration of one Focused Situation Processing Instance fp_x and is defined as a set of properties $\{p_l, ...\}_{l \in \mathbb{N}}$ that can be specified by the scenario-specific processing logic defined in the Scenario Processing Template τ. Each iteration has zero or up to three contexts assigned. Zero, if the iteration has not yet been initialized, and up to three contexts representing the results of each context-manipulating function of each iteration (see Subsection 4.6.2.2)

Area Registration
: In addition to the context, for each iteration it also zero or one Area Registration $ar_{fp_x,it} \in AR$ exist which contains the Locked Area, Focus Area and Time Frame of the corresponding iteration where the iteration is only allowed to have no Area Registration assigned if it has not yet been initialized.

Assigned Indications
: Further, for each iteration it always one set of Indication Events $E^{AssignedIndications}_{fp_x,it} \subseteq E^{Indication}$ exists which contains all events assigned to the current iteration for processing.

The described iteration specific environment is available during the whole iteration processing. In contrast to the situation *indication* processing in Phase 1, this includes the iteration stream processing which is also allowed to manipulate the current Focused Situation Processing Iteration Context.

4.6.2.2. Focused Situation Processing Iteration Context Use and Initialization

In order to initialize the Focused Situation Processing Iteration Context, the processing model defines two separate initialization steps which are both implemented by a scenario-specific function (Figure 4.6.4 and 4.6.2):

Focused Situation Processing Initialization (P3.1) for the general initialization of the Focused Situation Processing Iteration Context after instantiating a new Focused Sit-

uation Processing Instance. This step results in the initial processing context[4] $CX_{fp_x,0,3}$.

Iteration Pre-Processing (P3.2) for the initialization of the context specific to the current iteration. The Pre-initialization uses the final version (Version 3) of the context of the previous iteration $CX_{fp_x,it-1,3}$ and based on it generates the initial context version (Version 1) for the current iteration: $CX_{fp_x,it,1}$.

Once the processing context has been initialized for the current iteration (Figure 4.6.4 Step A), it is used to generate the new Stream Processing Topology for the current iteration (Step B). Based on the newly generated topology, the iteration stream processing takes place (Step C). During this phase, the context is also available in order to allow the ongoing stream processing to store processing results for later use. Due to the changes applied by the stream processing, the initial context version $CX_{fp_x,it,1}$ is transformed into $CX_{fp_x,it,2}$. Once the stream processing is finished, a *post processing* is done based on the context $CX_{fp_x,it,2}$ in order to sum up the results generated during the stream processing (Step D). The result of this transformation is the context $CX_{fp_x,it,3}$. Based on $CX_{fp_x,it,3}$ the possible publication of interim results takes place as well as the generation of a new Locked and Focus Area as preparation for the next iteration (Step E). In addition, the context is used to determine if the Focused Situation Processing Instance is finished or should continue with the next iteration. If the processing shall continue, the process starts once again which is illustrated in the remainder of Figure 4.6.4 after Step E.

4.6.2.3. Focused Situation Processing Initialization — State P3.1

In order to prepare for a new Focused Situation Processing Instance, a scenario-specific Focused Situation Processing Initialization allows the preparation of an initial Focused Situation Processing Iteration Context which serves as the basis for the context of the first iteration. The initialization function has access to the background knowledge $k \in K$, the Area Registration of the upcoming first iteration $ar_{fp_x,1}$ as well as the set of indication events assigned to the new Focused Situation Processing Instance $E_{fp_x,1}^{AssignedIndications}$. The result of the scenario-specific initialization is the processing context $CX_{fp_x,0,3}$ that is used in the next step as basis for the context of the first iteration (See following Section).

Definition 4.16 (Focused Situation Processing Initialization Function)**.**

$$Init_{\tau}^{FP} : K \times AR \times \mathcal{P}(E^{Indication}) \longrightarrow CX$$
$$k, ar_{fp_x,1}, E_{fp_x,1}^{AssignedIndications} \longmapsto CX_{fp_x,0,3}$$

[4]The context $CX_{fp_x,0,3}$ is numbered for iteration 0 context version 3 as the third version of a previous iteration is *always* used as basis for the definition of the context of the current iteration (See Subsection 4.6.2.4).

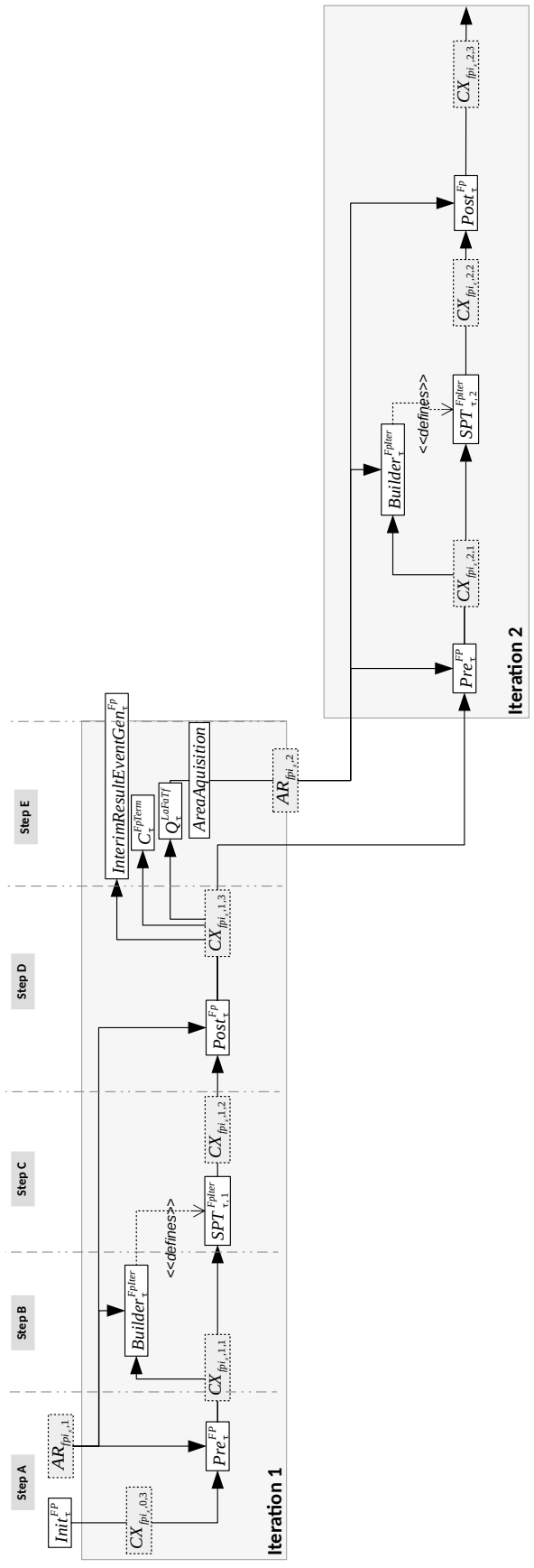


Figure 4.6.4.: Usage of the Focused Situation Processing Context as part of the iterative processing with two iterations and an adaptation between the first and the second iteration. (Note: The figure only includes context modifying steps and is thus not a complete representation of the Focused Situation Processing. See Algorithm 3 for a complete view.)

4.6.2.4. Iteration Pre-Processing State P3.2

Based on the context that resulted from the execution of the previous iteration[5], a scenario-specific iteration pre-processing function Pre_{τ}^{FP} allows the initialization of the processing context for the new iteration. Thus, it generates the context for the current iteration $CX_{fp_x,it,1}$. For this it accesses the final context of the previous iteration $CX_{fp_x,it-1,3}$. Furthermore, it can access the available background knowledge $k \in K$, the current Area Registration $ar_{fp_x,it}$ as well as the set of Indications that have been assigned to the current iteration $E_{fp_x,it}^{AssignedIndications}$.

Definition 4.17 (Pre-Iteration Processing Function)**.**

$$Pre_{\tau}^{FP} : CX \times K \times AR \times \mathcal{P}(E^{Indication}) \longrightarrow CX$$
$$CX_{fp_x,it-1,3}, k, ar_{fp_x,it}, E_{fp_x,it}^{AssignedIndications} \longmapsto CX_{fp_x,it,1}$$

4.6.2.5. Iteration Stream Processing and Iteration Stream Processing Topology Build State P3.3 & P3.4

The stream processing that is done for each Focused Situation Processing Iteration is based on a similar mechanism as the stream processing done for the Possible Situation Indication in Phase 1. The iteration stream processing is also defined as an acyclic directed graph of stream processing functions (nodes) with the interconnected event streams (edges) (see Subsection 4.3.2). In a similar way as for the Possible Situation indication Processing, the whole processing graph together with its required parameters is considered as the Stream Processing Topology $SPT^{FpIteration}$.

Iteration Stream Processing Topology

In contrast to the stream processing done during the Possible Situation Indication Processing, the processing done in each focused Situation Processing Iteration is altered in the following way:

1. Support access to the Focused Situation Processing Iteration Context $CX_{fp_x,it,1}$ in order to access previous results as well as to store current results for later use. The changes applied to $CX_{fp_x,it,1}$ during an ongoing processing thus result in a new version of the context $CX_{fp_x,it,2}$ which contains all changes done during the iterations stream processing.

2. The iteration stream processing does not lead to the generation of a possible situation indication stream but instead stores all its processing result into the stream processing context $CX_{fp_x,it,2}$.

[5] or for the very first iteration the context that resulted from the Focused Situation Processing Initialization Function: $CX_{fp_x,0,3}$.

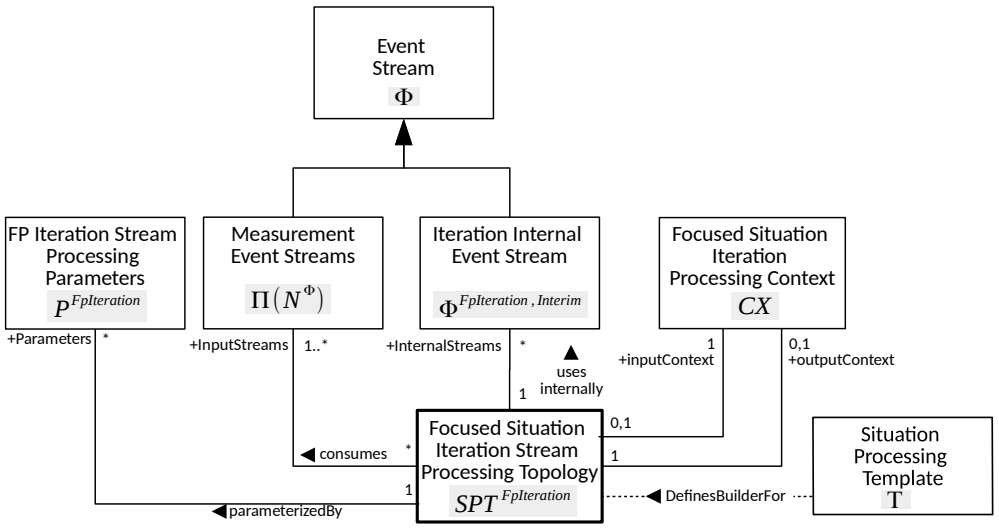


Figure 4.6.5.: A Focused Situation Iteration Stream Processing Topology $SPT^{Fpiteration}$ consumes several event streams from the set $\Pi(N^{\Phi})$. Further it uses the current Focused Situation Processing Context $\in CX$ as additional input to access results from previous iterations. As a result it produces a new version of the Focused Situation Processing Context $\in CX$ which contains the results of the stream processing so that it can be used by later process steps as well as following iterations. Internally it may create several event streams from the set $\Phi^{Indication,Internal}$ in order to create its processing graph.

As a result, only the following two kinds of event streams are defined (Figure 4.6.5):

$\Pi(N^{\Phi})$
: as the set of event streams which are the input of the Iteration Stream Processing Topology.

$\Phi_{\tau}^{FpIteration,Interim}$
: as the set of event streams that carry internal interim results from one stream processing function to another *within* the scope of *one* Iteration Stream Processing Topology of one Focused Situation Processing Instance.

In the same way as for the Indication Stream Processing in Phase 1, background knowledge can be provided to the stream processing functions contained in the topology by a set of parameters which is retrieved in a scenario-specific way from the available background knowledge:

$$P_{fp_x,it}^{FpIteration} := \{p_{fp_x,it,k}^{FpIteration}, ...\}_{k=1...n} \text{ for } n \text{ stream processing functions}$$

In addition to the event streams and the pre-fetched background knowledge, the topology has also access to the Focused Situation Processing Iteration Context $CX_{fp_x,it,1}$ of the current iteration *it* that resulted from the Pre-Processing (State P3.2). Based on the input context, the stream processing generates a new version of the context $CX_{fp_x,it,2}$ which

contains the results of the stream processing so that it can be used by the later steps in the process including the following iterations.

Based on these definitions, the stream processing functions contained in the Iteration Stream Processing Topology can be defined as stream processing functions that take as their input one or more event streams from the sets $\Pi(N^{\Phi})$ and $\Phi_{\tau}^{FpIteration,Interim}$ together with the current processing context $CX_{fp_x,it,2}$ and one of the parameters from $P_{fp_x,it}^{FpIteration}$. As their processing result, each function may produce one or no event stream from the set $\Phi_{\tau}^{FpIteration,Interim}$ together with the changed processing context $CX_{fp_x,it,2}$. Thus, the stream processing functions $sp^{FpIteration}$ for the Iteration Stream Processing Topology can be defined as follows:

$$sp^{FpIteration} : \mathcal{P}(\Pi(N) \cup \Phi^{Fpiteration,Interim}) \times P^{FpIteration} \times CX \longrightarrow (\Phi^{Fpiteration,Interim} \cup \emptyset) \times CX$$

Further the set $SP_{fp_x,it}^{FpIteration}$ which contains all stream processing functions of one iteration it of one focused situation processing instance fp_x can be defined as:

$$SP_{fp_x}^{FpIteration} := \{sp_{fp_x}^{FpIteration}, ...\}_{k=1...n} \text{ for } n \text{ stream processing functions}$$

As a result, the Iteration Stream Processing Topology $SPT_{fp_x,it}^{FpIteration}$ for an iteration it can now be defined as the tuple of the set of parameters $P_{fp_x,it}^{FpIteration}$, the set of stream processing functions $SP_{fp_x}^{FpIteration}$, the inbound event streams related to the current Focus Area $\Pi(FA_{fp_x,it})$, together with the interim event streams $\Phi_{fp_x,it}^{FpIteration,Interim}$ and lastly the current iteration processing context $CX_{fp_x,it,1}$ and the resulting changed processing context $CX_{fp_x,it,2}$:

$$SPT_{\tau}^{FpIteration} := (P_{fp_x,it}^{FpIteration}, SP_{fp_x}^{FpIteration}, \Pi(FA_{fp_x,it}), \Phi_{fp_x,it}^{FpIteration,Interim}, CX_{fp_x,it,1}, CX_{fp_x,it,2})$$

Based on the definitions from the indication stream processing, the focused processing iteration stream processing function build can be defined as follows:

Iteration Stream Processing Function Generation State P3.3

During the adaptation phase of the iterative processing, a new Stream Processing Topology needs to be derived. The definition of the topology during run-time is realized by a scenario-specific builder function $Build_{\tau}^{FpIteration}$. The builder function is similar to the function used for the initialization of the situation indication ind Phase 0. However, $Build_{\tau}^{FpIteration}$ has to generate the processing based on the current Focus Area, Locked Area and Time Frame, contained in the current Area Registration $AR_{fp_x,it}$ in combina-

tion with interim results from the previous iterations contained in the processing context $CX_{fp_x,it,1}$ in combination with the available background knowledge $k \in K$. Therefore, $Build_{\tau}^{FpIteration}$ can be declared as follows:

Definition 4.18 (Iteration Stream Processing Builder)**.**

$$\begin{aligned} Build_{\tau}^{FpIteration}: \quad & CX \times AR \times K \longrightarrow SPT^{FpIteration} \\ & (CX_{fp_x,it,1}, AR_{fp_x,it}, k \;) \longmapsto SPT_{fp_x,it}^{FpIteration} \end{aligned}$$

Iteration Stream Processing Execution State P3.4

Once the Stream Processing Topology $SPT_{fp_x,it}^{FpIteration}$ for the current iteration was defined by the previous step, the stream processing is executed. The iteration stream processing uses the iteration context $CX_{fp_x,it,1}$ and modifies it based on its stream processing results into the modified context $CX_{fp_x,it,2}$ which is used by the following steps to gather the results of the stream processing.

4.6.2.6. Post Iteration Processing State P3.5

Once the iteration stream processing is finished, the post iteration state is entered which allows a post processing of the generated results from the iteration stream processing in the previous phase. In particular this phase allows the structuring of the results in order to be usable for

1. the publication as interim or final result
2. the derivation of the next Locked Area and Focus Area and
3. the evaluation of the termination criteria of this Focused Situation Processing Instance

The scenario-specific post iteration processing function can be declared in the same way as the pre-iteration processing function (Definition 4.6.2.4). It takes the processing context that resulted from the iteration stream processing $CX_{fp_x,it,2}$ together with the background knowledge $k \in K$, the current Area Registration $ar_{fp_x,it} \in AR$ and the assigned Indication Events $E_{fp_x,it}^{AssignedIndications} \in E^{Indication}$ and produces a new version of the processing context $CX_{fp_x,it,3}$.

Definition 4.19 (Post-Iteration Processing Function)**.**

$$\begin{aligned} & Post_{\tau}^{FP}: CX \times K \times AR \times \mathcal{P}(E^{Indication}) \longrightarrow CX \\ & CX_{fp_x,it,2}, k, ar_{fp_x,it}, E_{fp_x,it}^{AssignedIndications} \longmapsto CX_{fp_x,it,3} \end{aligned}$$

4.6.2.7. Interim Focused Situation Processing Result Publication State P3.6

Aside from the final processing result of a finished Focused Situation Processing Instance, interim states within the ongoing process may be of interest for external systems. For example for the cloud tracking scenario, the current position, size, trajectory and speed of the tracked cloud is relevant for external systems to update their prognosis on the impact of the cloud on the energy production. As such, a running Focused Situation Processing Instance requires the capability to publish interim results during the ongoing processing.

In order to generate such interim result events, a scenario-specific function $InterimResultEventGen_{\tau}^{Fp}$ is defined, which for an iteration it of a Focused Situation Processing Instance fp_x, generates zero or more interim result events as the set $E_{fp_x,it}^{InterimResults} \in E^{InterimResults}$ based on the final processing context of the current iteration $CX_{fp_x,it,3}$ which contains the results of the previous post processing step:

Definition 4.20 (Interim Result Event Generation Function)**.**

$$\begin{aligned} InterimResultEventGen_{\tau}^{Fp} : CX &\longrightarrow E^{InterimResults} \\ CX_{fp_x,it,3} &\longmapsto E_{fp_x,it}^{InterimResults} \end{aligned}$$

Where if the resulting $E_{fp_x,it}^{InterimResults} \neq \emptyset$ the contained events are published to external systems to inform them about the state of the ongoing processing.

4.6.2.8. Next Iteration Locked Area, Focus Area and Time Frame Determination State P3.7

Similar to the determination of the first Locked Area $LA_{fp_x,1}$ and Focus Area $FA_{fp_x,1}$, a Locked Area $LA_{fp_x,it}$ and Focus Area $FA_{fp_x,it}$ for each following iteration it is required. Furthermore, the Time Frame for next iteration needs to be determined. Both, the Locked Area and the Focus Area as well as the Time Frame for the next iteration are determined in a scenario-specific way from the current iteration processing context together with the last iteration time frame $tf_{fp_x,it}$ and additional knowledge from $k \in K$.

With regard to the synchronization considerations discussed in Subsection 4.6.6, the Time Frame for the next iteration must not end before the end of the current iterations Time Frame $\pi_{endTime}(tf_{fp_x,it+1}) >= \pi_{endTime}(tf_{fp_x,it})$.

The corresponding query function can thus be declared as a function that takes the current processing context $CX_{fp_x,it,3}$ together with a subset of the available background knowledge $k \in K$ to generate a tuple of two sets of nodes and one Time Frame where the first set of nodes resembles the new Locked Area $LA_{fp_x,it+1}$ and the second the new Focus Area $FA_{fp_x,it+1}$ while the Time Frame is the Time Frame for the next iteration $tf_{fp_x,it+1}$.

Definition 4.21 (Iteration Locked Area, Focus Area and Time Frame Query Function)**.**

$$Q_{\tau}^{LaFaTf} : CX \times TF \times K \longrightarrow \mathcal{P}(N) \times \mathcal{P}(N) \times TF$$
$$(CX_{fp_x,it,3}, tf_{fp_x,it}, k) \longmapsto (LA_{fp_x,it+1}, FA_{fp_x,it+1}, tf_{fp_x,it+1})$$

with

$$LA_{fp_x,it+1} \neq \emptyset \wedge FA_{fp_x,it+1} \neq \emptyset \wedge$$
$$\pi_{endTime}(tf_{fp_x,it+1}) - \pi_{startTime}(tf_{fp_x,it+1}) > 0 \wedge$$
$$\pi_{endTime}(tf_{fp_x,it+1}) >= \pi_{endTime}(tf_{fp_x,it})$$

The definition of the function itself is specific to the current scenario as it needs to select *suitable* candidates for the scenario that is implemented by the current focused processing task following the conditions specified in Definition 4.3 and Definition 4.4 .

4.6.2.9. Iteration Focus Area and Locked Area Acquisition — State P3.8

Once a new Locked Area $LA_{fp_x,it+1}$ and Focus Area $FA_{fp_x,it+1}$ was derived for the next iteration $it + 1$, it needs to be registered for the time frame of the new iteration $tf_{fp_x,it+1}$ in order to link the both areas to the next Focused Situation Processing Iteration:

$$(col, AR_{\tau}) := RegisterArea_{\tau}(fpi_{fp_x,it+1}, LA_{fp_x,it+1}, FA_{fp_x,it+1}, tf_{fp_x,it+1}, AR_{\tau})$$

In the process of this acquisition, possible collisions with the registered areas of other Focused Situation Processing Instances are detected. If no collision is detected ($col = \emptyset$), the processing for the next iteration is started again with State P3.2. If however one or more collisions where detected ($col \neq \emptyset$), the focused situation processing transitions to State P3.10 to handle the collisions.

If no collision was detected, and the registration was therefore successful and the next Iteration concerns the same Time Frame as the current iteration did, the newly created Area Registration supersedes the old Area Registration for this Time Frame. Therefore, the old Area Registration is deleted:

$$AR_{\tau} := ReleaseAreaRegistration(fpi_{fp_x,it}, LA_{fp_x,it}, FA_{fp_x,it}, tf_{fp_x,it}, AR_{\tau})$$

The following example demonstrates such a process:

Example 10

This example continues Example 4.5.9 (Figure 4.5.3) where a cloud enters a monitored area from the north thus causing a new Focused Situation Processing Instance to be created.

For its first iteration, the new Focused Situation Processing Instance has only $P2$ as its Locked Area for the Time Frame tf_1 (Figure 4.6.6 Part 5). The processing of the first iteration determines that its Locked Area for the Time Frame tf_1 also needs to include the solar panel $P1$. Thus, it determines a new Locked and Focus Area for the next iteration where the Locked Area contains the nodes $P1$ & $P2$ and sets the 2nd iteration to the same time frame as the first iteration. As the new Locked and Focus Areas concern the same time frame, the area registration from iteration 1 is released and a new Area registration for iteration 2 is made, both for the same time frame but with different Locked and Focus Areas. The processing of the second iteration determines that its Locked Area is now correct for the time frame tf_2 and no further update is needed (Figure 4.6.6 Part 6).

Some time later, the cloud changes its position further to the south thus also covering the solar panels $P3$ & $P4$ (Figure 4.6.6 Part 7). The iteration responsible for the corresponding time frame, for this example iteration 3 with tf_3, again detects that its current Locked Area is not suitable for the Time Frame tf_3 and determines a new Locked Area that contains $P1$ to $P4$ and will replace the Area Registration of the current iteration as the new iteration will repeat the processing of the Time Frame tf_3. After this 4th iteration, the Locked Area for tf_3 is again correct and no update of the Locked Area for this time frame is needed (Figure 4.6.6 Part 8).

4.6.2.10. Focused Processing Merge Required — State P3.10

The current Focused Situation Processing Instance is stopped and shut down *without* releasing the current Area Registration as the investigation of the situation is still considered ongoing until a merging decision was made.

Based on the collision a separate handling process is executed which decides how to merge the two colliding ongoing Focused Situation Processing Instances. The task itself is described in Section 4.6.4.

4.6.2.11. Termination of Focused Processing and Final Result Publication — State P3.9

A running Focused Situation Processing Instance fp_x can be terminated after the processing of the current iteration it has finished. The termination is controlled by a scenario-specific termination condition C_{τ}^{FPTerm} that is evaluated against the focused situation processing context of the current iteration $CX_{fp_x,it,3}$.

Figure 4.6.6.: Locked Area updates to incorporate changes of the tracked situation. (Continuation of Example 4.5.9, Figure 4.5.3)

Definition 4.22 (Focused Situation Processing Termination Condition)**.**

$$\begin{aligned} C_{\tau}^{FpTerm} : CX &\longrightarrow \{true, false\} \\ CX_{fp_x,it,3} &\longmapsto term \end{aligned}$$

When the termination condition is positively evaluated, no further iteration is started and the Focused Situation Processing Instance is shut down.

Final Focused Situation Processing Result

Once a Focused Situation Processing Instance finishes, the overall result of its analysis has to be published. If the investigated potential situation turned out to be a false positive, the result of the processing is $FalseSituation$. If however, the situation was verified as a valid situation, a scenario-specific set of properties can be returned to provide the analysis results to external systems. Therefore, the processing result can be defined as

$$R^{FP} := \{FalseSituation, \{p_i, ...\}_{i=1...n}\}$$

with n scenario-specific properties representing the result of a successful focused situation processing.

In addition to the processing results, the decision needs to be made, whether the Area Registration from the last iteration is to be kept in order to mark the situation or if the Area Registration shall be released as the assumption under which the registration was made turned out to be incorrect. As the decisions that need to be made are specific to the current scenario, a scenario-specific function can be defined which produces the processing result $\in R^{FP}$ as well as the decision to keep the Area Registration $\in \{true, false\}$ based on the last Focused Processing Context $CX_{fp_x,it,3}$ and the last Area Registration $ar_{fp_x,it}$:

Definition 4.23 (Focused Situation Processing Result Query Function)**.**

$$\begin{aligned} Q_{\tau}^{FpResult} : \quad CX \times AR &\longrightarrow R^{FP} \times \{true, false\} \\ (CX_{fp_x,it,3}, ar_{fp_x,it}) &\longmapsto (R_{fp_x}^{FP}, keepAR) \end{aligned}$$

The function may also request to keep the Area Registration from the last iteration if the processing result states the investigated possible situation as a *FalseSituation* ($R_{fp_x}^{FP} = FalseSituation$). In this case keeping the Area Registration allows to prevent the creation of new Focused Situation Processing Instances for the *FalseSituation*.

4.6.3. Handling of Additional Situation Indications During an Ongoing Processing

The possible situation indication classification that takes place in processing Phase 2 may classify a raised indication as an *Additional Indication.* In this case, the indication is related to an ongoing Focused Situation Processing Instance. As such, the classification as *Additional Indication* is tied to a running Focused Situation Processing Instance fp_x.

In order to hand over the indication to the Focused Situation Processing Instance fp_x, the indication is injected into the set of additional indication events $E_{fp_x}^{AdditionalIndications}$ of the processing instance. The processing instance then incorporates them into the set of assigned indications during its preparation of the next iteration. Afterwards the actual usage of the indication is the responsibility of the scenario-specific parts of the processing logic.

4.6.4. Focused Processing Instance Collision-Handling

When a running Focused Situation Processing Instance fp_a requests a new Area Registration whose Locked Area would collide with the Locked Area of another Focused Situation Processing Instance fp_b, the processing model assumes that both tasks concern the same situation. Thus, a merge of the two processing tasks is required.

Several possibilities to handle the occurrence of such collisions exist. The processing model defines a simple mechanism which always terminates the Focused Situation Processing Instance fp_a that collided with the valid Area Registration of another Focused Situation Processing Instance fp_b. This process is defined in Subsection 4.6.4.1.

Aside from the mechanisms chosen here, other approaches could be considered like for example:

Replace both with one new Processing Task
: Create a new Focused Situation Processing Instance that replaces the two colliding ones. As a completely new task is started, the processing states of the colliding tasks will be lost or needs to be incorporated into the new instance in a scenario specific way. In order to create the new task, a suitable initial Locked and Focus Area needs to be found which would in its simplest form be the union of the Locked Areas of the colliding tasks and in the same way for the Focus Areas.

Merge Focused Situation Processing Tasks
: Merge the two processing tasks into one while keeping the processing states intact. Such a mechanism requires in-depth knowledge on the scenario that is being implemented by the tasks in order to decide how to merge the scenario and situation-specific processing states.

Aside from these extended approaches, the next subsection discusses the approach defined by the processing model.

4.6.4.1. Focused Situation Processing Collision-Handling Process

The processing model handles the collision of two Focused Situation Processing Instances fp_a and fp_b by dropping the processing instance fp_a whose Area Registration Request failed due to a collision with the already acquired Area Registration of the other processing instance fp_b.

As defined by the processing flow shown in Figure 4.6.2, the Focused Situation Processing Instance of the processing task fp_a that caused the collision is immediately halted. The other task fp_b participating in the collision, will also be stopped once its current iteration is finished and it tries to update its Locked Area and Focus Area registrations.

Once both Focused Situation Processing Instances have been halted for the execution of the merge, the following process is executed:

1. Release all Area Registrations of the losing Focused Situation Processing Instance fp_a that overlap with its last Area Registration Requests time frame. This removes all Area Registrations which would have been replaced if the Area Registration Request of fp_a would have been successful. As these registrations where thus considered obsolete by fp_a they can safely be removed so that the winning fp_b can claim them.

2. A scenario-specific merge function may then read processing state from the processing context $CX_{fp_a,itLast_{fp_a},3}$ of the last iteration $itLast_{fp_a}$ of the losing Focused Processing Instance fp_a and manipulate the processing context $CX_{fp_b,itLast_{fp_b},3}$ of the most recent iteration $itLast_{fp_b}$ of the surviving Focused Situation Processing Instance fp_b to incorporate interim processing results from the terminated processing instance.

3. The Area Registration Request for $fp_b$is executed as usual. If this leads to a successful registration, the processing instance continues with its next iteration. If the request also results in a collision, another collision-handling process is started.

The used scenario-specific Merging function can thus be defined as a function that takes two processing contexts as its input and generates a new processing context as result:

Definition 4.24 (Focused Situation Processing Collision-Handling Function)**.**

$$\begin{aligned} ColHandler_\tau : CX \times CX &\rightarrow CX \\ (CX_{fp_a,itLast_{fp_a},3}, CX_{fp_b,itLast_{fp_b},3}) &\longmapsto CX^{Merge}_{fp_b,itLast_{fp_b},3} \end{aligned}$$

Where fp_a is the Focused Situation Processing Instance that is to be terminated with $itLast_{fp_a}$ being its last iteration and fp_b the Focused Situation Processing Instance that survives the merge with $itLast_{fp_b}$ being the last iteration executed before it entered the merge process.

Collision Timing

Even though the focused situation processing tasks are synchronized based on the acquisition of Area Registrations, the processing model makes no guarantees which Focused Situation Processing Instance *first* claims its Area Registration. As a result, no guarantees can be made which one of two colliding Focused Situation Processing Tasks will be the one terminated by a merge.

4.6.5. Resulting Definition of the Focused Situation Processing Algorithm

Based on the discussions on the focused situation processing algorithm of the processing model can be defined as shown in Algorithm 3.

4.6.6. Synchronization Considerations

The Phase 3 Focused Situation Processing is intended to be executed for multiple Situations in parallel. The different FSP Instances are considered independent from each other as they concern separate situations. The only synchronization between the different FSP Instances is provided by the Area Registration mechanism.

In order to allow for the correct handling of colliding situations, multiple FSP Instances should not outrun each other but instead wait for slower FSP Instance to finish before they claim their Area Registrations for following time frames.

4.7. Conclusion

The chapter defined the focused processing model with its three processing phases (Figure 4.7.1). In order to be applicable for the processing of a certain scenario, it defines various functions that are specific to the scenario. The following chapter will therefore define a language for the specification of these functions as Scenario Processing Templates that can be used to specify processing tasks adhering to the model defined in this chapter.

Data: $e_\tau^{Indication}$, $FA_{e_\tau^{Indication}}$, $LA_{e_\tau^{Indication}}$, $tw_{e_\tau^{Indication}}$, k

$LA_{fp_x,1} \leftarrow LA_{e_\tau^{Indication}}$
$FA_{fp_x,1} \leftarrow FA_{e_\tau^{Indication}}$
$tf_{fp_x,1} \leftarrow tf_{e_\tau^{Indication}}$
$ar_{fp_x,1} \leftarrow (fpi_{x,1}, LA_{fp_x,1}, FA_{fp_x,1}, tf_{fp_x,1})$
$E_{fp_x,0}^{AssignedIndications} \leftarrow \{e_\tau^{Indication}\}$
$CX_{fp_x,0,3} \leftarrow Init_\tau^{FP}(k, ar_{fp_x,1}, E_{fp_x,1}^{AssignedIndications})$
$c \leftarrow true$, $it \leftarrow 1$ // it as iteration counter
while $c == true$ **do**
  $E_{fp_x,it}^{AssignedIndications} \leftarrow E_{fp_x,it-1}^{AssignedIndications} \cup E_{fp_x}^{AdditionalIndications}$
  $E_{fp_x}^{AdditionalIndications} \leftarrow \emptyset$
  $CX_{fp_x,it,1} \leftarrow Pre_\tau^{FP}(CX_{fp_x,it-1,3}, k, ar_{fp_x,it}, E_{fp_x,it}^{AssignedIndications})$
  $SPT_{fp_x,it}^{FpIteration} \leftarrow Build_\tau^{FpIteration}(CX_{fp_x,it,1}, ar_{fp_x,it}, k)$ // Stream Processing Topology Build
  $CX_{fp_x,it,2} \leftarrow SPT_{fp_x,it}^{FpIteration}(\Pi(FA_{fp_x,it}), CX_{fp_x,it,1})$ // Stream Processing Execution
  $CX_{fp_x,it,3} \leftarrow Post_\tau^{FP}(CX_{fp_x,it,2}, k, ar_{fp_x,it}, E_{fp_x,it}^{AssignedIndications})$ // Post Processing
  $E_{fp_x,it}^{InterimResult} \leftarrow InterimResultEventGen_\tau(CX_{fp_x,it,3})$
  if $E_{fp_x,it}^{InterimResult} \neq \emptyset$ **then**
    Output: $E_{fp_x,it}^{InterimResult}$
  end
  if $C_\tau^{FpTerm}(CX_{fp_x,it,3}) = true$ **then** // Termination
    $c \leftarrow false$
    $(R_{fp_x}^{Fp}, keepAR) \leftarrow Q_\tau^{FpResult}(CX_{fp_x,it,3}, ar_{fp_x,it})$
    if $keepAR = false$ **then** // Do not keep Area Registration
      $AR_\tau \leftarrow ReleaseAreaRegistration(fpi_{fp_x,it}, LA_{fp_x,it}, FA_{fp_x,it}, tf_{fp_x,it}, AR_\tau)$
    end
    Output: $R_{fp_x}^{Fp}$ as final result
    // Publish Final Result
  else
    $(LA_{fp_x,it+1}, FA_{fp_x,it+1}, tf_{fp_x,it+1}) \leftarrow Q_\tau^{LaFaTf}(CX_{fp_x,it,3}, tf_{fp_x,it}, k)$ // Prepare for Next Iteration
    // Locked and Focus Area Acquisition
    $(collisions, AR_\tau) \leftarrow RegisterArea_\tau(fpi_{fp_x,it+1}, LA_{fp_x,it+1}, FA_{fp_x,it+1}, tf_{fp_x,it+1}, AR_\tau)$
    if $collisions = \emptyset$ **then**
      ; // Locked and Focus Area Acquisition Successfull
      if $tf_{fp_x,it} = tf_{fp_x,it+1}$ **then**
        ; // New AR superseeds old AR
        $AR_\tau \leftarrow ReleaseAreaRegistration(fpi_{fp_x,it}, LA_{fp_x,it}, FA_{fp_x,it}, tf_{fp_x,it}, AR_\tau)$
      end
    else
      $c \leftarrow false$, **Output:** $CX_{fp_x,it,3}$ for merging via Collision Handling Function
    end
  end
  $it \leftarrow it + 1$
end

Algorithm 3: Focused Situation Processing Algorithm

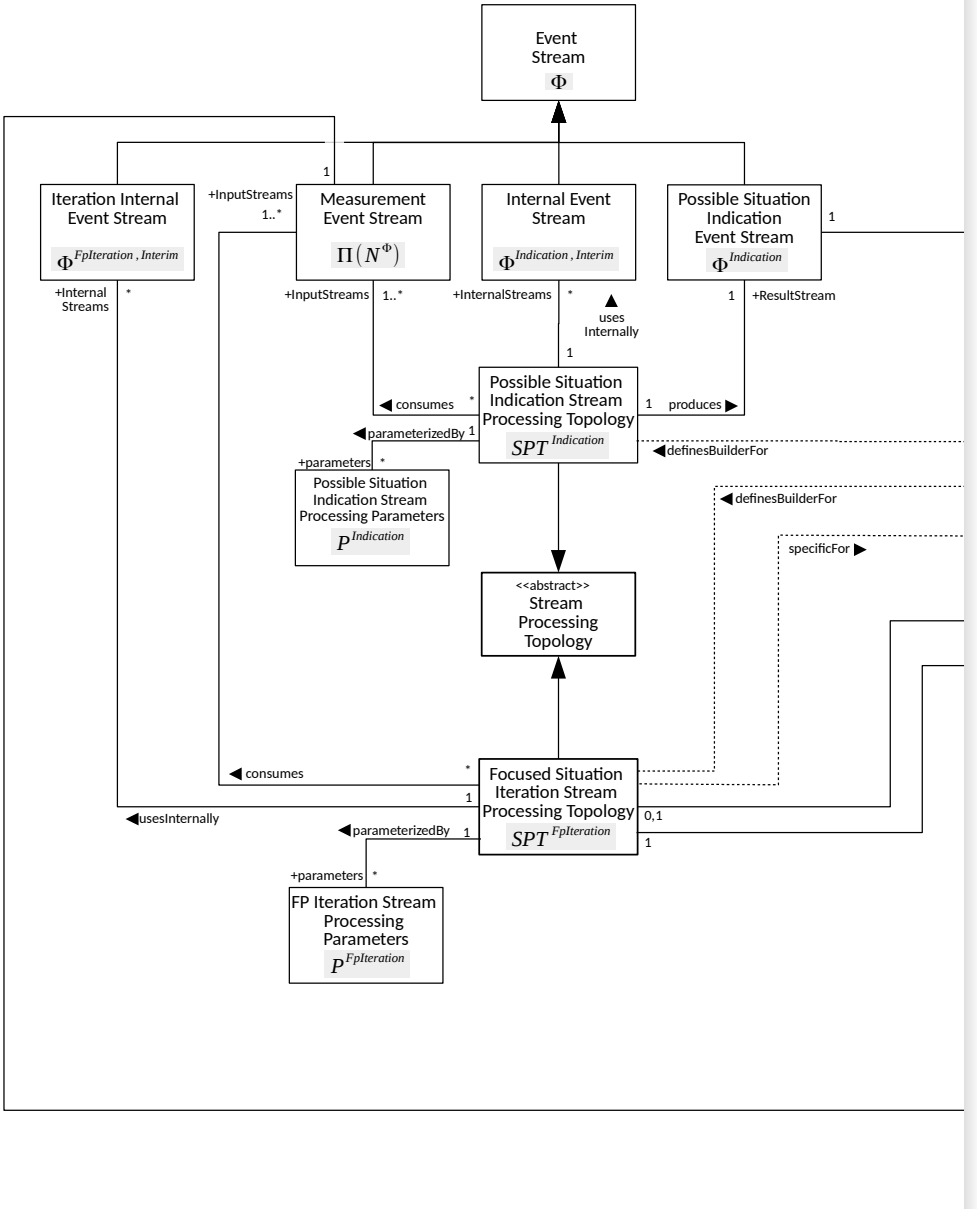
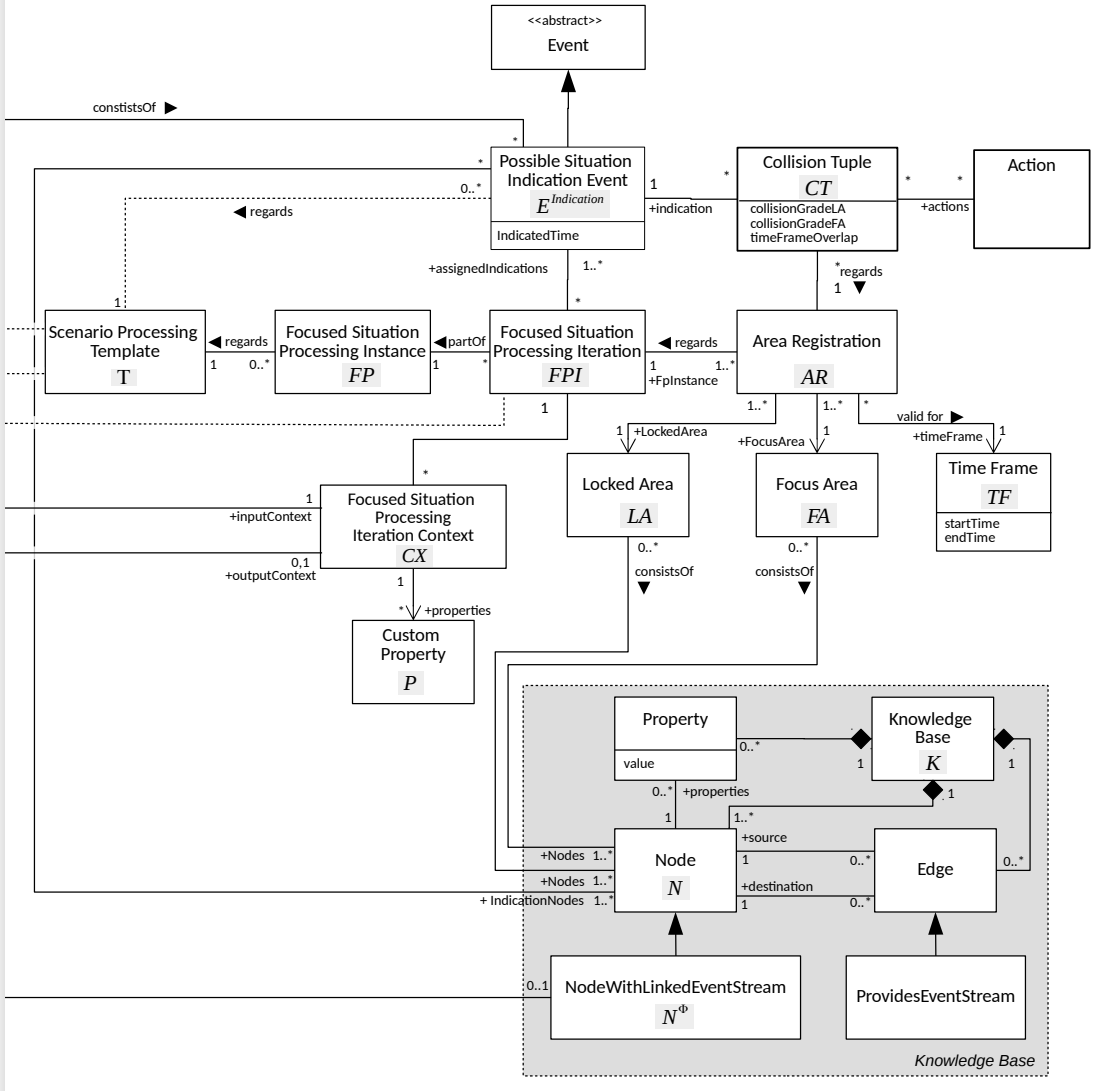


Figure 4.7.1.: Overview of the whole Processing Model


<<abstract>>
Event
constistsOf
Possible Situation Indication Event
E Indication
IndicatedTime
Collision Tuple
CT
collisionGradeLA
collisionGradeFA
timeFrameOverlap
Action
+indication
+actions
regards
+assignedIndications
Scenario Processing Template
T
Focused Situation Processing Instance
FP
partOf
Focused Situation Processing Iteration
FPI
Area Registration
AR
+FpInstance
+LockedArea
+FocusArea
valid for
+timeFrame
Time Frame
TF
startTime
endTime
Locked Area
LA
Focus Area
FA
consistsOf
Focused Situation Processing Iteration Context
CX
+inputContext
+outputContext
+properties
Custom Property
P
Property
value
Knowledge Base
K
Node
N
Edge
+source
+destination
+Nodes
+ IndicationNodes
NodeWithLinkedEventStream
N Φ
ProvidesEventStream
Knowledge Base

5. Language Definition

Contents

In order to instruct a processing system that implements the Focused Situation Processing Model defined in the previous chapter, the definition of several scenario-specific functions and parameters is required to configure a processing system for a concrete scenario. In order to allow for these definitions, the *Scenario Processing Template Language* (SPTL) was designed which allows the specification of these properties as a „Scenario Processing Template“. The language and its interpretation is presented in this chapter while the prototype discussed in the next chapter provides an implementation of the language and the processing model.

5.1. Overview

5.1.1. Scenario Processing Template Structure

A scenario processing template contains all scenario-specific information to parameterize a processing system for a scenario (e.g. How to detect a cloud and how to determine its trajectory). The template is divided into a preamble and three blocks which resemble the three major phases defined in the processing model (Section 4.1):

```
name "SmartGridCloudTracking"

PossibleSituationIndication {
```

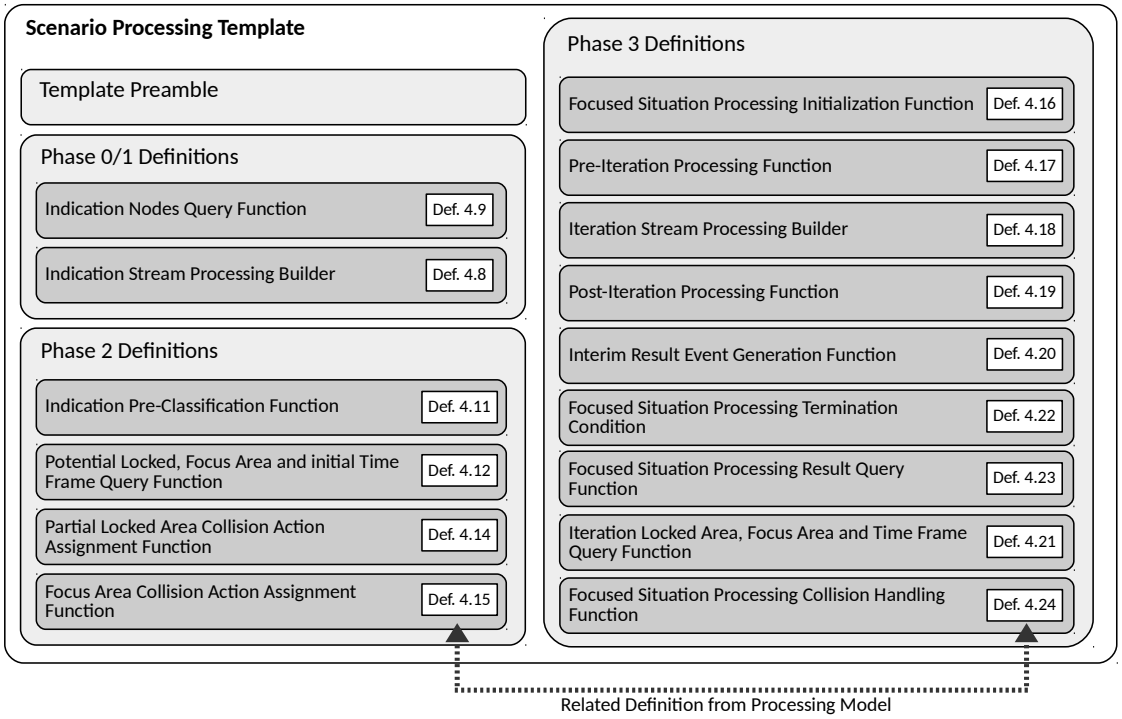


Figure 5.1.1.: Structure of a Scenario Processing Template with references to the corresponding definitions from the Processing Model.

```
  ...
}
FocusedSituationProcessingInitialization {
  ...
}
FocusedSituationProcessing {
  ...
}
```

Each block contains the specifications required for the setup and execution of the corresponding phase (Figure 5.1.1).

In EBNF the processing template is defined as follows, where the used symbols are defined and discussed in the remainder of this chapter:

⟨*ScenarioProcessingTemplate*⟩ ::= ⟨*TemplatePreamble*⟩
⟨*PossibleSituationIndication*⟩
⟨*FocusedProcessingInitialization*⟩
⟨*FocusedSituationProcessing*⟩

The complete EBNF definition of the SPTL is summarized in Appendix A.

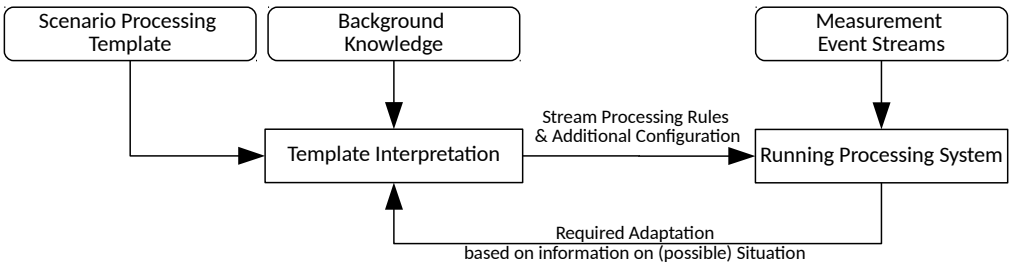


Figure 5.1.2.: High level view of the Scenario Processing Template interpretation.

5.1.1.1. Embedded Languages

Aside from general configuration parameters, the template also needs to specify several knowledge retrieval functions, context manipulation functions as well as the stream processing builder functions. As already several languages for knowledge retrieval as well as general purpose scripting languages exist, the SPTL embeds three existing languages for such purposes (SPARQL, Drools and MVEL):

- In order to provide knowledge query support, the template uses knowledge query statements which embed a **SPARQL** [Gro13] query fragment that is used with the background knowledge repository.
- The specification of stream processing builder functions is based on an own language that describes in a procedural form how the actual stream processing rules should be generated. However, the rules themselves are specified in the **Drools Rule Language** as used by the rule engine employed in the prototype.
- For the specification of processing context manipulation functions as well as conditions, the SPTL uses the **MVFLEX Expression Language** (**MVEL**) [MVE] which is a general purpose Java Virtual Machine based scripting language and is already used as part of the Drools Rule Language.

5.1.2. Template Interpretation

The template on purpose omits any information on the concrete system that is the source of the measurement data (e.g. the set of available solar panels and their actual geographical position). The information on the monitored system instead has to be provided by the background knowledge repository.

Once a processing system is instantiated, the scenario processing template is combined with the background knowledge for the actual system that is to be monitored in order to provide the correct processing setup (Figure 5.1.2).

5.2. General Elements of the Template Language

5.2.1. Variables

The processing template language supports variables for example to store retrieved background knowledge or interim processing results. Within the template variables are prefixed with „$$“:

⟨*VAR*⟩ `::= '$$' [a-zA-Z0-9_]+`

5.2.2. Embedded Language: SPARQL

In order to access the background knowledge from within the Scenario Processing Templates, SPARQL Queries [Gro13] can be used to query the background knowledge and to assign the result to variables (e.g. define a set of nodes that is to be monitored for possible situation indications) or as parameters for the processing model (e.g. to specify a Focus Area).

Within the template, SPARQL queries are always specified as follows:

⟨*SPARQL*⟩ `::= 'from' 'sparql'` ⟨*STRING*⟩

Instead of the specification of the complete SPARQL query, only the condition part of the query needs to be specified. The condition is automatically extended to the full query before execution.

The specified SPARQL query condition must always provide the *?VALUE* variable which is used as the result of the query. All other result variables are ignored. Furthermore, a query may contain placeholders which refer to variables in the current context of the Scenario Processing Template. If such placeholders exist, they are replaced before the execution of the query with the current value of the variable from the context where the query was specified. Such embedded variables can be specified as *$${{varName}}* like for example:

```
from sparql "?VALUE some:relation $${{indicatedTime}}"
```

If the embedded variables value contains a set of elements, the query is automatically expanded so that it queries for all values of the variable separately and returns the union of the results.

All SPARQL queries are only allowed to retrieve data from the background knowledge base *without* making any changes to the knowledge base.

5.2.3. Embedded Language: MVEL

The MVEL language is used in many places in the template in order to provide a flexible definition of conditions and scenario-specific functions. MVEL is a Java inspired scripting

language which is also used within DROOLS and was thus chosen as scripting language for the SPTL. The MVEL Language Guide [MVE] provides an introduction to the MVEL language.

Whenever MVEL is used in the template, the MVEL statements must be encapsulated in *[MVEL]* and *[/MVEL]* :

⟨*MVEL*⟩ ::= '[MVEL]'.*?'[/MVEL]'

MVEL statements can access variables from the scope of the Scenario Processing Template via the „$$“ prefix. In the Focused Situation Processing phase, MVEL expressions are also allowed to edit existing variables and define new variables in the Focused Situation Processing Context (Subsection 5.6.1). Except for this, MVEL expressions must be side-effect free, thus they are not allowed to modify the state of the processing system other than explicitly allowed for the current expression. In general, they are not allowed to create and publish their own events and must not make any assumptions on their execution environment other than explicitly specified for the current function. In particular, they are not allowed to access external data sources (local or remote) other than the defined background knowledge base.

5.2.3.1. Access to the Knowledge Base from MVEL

In order to provide access to the background knowledge base from MVEL, two functions are provided via a globally available processing context „CONTEXT“ which allows the execution of complete SPARQL queries:

```
[MVEL]                                                          1
$$result1 = CONTEXT.querySet("sparql query");                   2
$$result2 = CONTEXT.queryScalar("sparql query");                3
[/MVEL]                                                         4
```

The queries issued are *not allowed* to modify the contents of the background knowledge base.

5.2.3.2. Domain Specific Functions

In order to provide complex domain- or scenario-specific functions, which would be completely to develop and test in MVEL, custom domain-specific functions may be provided as static Java methods which can then be called from within MVEL statements.

The domain-specific function must adhere to the same rules as MVEL statements. If the function needs access to the current processing context, the context may be provided as a parameter to the function. Via the context, the domain-specific function may access the background knowledge.

One example for such a function is the clustering of solar panel nodes based on their geographical position as needed by the cloud tracking scenario (Appendix B.1.1).

5.2.4. Embedded Language: DROOLS

The Drools language allows the specification of stream processing rules for the JBoss Drools Fusion Rule Engine and is used for this purpose in the Situation Processing Templates (See Section 3.2.3 for an example rule).

As the actual Stream Processing rules need to be specific for the situation that is to be investigated, the Stream Processing Templates also only contain the Stream Processing Rules as templates. Such rule templates contain placeholders that need to be inserted in order to generate the actual processing rule for the investigation of a specific situation. This generation process is implemented by the Stream Processing Builder function which is discussed in Section 5.7. Within the Scenario Processing Template, Drools rule templates are always encapsulated by *[DROOLS_ TEMPLATE]* and *[/DROOLS_ TEMPLATE]* :

⟨*DROOLS*⟩ ::= '[DROOLS_TEMPLATE]'.*?'[/DROOLS_TEMPLATE]'

Variables can be embedded into the DROOLS templates by specifying them as *$${{variableName}}*. Variables can only be directly embedded if the variable does not contain a set of elements but only a single element.

5.3. Scenario Processing Template Preamble

The scenario processing template preamble contains general configuration attributes regarding the template, like an identifier of the template as well as optional prefixes for the embedded SPARQL and DROOLS language parts that are to be used by the run-time system in addition to its own definitions:

⟨*TemplatePreamble*⟩ ::= ⟨*TemplateName*⟩ ⟨*DroolsPrefix*⟩? ⟨*SPARQLPrefix*⟩?

⟨*TemplateName*⟩ ::= 'name' ⟨*STRING*⟩ ';'

⟨*DroolsPrefix*⟩ ::= 'drools prefix' ⟨*STRING*⟩ ';'

⟨*SPARQLPrefix*⟩ ::= 'sparql prefix' ⟨*STRING*⟩ ';'

If no prefixes are specified, the run-time system has to provide sensible defaults which for the SPARQL prefix define basic namespaces and for the Drools prefix specify basic imports needed for a template to work with the run-time system.

5.4. Possible Situation Indication Processing Specification

The possible situation indication processing specification consists of two parts: (1) The selection of nodes to monitor for situation indications as specified in Definition 4.8 and (2) the specification of a stream processing function builder that can be used together

Variable Name	Access	Contents	Defined in Model
$$indicationNodes	Read Only	The set of nodes for which a possible situation indication should be set up.	Variable *indicationNodes* in Algorithm 1.

Table 5.4.1.: Variables available to the Stream Processing Builder of the Possible Situation Indication Processing Specification.

with selected nodes by (1) to generate the actual event stream processing function for the situation indication processing as specified in Definition 4.8:

⟨*PossibleSituationIndication*⟩ ::= 'PossibleSituationIndication' '{'
    ⟨*IndicationNodesQueryFunction*⟩
    ⟨*IndicationStreamProcesssingBuilder*⟩
    '}'

For the selection of the nodes which should be monitored, Definition 4.9 specifies a query function that retrieves the relevant nodes from the knowledge base. In the processing template, this function is specified as a SPARQL query that results in the set *$$indicationNodes*:

⟨*IndicationNodesQueryFunction*⟩ ::= '$$indicationNodes' ⟨*SPARQL*⟩ ';'

Based on the query results, the actual Event Stream Processing Topology that detects possible situations is generated. The topology generation is specified in the template within the *IndicationStreamProcessingBuilder* block which uses the stream processing build mechanisms specified in Section 5.7. This builder block represents the Situation Indication Stream Processing Builder function as defined in Definition 4.8:

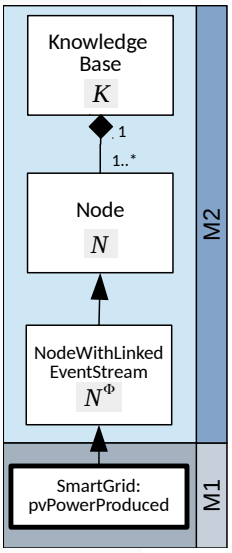
⟨*IndicationStreamProcesssingBuilder*⟩ ::= 'IndicationStreamProcessingBuilder' '{'
    ⟨*StreamProcessingBuilder*⟩
    '}'

The Stream Processing Builder function has access to the variable *$$indicationNodes* in order to create a suitable stream processing function (Table 5.4.1).

Example 11

Based on these definitions, the following listing shows a simplified example of the possible situation indication processing specification for the cloud tracking scenario. The query fragment in Line 2 retrieves a set of nodes from the knowledge base. All retrieved nodes will be of the type *smartgrid:PVPowerProduced,* a subclass of the type *NodeWithLinkedEventStream* which is part of the processing model knowledge base (Figure on the right hand side). The resulting nodes are assigned to the variable *$$indicationNodes* which is then used within the *IndicationStreamProcessingBuilder.*

```
1 PossibleSituationIndication {
2   $$indicationNodes from sparql "?VALUE rdf:type smartgrid:
        PVPowerProduced"
3
4   IndicationStreamProcessingBuilder {
5     foreach $$indicationNodes as $$pv {
6       rule [DROOLS_TEMPLATE]  ... from entry-point "$${{pv}}"
            ... publishIndication( "$${{pv}}"  ) ...  [/
            DROOLS_TEMPLATE] publishes indications;
7     }
8   }
9 }
```



Excerpt from Fig. 4.2.2.

5.5. Focused Situation Processing Initialization

The Focused Situation Indication Processing Initialization (Phase 2) of the processing model defines the following four functions and properties as scenario-specific (Subsection 4.5.1ff). Thus, their definition needs to be supported by the template language:

- Indication Pre-Classification threshold value: $p_{\tau}^{Indication,Dup}$.
- Potential Locked and Focus Area Query and Time Frame determination Function: $Q_{\tau}^{PotentialLaFa}$.
- Partial Locked Area Collision Action Assignment Function: $ActionAssignment_{\tau}^{PartialLaOverlap}$.
- Focus Area Collision Action Assignment Function: $ActionAssignment_{\tau}^{FaOverlap}$.

Within the template, the following constructs allow the definition of the above functions and properties which are all part of the *FocusedProcessingInitialization* block:

⟨*FocusedProcessingInitialization*⟩ `::= 'FocusedProcessingInitialization' '{'`
⟨*IndicationPreClassificationThreshold*⟩
⟨*PotentialLockedFocusAreaInitialTimeFrameQueryFunction*⟩
⟨*PartialLockedAreaCollisionActionAssignmentFunction*⟩
⟨*FocusAreaCollisionActionAssignmentFunction*⟩

Variable Name	Access	Contents	Defined in Model
$$indicatedNodes	Read Only	The set of nodes of the current indication event.	Projection from the Possible Situation Indication Event $e_{\tau,i}^{Indication}$ from Algorithm 2: $\pi_{indicationNodes}(e_{\tau,i}^{Indication})$
$$indicatedTime	Read Only	The indicated time of the current indication event.	Projection from the Possible Situation Indication Event $e_{\tau,i}^{Indication}$ from Algorithm 2: $\pi_{indicatedTime}(e_{\tau,i}^{Indication})$

Table 5.5.1.: Variables available in the scope of the Focused Situation Processing Initialization definition.

```
    '}'
```

Within the scope of the Focused Processing Initialization's execution, the contents of the current indication event are available through variables as defined in Table 5.5.1.

5.5.1. Indication Pre-Classification Function

With regard to the pre-classification of raised possible situation indications discussed in Subsection 4.5.1, duplicate indication events are detected based on a scenario-specific threshold $p_{\tau}^{Indication,Dup} \in \mathbb{N}$ (Definition 4.11). Within the processing template, this threshold is specified as a time duration (e.g. 30 seconds) via the *duplicationThreshold* property[1]:

⟨*IndicationPreClassificationThreshold*⟩ `::= 'duplicationThreshold'` ⟨*TIME_DURATION*⟩ `';'`

It is also allowed to set the duration to zero which deactivates the Pre-Classification mechanism.

5.5.2. Potential Locked, Focus Area and initial Time Frame Query Function

The definition of the potential Locked Area, Focus Area and the initial time frame is based on three separate statements:

⟨*PotentialLockedFocusAreaInitialTimeFrameQueryFunction*⟩ `::=` ⟨*PotentialLockedArea*⟩
⟨*PotentialFocusArea*⟩
⟨*InitialTimeFrame*⟩

5.5.2.1. Potential Locked Area and Focus Area Query

For the collision detection, a potential Locked Area needs to be derived in a scenario-specific way. Further a potential Focus Area needs to be derived, also in a scenario-specific

[1] TIME_DURATION is defined in Appendix A

way for each indication that results in a new Focused Situation Processing Instance. As such, the two corresponding query functions are part of the template definition.

The potential locked area and Focus Area query part of the function $Q_{\tau}^{PotentialLAFA}$ (Definition 4.12) is defined by two statements, one for the potential Locked Area and one for the potential Focus Area.

⟨*PotentialLockedArea*⟩ ::= 'potentialLockedArea' (⟨*VAR*⟩ | ⟨*SPARQL*⟩) ';'

⟨*PotentialFocusArea*⟩ ::= 'potentialFocusArea' (⟨*VAR*⟩ | ⟨*SPARQL*⟩) ';'

Both statements allow the specification of the corresponding area in two different ways:

1. Based on the contents of a variable. For example to set the potential Locked Area to the contents of the variable *$$indicationNodes*:

```
potentialLockedArea $$indicatedNodes;
```

2. Based on the result of a SPARQL query that is expanded based on the given nodes contained in an embedded variable like *$$indicationNodes*. The query expansion mechanism itself is discussed in Subsection 5.2.2.

```
potentialLockedArea from sparql "?NAME some:relation $${{indicatedNodes}}";
```

5.5.2.2. Timing Specification

The initial time frame determination is defined as part of the function $Q_{\tau}^{PotentialLAFA}$ (Definition 4.12) specified in the template as follows:

⟨*InitialTimeFrame*⟩ ::= 'initialTimeFrame' 'startsAt' (⟨*VAR*⟩ | ⟨*MVEL*⟩)
'withDurationOf' (⟨*TIME_DURATION*⟩ | ⟨*MVEL*⟩) ';

The initial time frame is defined by specifying its start time and its duration. The start value can be set to the value of the available variable *$$indicatedTime* or to another value by defining an MVEL expression which provides an Unix time stamp in seconds. The duration of the time frame can be defined as a static value or any other *positive* scalar value by defining an MVEL expression which provides the duration in seconds.

Example 12

For example for the cloud tracking scenario, the following statements can be used (as discussed in Subsection 7.3) in order to set the initial Locked Area to the set of nodes contained in the indication event (Line 1). To determine the initial Focus Area as a set of nodes within a certain geographical distance from the indicated nodes, a suitable SPARQL query is used (Lines 2 - 12). Finally, the initial Time Frame is defined to begin with the indicated time (*$$indicatedTime*) with a fixed duration of 300s (Line 14).

```
potentialLockedArea $$indicatedNodes;
potentialFocusArea from sparql "
  $${{indicatedNodes}} smartgrid:hasLocation ?LOC1.
  ?NAME  smartgrid:hasLocation ?LOC2.
  ?LOC1  smartgrid:hasLat ?LAT1.
  ?LOC2  smartgrid:hasLat ?LAT2.
  ?LOC1  smartgrid:hasLon ?LON1.
  ?LOC2  smartgrid:hasLon ?LON2.
  FILTER ( ?LAT1+0.0041 > ?LAT2 ).
  FILTER ( ?LAT1-0.0041 < ?LAT2 ).
  FILTER ( ?LON1+0.0041 > ?LON2 ).
  FILTER ( ?LON1-0.0041 < ?LON2 ). ";

initialTimeFrame startsAt $$indicatedTime withDurationOf 300s ;
```

5.5.3. Collision Action Assignment

After the Pre-Classification is finished, the processing model defines a collision detection in order to prepare for the classification for the remaining indications. The collision detection itself is scenario independent (Definition 4.13) and does not need any special properties in the template. The handing of found collisions is, however, scenario-specific as discussed in Subsection 4.5.4.

For the collision classification, the processing model defines two scenario-specific functions:

- $ActionAssignment_{\tau}^{PartialLaOverlap}$ for the case were at least one partial overlap with the *Locked Area* of an already instantiated Focused Situation Processing Instance occurred (Definition 4.14) and
- $ActionAssignment_{\tau}^{FaOverlap}$ for collisions where the only overlap occurred with one or more *Focus Areas* of already instantiated Focused Situation Processing Instances (Definition 4.15).

The scenario-specific behavior of these two functions can be specified in two different ways in the template:

Option 1: Define the two collision-handling functions by specifying two MVEL statements, each representing one of the functions.

Option 2: Define one or more collision action rules which together define the functionality of the two functions.

Thus, the template language allows two mutually exclusive definitions for the scenario-specific collision-handling which are discussed in the following two subsections:

⟨*PartialLockedAreaCollisionActionAssignmentFunction*⟩ ::= (⟨*PartialLAcollisionFunction*⟩
| ⟨*CollisionRules*⟩)

⟨*FocusAreaCollisionActionAssignmentFunction*⟩ ::= (⟨*FaCollisionFunction*⟩ | ⟨*CollisionRules*⟩)

If neither option is used and thus no scenario-specific definition of the collision-handling function is given, a default rule is used. The default rule defines that for collisions with a partial Locked Area overlap, no action is assigned for the indication and the indication is thus ignored. If no partial Locked Area overlap occurred, but one or more partial or complete Focus Area overlaps, the processing system triggers the instantiation of a new Focused Situation Processing Instance for the indicated possible situation.

5.5.3.1. Option 1: MVEL based collision Function definition

The collision-handling can be specified by defining the two scenario-specific functions $ActionAssignment_{\tau}^{PartialLaOverlap}$ and $ActionAssignment_{\tau}^{FaOverlap}$ declared by the processing model (Definitions 4.14 and 4.15) as MVEL statements:

⟨*PartialLAcollisionFunction*⟩ ::= 'partialLACollision' ⟨*MVEL*⟩ ';'

⟨*FaCollisionFunction*⟩ ::= 'FACollision' ⟨*MVEL*⟩ ';'

Both statements have access to the set of all occurred collisions as Collision Tuples (Appendix A.2.1) via the variable *$$collisions*. In order to assign an action for each collision from *$$collisions*, the following function is provided:

```
void CONTEXT.setAction(CollisionTuple collision, CollisionAction action);
```

Actions that can be assigned for each of the tules are „AddToExisting“ and „NoAction“ as specified by the *CollisionAction* Enum (Subsection A.2.1.1) based on the processing models Definitions 4.14 & 4.15. If the *setAction* function is not called for a collision tuple from *$$collisions*, this is equivalent to calling it with the „NoAction“ action for this Collision Tuple.

Furthermore, the Focus Area Collision Function is allowed to request the creation of a new Focused Situation Processing Instance by calling the following provided function (see Definition 4.15):

```
void CONTEXT.requestStartNew();
```

All variables available to the functions are specified in Table 5.5.2.

Example 13

For example for the cloud tracking scenario, the collision-handling can be defined by the following functions which will only start a new Focused Situation Processing Instance for a raised indication, if the indication causes no collision at all or only

Variable Name	Access	Contents	Defined in Model
Also all variables specified in Table 5.5.1.			
$$collisions	Read Only	The set of Collision Tuples for the current indication event.	The set $CT_{e_{\tau,i}^{Indication}}$ defined in Algorithm 2.
$$potentialLA	Read Only	The set of nodes that resemble the potential Locked Area for the current indication event.	The set $LA_{e_{\tau,i}^{Indication}}$ defined in Algorithm 2.
$$potentialFA	Read Only	The set of nodes that resemble the potential Focus Area for the current indication event.	The set $FA_{e_{\tau,i}^{Indication}}$ defined in Algorithm 2.
$$initialTF	Read Only	The initial Time Frame for the current indication event.	The tuple $tf_{e_{\tau,i}^{Indication}}$ defined in Algorithm 2.

Table 5.5.2.: Variables available in the scope of the Collision Action Assignment.

a *partial* collision with the Focus Area of an existing Focused Situation Processing Instance (as discussed in Subsection 7.3):

```
// noAction if a partial Locked Area collision occured
partialLACollision [MVEL][/MVEL];

// only start a new Focused Processing if no 100% Focus Area collision
    occured
FACollision [MVEL]
  fullFACollisionFound = false;
  foreach ( col : $$collisions ) {
    // check for 100% FA overlap
    if( col.getGradeFa() == $$potentialLA.size() )
      fullFACollisionFound = true;
  }
  if( !fullFACollisionFound ){
    CONTEXT.requestStartNew();
  }
[/MVEL];
```

5.5.3.2. Option 2: Collision Action Rules

In order to allow for a simpler definition of the collision-handling behavior in cases where no overall view over all collisions for the current indication event is required, the template allows for a rule based specification of the scenario-specific collision handing actions.

Each rule may specify one or more conditions on the Locked and Focus Area overlap grades when it should match. When a rule matches the properties of a collision, one or more actions that where specified for the rule are assigned for handling the current collision. The order in which the rules are specified in the template is used as processing order of the rules where the rule processing for each separate collision stops once a rule matches for this collision. The rules are executed *separately* for each collision and thus do not allow the specification of a single action based on *all* occurred collisions for the current indication.

The actions can be chosen from the following set:

startNew
: Request the creation of a new Focused Situation Processing Instance for the indicated possible situation. This action can be assigned multiple times but will only be executed once. However, the conditions defined in Subsection 4.5.4 for a partial Locked Area overlap require that the action *startNew* is only possible if *no* partial Locked Area collision was detected for the received indication. If the action is assigned by any rule in such a case, the action is not executed.

addToExisting
: Request the assignment of the indication event to the Focused Situation Processing Instance responsible for the current collision so that this already running instance may handle the indication event.

noAction
: Do not take any action for handling this collision.

preventNew
: Request that no new Focused Situation Processing Instance is to be started for this indication event even if for another collision the *startNew* action was assigned.

stopActionExecution
: Request the stop of the action processing for all collisions that where not yet handled by the rule processing.

As the *stopActionExecution* action requires a deterministic ordering of the collision rule matching process. To ensure this, the set of collisions is sorted based on their collision grade (first on the Locked Area overlap grade then on the Focus Area overlap grade) were the collisions with the highest grade are processed first. This allows a high ranked collision to prevent the rule processing of all other collisions with a lower grade than its own.

Within the template, collision action rules can be specified as follows[2]:

⟨*CollisionRules*⟩ ::= (⟨*CollisionRule*⟩)*

⟨*CollisionRule*⟩ ::= 'collisionAction' ⟨*CollisionAction*⟩ (',' ⟨*CollisionAction*⟩)*
'if' ⟨*Condition*⟩ ('and' ⟨*Condition*⟩)* ';'

⟨*Condition*⟩ ::= ('LA'|'FA') 'overlap' ('⟨' | '⟩' | '==' | '⟨=' | '⟩=')
⟨*PERCENTAGE*⟩

⟨*CollisionAction*⟩ ::= 'startNew' | 'addToExisting' | 'noAction' |
'preventNew' | 'stopActionExecution'

[2]PERCENTAGE is defined in Appendix A

Example 14

For example for the cloud tracking scenario, the collision-handling can be defined by the following rules which will only start a new Focused Situation Processing Instance for a raised indication, if the indication causes no collision at all or only a *partial* collision with the Focus Area of an existing Focused Situation Processing Instance (as discussed in Subsection 7.3)[a]:

```
collisionAction preventNew if FA overlap = 100%;
collisionAction startNew if FA overlap < 100%;
```

[a] A separate rule that prevents the creation of a new Focused Situation Processing Instance in case of a partial Locked Area overlap is not required, as the creation of a new instance is not allowed by the processing model in this case (Subsection 4.5.4).

5.6. Focused Situation Processing

The Focused Situation Processing part of the language defines all aspects relevant for an ongoing Focused Situation Processing Instance (Phase 3 of the processing model). This includes Pre- and Post-Processing steps as well as the result publication and the termination rules. Further it contains the specification of the iteration stream processing builder function.

Within the template, all situation-specific processing configuration is given in the *FocusedSituationProcessing* block:

⟨*FocusedSituationProcessing*⟩ ::= 'FocusedSituationProcessing' '{'
⟨*FocusedSituationProcessingInitializationFunction*⟩
⟨*PreIterationProcessingFunction*⟩
⟨*IterationStreamProcessingBuilder*⟩
⟨*PostIterationProcessingFunction*⟩
⟨*InterimResultEventGenerationFunction*⟩
⟨*FocusedSituationProcessingTerminationConditionAndTerminationResult*⟩
⟨*IterationLockedAreaFocusAreaTimeFrameQueryFunction*⟩
⟨*FocusedSituationProcessingCollisionHandlingFunction*⟩
'}'

5.6.1. Focused Situation Processing Context

As specified in Subsection 4.6.2.1 of the processing model definition, the Focused Situation Processing context provides access to several properties defined by the processing model as well as freely definable template specific properties. In order to provide this functionality, the *processing context* is defined in the template as a set of key value pairs with a number of reserved keys for framework specific values as specified in Table 5.6.1.

The processing context is available to the Focused Situation Processing Instance during all MVEL based executions as well as during the Stream Processing Builder Function

Variable Name	Access	Contents	Defined in Model
$$focusArea	Read Only	The set of nodes forming the Focus Area of the current iteration	The tuple $FA_{fp_x,it}$ defined in Algorithm 3 with it as the current iteration.
$$lockedArea	Read Only	The set of nodes forming the Locked Area of the current iteration	The tuple $LA_{fp_x,it}$ defined in Algorithm 3 with it as the current iteration.
$$timeFrame	Read Only	The time frame of the current iteration	The tuple $tf_{fp_x,it}$ defined in Algorithm 3 with it as the current iteration.
$$indications	Read Only	The set of indication events assigned to this Focused Situation Processing Instance	The set $E_{fp_x,it}^{AssignedIndications}$ defined in Algorithm 3 with it as the current iteration.
$$iterationCounter	Read Only	The number of the current iteration	The variable it defined in Algorithm 3.
$$indicatedNodes	Read Only	The set of nodes of the indication event that triggered the Focused Situation Processing Instance	Projection from the Possible Situation Indication Event from Algorithm 3 that triggered the processing instance: $\pi_{indicationNodes}(e_\tau^{Indication})$
$$indicatedTime	Read Only	The indicated time of the indication event that triggered the Focused Situation Processing Instance	Projection from the Possible Situation Indication Event from Algorithm 3 that triggered the processing instance: $\pi_{indicatedTime}(e_\tau^{Indication})$

Table 5.6.1.: Variables available in the scope of the Focused Situation Processing.

execution. Further, during the Focused Situation Iteration Stream Processing, the context can be accessed from within the event stream processing functions.

As defined in Subsection 4.6.2.2, the processing context is kept across the boundary of a single Focused Situation Processing Iteration and thus allows the transfer of information from one iteration to the next.

5.6.2. Focused Situation Processing Initialization Function

In order to initialize the very first Focused Situation Processing Context, the processing model defines the initialization function $Init_\tau^{FP}$ (Definition 4.16). Within the template, this function has to be specified as an MVEL expression as follows:

⟨*FocusedSituationProcessingInitializationFunction*⟩ `::= 'contextInitialization'` ⟨*MVEL*⟩ `';'`

The function is allowed to populate the Focused Situation Processing Context by defining variables with the „$$“ prefix which are automatically assigned to the initial Focused Situation Processing Context.

In order to create a suitable context for the assigned possible situation, the function has access to the variables specified in Table 5.6.1 which contain the properties for the first iteration as they were determined during the Phase 2 processing. Furthermore, access to the knowledge base is possible as defined in Section 5.2.3.

5.6.3. Pre-Iteration Processing Function

In a similar way to the general context initialization, an iteration context initialization function Pre_{τ}^{FP} (Definition 4.17) needs to be specified. The function is executed at the beginning of each iteration in order to set up its processing context for the next iteration in a scenario-specific way. The new processing context is pre-populated based on values from the previous processing context $CX_{fp_x,it-1,3}$ as specified in Subsection 4.6.2.4. Within the processing template, the initialization function is specified in MVEL as follows:

⟨*PreIterationProcessingFunction*⟩ `::= 'preIterationProcessing'` ⟨*MVEL*⟩ `';'`

5.6.4. Iteration Stream Processing Builder

The definition of the builder function follows the mechanism described in Section 5.7. It is based on the current processing context $CX_{fp_x,it,1}$ that was the result of the iteration Pre-Processing function.

⟨*IterationStreamProcessingBuilder*⟩ `::= 'IterationStreamProcessingBuilder' '{'`
⟨*StreamProcessingBuilder*⟩ `'}'`

5.6.5. Post-Iteration Processing Function

After the stream processing is finished, the Focused Situation Processing Instance executes a scenario-specific post-processing function $Post_{\tau}^{FP}$ (Definition 4.19) in order to prepare the processing context resulting from the stream processing $CX_{fp_x,it,2}$ for the evaluation of the result publication and termination rules as well as for finding the new Locked Area and Focus Area for the next iteration, which takes place based on the resulting context $CX_{fp_x,it,3}$. The Post-Processing function $Post_{\tau}^{FP}$ is specified in the same way as the Pre-Processing function as MVEL. Within the processing template it needs to be specified as follows:

⟨*PostIterationProcessingFunction*⟩ `::= 'postIterationProcessing'` ⟨*MVEL*⟩ `';'`

5.6.6. Interim Result Event Generation Function

In order to publish interim processing results, a scenario template needs to define an interim result publication function $InterimResultEventGen_{\tau}^{Fp}$ (Definition 4.20). Within

the template, this function is defined by zero or more *interim result publication rules*. The rules are evaluated against the current Focused Situation Processing Context $CX_{fp_x,it,3}$ to determine if a result has to be published. If a rule is positively evaluated, a result event is generated based on a set of values from the processing context and published. Within the template, the publish rules can be specified as follows:

⟨*InterimResultEventGenerationFunction*⟩ `::=` ⟨*publishRuleDef*⟩`*`

⟨*publishRuleDef*⟩ `::= 'publish' 'result'` ⟨*vars*⟩ `'when'` ⟨*MVEL*⟩ `';'`

⟨*vars*⟩ `::=` ⟨*VAR*⟩ `( ',' ` ⟨*vars*⟩ `)?`

Where the keys specify which elements of the current processing context should be embedded in the event. The condition is evaluated against the same processing context and triggers the publication of the defined event. If no rule is specified, no interim results will be published.

5.6.7. Focused Situation Processing Termination

In order to determine if a Focused Situation Processing Instance needs to be terminated, the processing model defines a Focused Situation Processing Termination Condition C_τ^{FpTerm} (Definition 4.22).

In the template, this function is defined by one or more termination rules. Furthermore, the rules define the Focused Situation Processing Result Query Function $Q_\tau^{FpResult}$ (Definition 4.23) by specifying if the processing resulted in a $FalseSituation$ or a valid situation and in the later case provide appropriate processing results in the same way as defined for the interim result publication (Subsection 5.6.6). Further the termination rule needs to specify if the situations Locked Area and Focus Areas are to be kept after the processing has finished in order to mark the situation (Definition 4.23).

Within the processing template, the rules can be specified as follows:

⟨*FocusedSituationProcessingTerminationConditionAndTerminationResult*⟩ `::=` ⟨*TerminationRule*⟩`+`

⟨*TerminationRule*⟩ `::= 'terminate' 'if'` ⟨*MVEL*⟩
`'with' 'result' ( 'FalseSituation' |` ⟨*vars*⟩ `)`
`'keep' 'area' 'registration' 'if'` ⟨*MVEL*⟩

5.6.8. Iteration Locked Area, Focus Area and Time Frame Query Function

If a Focused Situation Processing Instance is not terminated, the next processing iteration needs to be prepared by determining the next iterations Locked Area and Focus Area as well as its Time Frame. For this purpose the processing model defines the Iteration Locked Area, Focus Area and Time Frame Query Function Q_τ^{LaFaTf} (Definition 4.21). In

the scenario processing template the function is defined by three separate statements which are discussed in the following two subsections:

⟨*IterationLockedAreaFocusAreaTimeFrameQueryFunction*⟩ ::= ⟨*NextIterationLockedArea*⟩
⟨*NextIterationFocusArea*⟩ ⟨*NextTimeFrame*⟩

5.6.8.1. Iteration Locked Area and Focus Area Determination

To prepare for the next iteration, a new Locked Area and Focus Area needs to be derived from the processing context. The determination conditions are specified in the same way as for the initial Locked Area and Focus Area as a SPARQL query fragment (Subsection 5.2.2) that is expanded to a full query based on the referenced contents from the processing context $CX_{fp_x,it,3}$. Aside from the specification based on a SPARQL query, the areas can also be set to the contents of a variable available from the processing context $CX_{fp_x,it,3}$. If a variable is provided it must contain one or more references to nodes from the knowledge base so that they can be used for the next iteration's area registration.

Within the processing template, the Locked and Focus Area determination mechanism needs to be specified as follows:

⟨*NextIterationLockedArea*⟩ ::= **'nextFocusArea'** (⟨*VAR*⟩ | ⟨*SPARQL*⟩) **';'**

⟨*NextIterationFocusArea*⟩ ::= **'nextLockedArea'** (⟨*VAR*⟩ | ⟨*SPARQL*⟩) **';'**

5.6.8.2. Timing Specification

The timing of the Focused Situation Processing (Subsection 4.6.2.8) is specified in the same way as the initial time frame during the Focused Situation Processing Initialization (Subsection 5.5.2.2) but with the keyword *nextIterationTimeFrame*:

⟨*NextTimeFrame*⟩ ::= **'nextIterationTimeFrame' 'startsAt'** (⟨*VAR*⟩ | ⟨*MVEL*⟩)
'withDurationOf' (⟨*TIME_DURATION*⟩ | ⟨*MVEL*⟩) **';'**

The same restrictions as defined in Subsection 5.5.2.2 regarding the start time and duration apply here with the difference that the set of available variables is the current Focused Situation Processing Context $CX_{fp_x,it,3}$ (Subsection 5.6.1).

5.6.9. Focused Situation Processing Collision-Handling Function

As defined in Subsection 4.6.4.1, two Focused Situation Processing Instances may need to be merged into one if they collide. In this case one of the two Focused Situation Processing Instances is chosen to be terminated (fp_a) while the other may continue (fp_b). In order to allow the continuing instance to incorporate interim results from the terminating instance, a scenario-specific function $ColHandler_\tau$ is defined by the processing model (Definition 4.24). In the template the merge function needs to be specified in MVEL:

Variable Name	Access	Contents	Defined in Model
CONTEXT_A	Read Only	Allows read only access to the last Focused Situation Processing Context of the Focused Situation Processing Instance fp_a.	The two sets $CX_{fp_a,itLast_{fp_a},3}$ and $CX_{fp_b,itLast_{fp_b},3}$ from Definition 4.24 with $itLast_{fp_a}$ and $itLast_{fp_b}$ as the last iteration of the corresponding processing task before the collision occurred.
CONTEXT_B	Read / Write	Allows read write access to the last Focused Situation Processing Context of the Focused Situation Processing Instance fp_b which will be used by this processing instance to initialize its next processing iteration.	

Table 5.6.2.: Variables available in the scope of the Focused Situation Processing Merge Function.

⟨*FocusedSituationProcessingCollisionHandlingFunction*⟩ `::= 'mergeFunction'` ⟨*MVEL*⟩

The merge function allows access to the processing contexts of both processing instances and thereby allows to copy values from the terminating instance to the continuing.

For the execution of the merge function, the instance fp_a is always the instance that is about to be terminated while the instance fp_b is always the instance that will continue its processing after the merge. In order to access the processing context of each of these two processing instances, the two objects $CONTEXT\_A$ and $CONTEXT\_B$ are available as defined in Table 5.6.2. The two objects are available during the MVEL function execution as java.util.Map objects and thus support the Map API to retrieve and add elements. In contrast to the typical way to assign properties to the processing context defined in Subsection 5.2.3, within the merge function, all access to the processing context needs to be made via these two objects in order to explicitly state which context should be used.

As the purpose of the merging function is to combine the information from processing instance fp_a with processing instance fp_b, no access to the knowledge base is possible.

5.7. Stream Processing Builder Function Definition

This section describes the language elements for the stream processing builder function definition that is employed for the preparation of the Phase 1 Situation Indication Processing as well as during the Phase 3 Focused Situation Processing.

Within the processing template, the builder functions are specified as procedural descriptions which contain templates for the event stream processing. The definition is executed with a given set of inputs and generates the *actual* stream processing functions by defining the actual stream processing topology for the event streams that are to be processed. Based

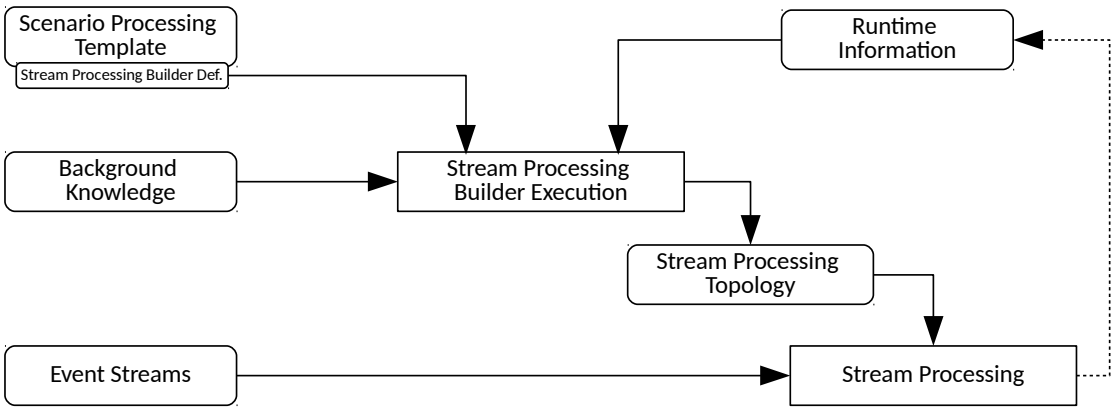


Figure 5.7.1.: Schematic view of the Stream Processing Builder execution.

on the generated topology, the stream processing is then instantiated by the processing system (Figure 5.7.1).

To specify the Stream Processing Rule templates, the Scenario Processing Template language embeds the Drools rule language. Within the Scenario Processing Template, the Drools rules are defined with certain placeholders which are later filled when the builder function is executed. This placeholder replacement then results in the definition of the set of actual stream processing rules which form the stream processing topology which in turn can then be deployed to the rule engine.

The procedural definition of the builder functions has to be given within either the $IndicationStreamProcessingBuilder$ (Section 5.4) or the $IterationStreamProcessingBuilder$ (Subsection 5.6.4) blocks where their content is defined as follows:

```
⟨StreamProcessingBuilder⟩ ::= ⟨ProcOperation⟩+

⟨ProcOperation⟩ ::= ⟨BackgroundKnowledgeQuery⟩
      | ⟨ForEach⟩
      | ⟨ForEachGroup⟩
      | ⟨PublishStatement⟩
      | ⟨SetOperation⟩
      | ⟨Conditional⟩
```

Within this block, several language constructs are available in order to specify the builder function by for example iterating over sets of nodes, grouping them or to retrieve additional background knowledge. Based on this general functionality, the SPTL allows the specification of stream processing statements which define the actual stream processing topology.

The following subsections describe the different language constructs and their interpretation, each with short examples. The section concludes with a complete stream processing

builder execution example for the possible Situation Indication Processing for the cloud tracking scenario (Subsection 5.7.7).

5.7.1. Stream Processing Builder Context

The stream processing builder function has its own context during the execution with read write access to its contents. Before the builder function execution, the context is populated with the variables available to the current block in the Scenario Processing Template:

Phase 0 (Possible Situation Indication Processing):
: The variable *$$indicationNodes* is copied to the Stream Processing Builder Context.

Phase 1 (Focused Situation Processing):
: The current Focused Situation Processing Context is copied to the Stream Processing Builder Context.

After the function's execution, the context is dropped, so changes to the context are only scoped to the current builder function's execution.

5.7.2. Background Knowledge Queries

In order to access the background knowledge from within the stream processing builder function, the template language supports the specification of SPARQL based queries where the results are assigned to variables in the scope of the builder function execution. The queries can be specified as follows:

⟨*BackgroundKnowledgeQuery*⟩ ::= ⟨*VAR*⟩ ⟨*SPARQL*⟩ ';'

The SPARQL query follows the definitions in Subsection 5.2.2. Embedded variables are resolved against the Stream Processing Builder Context.

5.7.3. Control Structures: Loops

The language supports two kinds of loops to handle variables, *foreach* and *foreach group*. They are similar to the „for each" construct known in many programming languages. However, the *foreach group* loop extends this basic concept with the capabilities to form groups of elements based on a grouping relation.

Foreach
: The foreach loop takes a variable and executes its loop body for each element in the set. The current element is available within the scope of the loop via the specified variable:

```
foreach $$SetVariable as $$InternalVariable{                                    1
  // Body called for each value of $$SetVariable assigned to                    2
      $$InternalVariable
}                                                                               3
```

The foreach loop is defined as follows:

⟨*ForEach*⟩ ::= 'foreach' ⟨*VAR*⟩ 'as' ⟨*VAR*⟩ '{' ⟨*ProcOperation*⟩+ '}'

Foreach Group

In a similar way as the foreach statement, the foreach group statement can be used to iterate over the contents of a variable. In contrast to the normal foreach, the loop body is executed for subsets of the elements from the provided variable. The groups are generated based on the grouping relation provided after the `group by` statement:

```
foreach $$SetVariable as $$InnernalVariable group by "grouping relation" {
  // Body called for each group that was formed from the $$SetVariable
      contents with the corresponding grouping relation. The set that
      corresponds to the current group is assigned to $$InnernalVariable
}
```

The foreach group is defined as follows:

⟨*ForEachGroup*⟩ ::= 'foreach' ⟨*VAR*⟩ 'as' ⟨*VAR*⟩ 'group by' ⟨*STRING*⟩
'{' ⟨*ProcOperation*⟩+ '}'

The given grouping relation is used to form a SPARQL query to retrieve a single grouping value for each element from the given input variable. The groups are then built based on the grouping values retrieved by the query, were all elements with the same grouping value are assigned to the same group as shown in the following example:

Example 15

Assuming the following triples are available from the background knowledge:

$$\begin{array}{lllll} L1 & - & telco:SubLink & - & Agg1 \\ L2 & - & telco:SubLink & - & Agg1 \\ L3 & - & telco:SubLink & - & Agg2 \\ L4 & - & telco:SubLink & - & Agg2 \end{array}$$

And assuming that the variable *$$In* has the following contents: L1,L2,L3,L4. Then based on this, the following *foreach group* statement is evaluated as follows:

```
foreach $$In as $$Group group by "telco:SubLink" { ... }
```

In order to form the groups, for each value of $$$In$, the grouping criteria is queried from the background knowledge by creating and executing the following query:

```
select distinct ?VALUE where {{ ?VALUE telco:subLink $${currentElement} }
    UNION { $${currentElement} telco:subLink ?VALUE}}
```

The query is executed for each value of $$$In$ with the following results:

$$\begin{array}{lcll} L1 & \Longrightarrow & Agg1 & \rightarrow Group\,1 \\ L2 & \Longrightarrow & Agg1 & \rightarrow Group\,1 \\ L3 & \Longrightarrow & Agg2 & \rightarrow Group\,2 \\ L4 & \Longrightarrow & Agg2 & \rightarrow Group\,2 \end{array}$$

The results are used to form groups based on equal results. In this case two groups are generated: $Group\,1 = \{L1, L2\}$ $Group\,2 = \{L3, L4\}$. The body of the loop is then executed for each of the groups which will be assigned to the specified variable $\$\$Group$.

5.7.4. Event Stream Processing Statements

In order to specify the actual event stream processing functions from Phase 1 ($sp_{\tau}^{Indication}$, Subsection 4.3.2) and Phase 3 ($sp^{FpIteration}$, Subsection 4.6.2.5), the SPTL provides a *rule* statement which allows for the specification of Drools stream processing rule templates.

Each rule statement specifies one stream processing rule as a template with placeholders for variables from the processing context as well as placeholders for the required inbound event streams that should be assigned to the rule when it is instantiated. Furthermore, the result of the stream processing rule is specified by the rule statement.

In order to accommodate for the different requirements of the two event stream processing functions, the rule statement requires the explicit specification of the type of result produced by a given rule:

- For realizing $sp_{\tau}^{Indication}$ (Phase 1) the rule statement allows the publication of *interim result streams* and the *publication of possible situation indication events.*
- For realizing $sp^{FpIteration}$ (Phase 3) the rule statement allows the publication of *interim result streams* but also the *modification of the Focused Situation Processing Context.*

The publish statement for both rule types is defined as follows:

```
⟨StreamProcessingRule⟩ ::= 'rule' ⟨DROOLS⟩ 'publishes' (
    'indications'
    | ( 'stream' ( ⟨VAR⟩ '.'  ⟨ID⟩ | ⟨ID⟩ ) | 'no' 'stream' ) ('manipulates' 'context')?
    )  ';'
```

For example a stream processing rule for Phase 1, that produces possible situation indication events is specified as:

```
// Note: Only possible for Phase 1 Processing                                    1
rule [DROOLS_TEMPLATE]...[/DROOLS_TEMPLATE] publishes indications;               2
```

A stream processing rule for Phase 1 or 3, that produces an interim result stream named „myInterimResultStream“ related to *$$var* but does not manipulate the current processing context is specified as:

```
// Note: Possible for Phase 1 and Phase 3 Processing
rule [DROOLS_TEMPLATE]...[/DROOLS_TEMPLATE] publishes stream $$var.
    myInterimResultStream;
```

When a rule also stores processing results in the current processing context, the rule statement needs to be suffixed by „manipulates context“ (which is only allowed in the Phase 3 processing):

```
// Note: Only possible for Phase 3 Processing
rule [DROOLS_TEMPLATE]...[/DROOLS_TEMPLATE] publishes stream $$var.
    myInterimResultStream manipulates context;
```

Furthermore, a rule for the Phase 3 processing may completely omit the generation of an event stream and *only* store its processing results in the processing context:

```
// Note: Only possible for Phase 3 Processing
rule [DROOLS_TEMPLATE]...[/DROOLS_TEMPLATE] publishes no stream manipulates
    context;
```

The different types on how to specify result streams are discussed in the following subsections.

5.7.4.1. Situation Indication Stream Processing Rule

If a stream processing rule is defined with „*publishes indications*“ it may publish Possible Situation Indication Events for the current Scenario. In order to publish such indication events, the Drools rule has to use the provided *publishIndication*(...) function which takes as first argument the set of nodes for which the possible situation is indicated and as optional second argument the time for which the indication should be created. If the time is omitted, the current time of the rule engine is used. The two versions of the function are declared as follows:

```
/**
Publishes a Possible Situation Indication Event with the given set of
    indicatedNodes and the indicatedTime set to the current time of the rule
    engine
*/
void publishIndication(Set indicatedNodes);

/**
Publishes a Possible Situation Indication Event with the given set of
    indicatedNodes and the indicatedTime set to the specified time.
*/
void publishIndication(Set indicatedNodes, long indicatedTime);
```

For example:

```
rule [DROOLS_TEMPLATE]
when
```

```
     Number( $average : doubleValue ) from accumulate(                                    3
        SingleMeasurement( $val:doubleValue ) over window:length( 5 ) from entry-        4
            point "$${{childlink.traffic_in_average}}", average( $val )
    )                                                                                    5
    eval( $average > 10  )                                                               6
then                                                                                     7
    publishIndication( $${{childlink}} );                                                8
end                                                                                      9
[/DROOLS_TEMPLATE] publishes indications;                                                10
```

5.7.4.2. Interim Result Event Stream Generating Rule

If a stream processing rule is defined with „publishes stream“ it may publish own interim result events which can be used by other rules of the same FSP Instance as input. To publish such an event, the Drools rule has to use the provided *publish(...)* function which takes the event object as parameter and publishes it to the interim result stream defined for this rule. The function is declared as follows:

```
void publish(Object event);                                                              1
```

The resulting stream is only available within the scope of the current stream processing topology of the current FSP Instance in the scope of the current scenario. In contrast, the measurement data streams provided to the processing system like the measurements from the solar panels are globally available.

In order to reference the newly created streams within the template, they are assigned as either a sub-stream to a given node or set of nodes or get a unique name within the scope of the stream processing topology . If multiple stream processing rules in the same stream processing topology publish to the same interim result event stream, the results are merged into one event stream.

5.7.4.3. Context Access and Context Manipulating Stream Processing Rule

If a stream processing rule for the Phase 3 processing is defined with „manipulates context“, it may change variables in the current processing context by calling the following function on the provided context:

```
void   CONTEXT.set("$$variableName",Object value);                                       1
```

In order to access variables from the processing context, the following function may be used:

```
Object CONTEXT.get("$$variableName");                                                    1
```

Additional access methods may be provided by the implementation.

5.7.4.4. Variable Placeholders

Stream processing statements can contain embedded references to variables from the processing context of the stream processing builder where the embedded variable is specified

as *$${variableName}* like for example:

```
rule [DROOLS_TEMPLATE] ... $${{someVariable}} == 0 ... [/DROOLS_TEMPLATE]
    publishes stream someInternalStream;
```

Before the execution of the stream processing rule takes place, this placeholder is replaced by the current value of the specified variable.

5.7.4.5. Inbound Event Stream Assignment

The inbound event stream subscription is based on the embedded variable replacement of variables in a processing rule specification. As the actual event stream processing rule definition is based on the Drools language, the inbound streams need to be specified according to Drools based on so called „entry-point“ definitions:

```
rule [DROOLS_TEMPLATE] ... $ev : MeasurementEvent from entry-point "$${{childlink
    }}" ... [/DROOLS_TEMPLATE] ...;
```

Two general types of event streams can be assigned to a stream processing rule which are discussed in the following paragraphs:

1. *External event streams* provided by the monitored system for example the energy production measurements of solar panels.
2. *Internal interim result event streams* within the scope of a Stream Processing Topology like for example some initial aggregation of measurement data.

External Event Stream Assignment

The selection of the actual measurement event streams is based on suffixes appended to variables containing node references in Drools stream processing templates. For example the following statement requests the stream „PVPowerProduced“ for the node contained in the variable *$$someNode*.

```
rule [DROOLS_TEMPLATE] .... from entry-point "${{someNode?PVPowerProduced}}"....
    [/DROOLS_TEMPLATE]  ... ;
```

The prefixes are domain and implementation specific mappings to available event stream types from the background knowledge (nodes in the background knowledge that are subclasses of *NodeWithLinkedEventStream* as illustrated in Figure 4.2.1). For example for providing access to photo-voltaic power production measurements, the prototype uses the „PVPowerProduced“ prefix which is mapped to nodes of type *pvPowerProduced* in the background knowledge as shown in Figure 4.2.2. Similarly, for the telecommunications domain the prefix „TrafficIn“ is available which is mapped to nodes of type *trafficIn* which provide the appropriate event stream as shown in Figure 4.2.3.

The variable to which the prefix is applied to must at the time of the template interpretation only contain a single node and not a set of nodes as no automatic merge of multiple inbound event streams is supported.

Internal Interim Result Event Streams

Within the scope of the current Stream Processing Topology, interim result event streams generated by another event stream processing rule in the same topology can be assigned as input in a similar way as the external streams:

```
rule [DROOLS_TEMPLATE] .... from entry-point "${{someNode.myInterimResultStream}}"
    .... [/DROOLS_TEMPLATE]  ... ;
```

The streams can be referenced by the name assigned to them during the processing rule specification (see Subsection 5.7.4.2).

5.7.5. Set Operations

As the language is aimed at handling sets of nodes that should be set up for a monitoring or further analysis, the language supports basic set operations like building a union or intersection of two sets of nodes.
The following language constructs are supported:

- Union: '+'
- Intersection: '&&'
- Difference: '-'

Which are defined in the template language as:

⟨*SetOperation*⟩ ::= ⟨*VAR*⟩ '=' ⟨*VAR*⟩ ('+' | '&&' | '-') ⟨*VAR*⟩ ';'

5.7.6. Conditional Statement

The language supports the typical if/else conditional statement where the condition needs to be specified in MVEL and must return a boolean value:

⟨*Conditional*⟩ ::= 'if' ⟨*MVEL*⟩ '{' ⟨*ProcOperation*⟩* '}' ('else' '{' ⟨*ProcOperation*⟩* '}')?

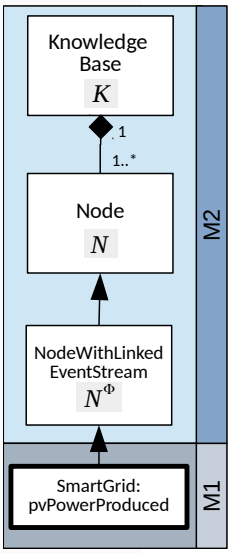
5.7.7. Stream Processing Builder Example

This section discusses an example on how the stream processing builder functions are defined and evaluated based on the cloud tracking scenarios possible situation indication. The complete Scenario Processing Template is given in Appendix B.1 and is discussed in Section 7.3.

Example 16

The Possible Situation Indication Processing for the cloud tracking scenario takes place for all solar panels in parallel. For this purpose the variable $$indicationNodes is assigned with a set of all available solar panels from the background knowledge (See Section 5.4). Based on this input, the following stream processing builder is evaluated to generate the Indication Stream Processing Topology for the scenario:

```
IndicationStreamProcessingBuilder{
      foreach $$indicationNodes as $$pv {
        rule [DROOLS_TEMPLATE]
            when
                Number( $delta : doubleValue )
            from accumulate(
              MeasurementEvent( $val:value )
              over window:length( 2 )
              from entry-point "$${{pv?PVPowerProduced}}",
              SuddenChangeDetector( $val )
          ) eval($delta > 50)
              then
                publishIndication( "$${{pv}}" );
              end
        [/DROOLS_TEMPLATE] publishes indications;
  }
}
```



Excerpt from Fig. 4.2.2.

During the builder execution, the foreach loop iterates over all nodes in *$$indicationNodes* and generates with the *rule* statement one stream processing rule for each solar panel that is to be monitored. Table 5.7.2 illustrates a generated rule with its inbound event stream assignment for the node *provider:panel1.* In the same way, processing rules will be generated for all other elements in the *$$indicationNodes* set.

Possible Situation Indication Rule for Node: *provider:panel1*

Input Data Stream:	*provider:panel1.production* as internal entry-point *„datastream_ 0“*
Output Data Stream:	possible situation indication stream for the current scenario
Processing Rule:	when Number($delta : doubleValue) from accumulate(MeasurementEvent($val:value) over window:length(2) from entry-point "datastream_0", SuddenChangeDetector($val)) eval($delta > 50) then publishIndication("*provider:panel1*"); end

Figure 5.7.2.: One processing rule generated by the stream processing builder for the node *provider:panel1* from the set *$$indicationNodes* with the assigned inbound and outbound event streams.

5.8. Summary

The chapter defines the Scenario Processing Template Language (SPTL) to allow the specification of scenario-specific functions and parameters for a processing system implementing the defined processing model (Chapter 4). The language embeds existing languages for certain purposes like the retrieval of background knowledge or the definition of stream processing rules which can ease the usage for users experienced with one or more of the embedded languages.

The processing templates defined in SPTL are combined with background knowledge during run-time by a processing system to configure itself for the scenario. The next chapter discusses the prototype of a processing system implementing the defined language and model.

6. Prototype

Contents

After the two previous chapters defined the processing model and the Scenario Processing Template Language, this chapter discusses the architecture of the prototypical implementation of the model and language. The prototype discussed here was used for the evaluation of the designed processing model and language discussed in the following Chapter 7.

The chapter discusses the prototype's architecture loosely based on the structure discussed in [GKRS15, Chapter 4] by first discussing the goals of the prototype, then continues with a component view of the architecture followed by the discussion of several run-time aspects. The chapter concludes with a brief discussion of the prototype's deployment and surrounding systems (the Data Simulation and Result Visualization).

6.1. Goal of the Prototype

The goal of the designed and implemented prototype is to provide a test environment that allows the verification of the processing model and specification language. As such the prototype focuses on these aspects and is not aimed at providing a feature-rich distributed scalable application directly usable for large amounts of streaming data and various scenarios. In the same way the following discussions of the prototype are focused on how the processing model is implemented by the processing system.

Even though not the focus of the prototype, its architecture outlines how a processing system implementing the focused situation processing model can be divided into components with distinct responsibilities and limited linkage with other components which would

also be suitable for the implementation of a distributed scalable version of the processing system.

As the prototype is intended as a test environment only, the current implementation does not verify if a given scenario processing template adheres to all rules specified in Section 5. Thus the prototype allows a template to specify processing functions, expressions or queries that break with the processing model. However, for the implemented scenario templates (Appendix B.1 and B.2) all given statements where carefully checked against the specified rules in order to allow for a correct evaluation of the capabilities of the model and language.

6.2. Component View

The prototype consists of 9 components, each with their distinct responsibility in implementing parts of the processing model or providing support functionalities. Figure 6.2.1 gives an overview over the components and their dependencies and interactions while the following sub-sections discuss each of these components.

6.2.1. Core Components

The following 4 components are considered core components as they *implement the defined processing model* while the other components provide supporting functions:

6.2.1.1. Area Registration Manager

The Area Registration Manager (ARM) implements a central registry for Area Registrations as defined by the processing model (Subsection 4.2.3.1). Furthermore, it decides if requests for new Area Registrations are granted based on the defined constraint from the processing model (Condition 4.1 on page 61) and acts as the central synchronization point between the Phase 3 Focused Situation Processing Instances as well as the Phase 2 Possible Situation Indication Event classification implemented by the Processing Manager.

In order to allow the creation, update or removal of Area Registrations, the ARM provides the *Area Registration Service* used by the Processing Manager and Focused Situation Processing Manager.

The ARM *internally* manages multiple Area Registration Manager Instances, one for each Scenario Processing Template in order to track its Area Registrations.

6.2.1.2. Processing Manager *(Implements Phases 0 & 2)*

The Processing Manager (PM) oversees the overall Focused Situation Processing process by implementing the initial setup of new processing templates (Phase 0) as well as the

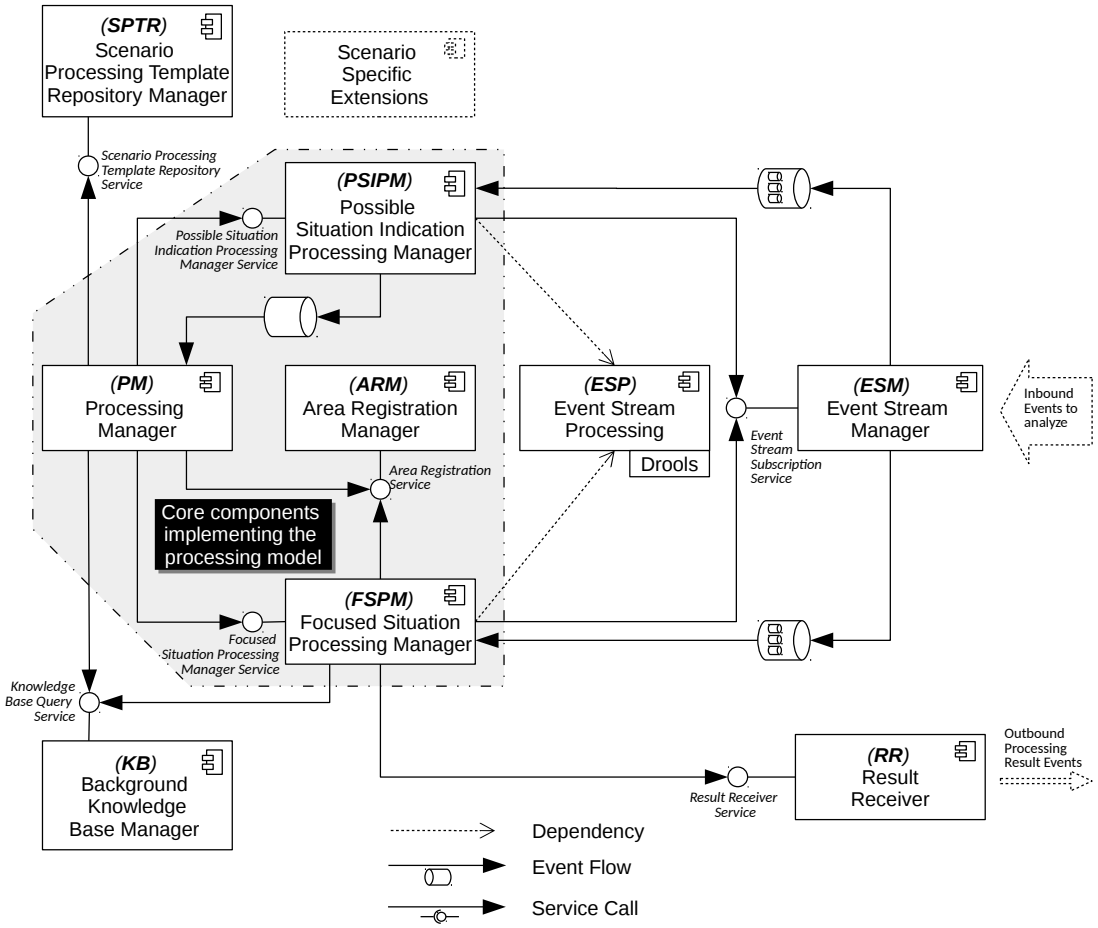


Figure 6.2.1.: Component view of the Prototype as discussed in Sections 6.2.1 and 6.2.2.

classification of raised Possible Situation Indication Events[1] (Phase 2). The PM offers no services to the other components.

For Phase 0 the PM utilizes the Scenario Processing Template Repository Service to retrieve the available Scenario Processing Templates. The retrieved templates are used to generate the Phase 1 Possible Situation Indication Stream Processing Topologies together with information obtained from the Background Knowledge Base Query Service. For the generated topologies the PM triggers the set up and start of the Phase 1 Possible Situation Indication Processing by using the Possible Situation Indication Processing Manager Service.

For Phase 2 the Processing Manager receives Possible Situation Indication Events from the Phase 1 - Possible Situation Indication Processing (implemented by the Possible Situation Indication Processing Manager component) via an event queue. For the received events, the PM implements the Phase 2 classification (Section 4.5). If the classification results in a new possible situation, the PM triggers the set up and start of a new Focused Situation Processing Instance by invoking the Focused Situation Processing Manager Service.

6.2.1.3. Possible Situation Indication Processing Manager *(Implements Phase 1)*

The Possible Situation Indication Processing Manager (PSIPM) implements the Possible Situation Indication stream processing (Phase 1) based on a given stream processing topology (Subsection 4.2.4), provided by the Processing Manager. The PSIPM utilizes the functionality of the Event Stream Processing component for the stream processing. Further it obtains the needed event streams from the Event Stream Subscription Service provided by the Event Stream Manager component. The processing results of the implemented Phase 1 processing, a stream of Possible Situation Indication Events, is published to an event queue which is consumed by the Processing Manager to classify the raised indications.

The PSIPM internally creates a Possible Situation Indication Processing Instance for each deployed stream processing topology (Figure 6.2.2). To trigger the creation of new instances, the component provides the Possible Situation Indication Processing Manager Service which is used by the Processing Manager.

6.2.1.4. Focused Situation Processing Manager *(Implements Phase 3)*

The Focused Situation Processing Manager (FSPM) implements Phase 3 of the processing model by providing the situation specific iterative processing as defined in Section 4.6. Similar to the PSIPM, this component uses the Event Stream Processing component to

[1] This includes triggering the creation of new Focused Situation Processing Instances if needed.

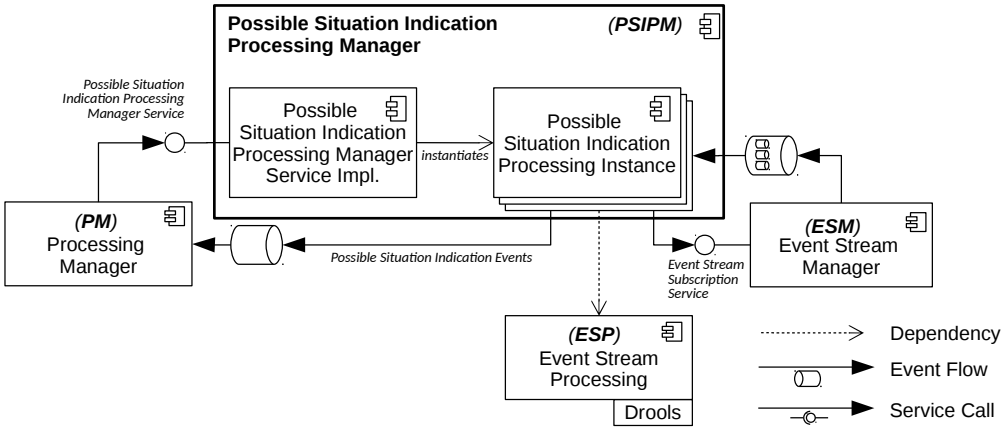


Figure 6.2.2.: Components of the Possible Situation Indication Processing Manager (PSIPM)

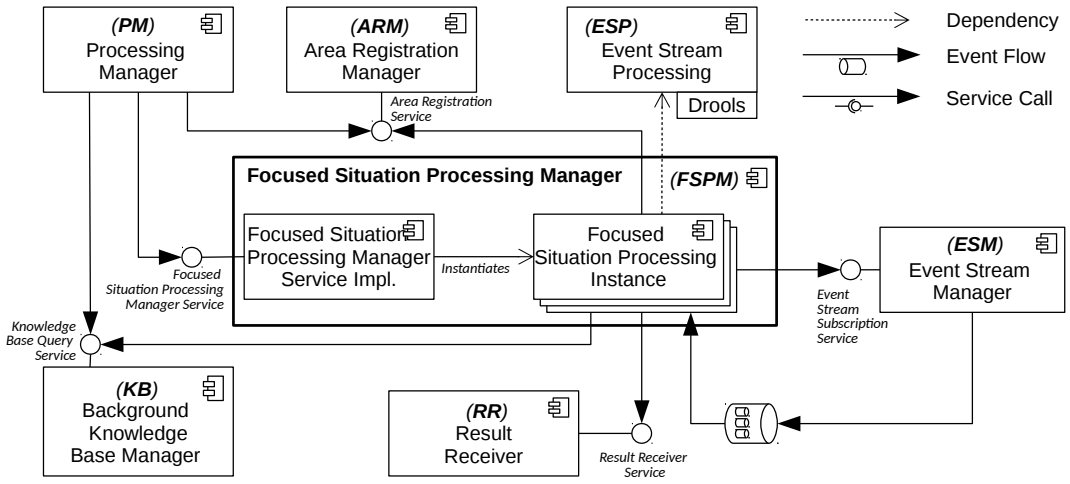


Figure 6.2.3.: Components of the Focused Situation Processing Manager (FSPM)

implement the event stream processing. Further it uses the Event Stream Subscription Service from the Event Stream Manager component to subscribe to the required event streams.
As the Phase 3 processing is allowed to retrieve additional background knowledge, the FSPM uses the Background Knowledge Base Query Service provided by the Background Knowledge Base Manager. The results from the Phase 3 processing are forwarded to the Result Receiver component via its Result Receiver Service.

Similar to the PSIPM, the FSPM creates several Focused Situation Processing Instances (possibly multiple per scenario template) in order to analyze multiple (potential) situations (Figure 6.2.3). To allow the creation of new Focused Situation Processing Instances, the FSPM provides a Focused Situation Processing Manager Service used by the PM to create new instances.

6.2.2. Supporting Components

Aside from the so far presented core components that implement the processing model, several supporting components exist, which are discussed in the following subsections.

6.2.2.1. Scenario Processing Template Repository Manager

The Scenario Processing Template Repository Manager (SPTRM) provides access to the Scenario Processing Templates so that they can be implemented by the processing system. The SPTRM loads the templates from files where they are specified using the defined SPTL (Chapter 5). To load the templates, the component implements a parser for the SPTL based on the ANTLR [Ant] parser generator.

To access the loaded templates, the SPTRM provides the Scenario Processing Template Repository Service which is used by the PM to retrieve the available templates when the overall processing system starts.

6.2.2.2. Event Stream Processing

The Event Stream Processing (ESP) component implements the common functionality needed to execute event stream processing topologies based on the JBoss Drools Fusion [dro] rule engine which is utilized by the PSIPM and FSPM components. The component can be seen as a supporting library and does not offer its own service.

6.2.2.3. Event Stream Manager

The Event Stream Manager (ESM) provides the capability to subscribe to event streams by providing an Event Stream Subscription Service. The service is used by the PSIPM and FSPM components. Once a subscription is made, the ESM creates an event queue for

the subscription and streams the requested events to the receiving component through the created event queue.

For this prototype the ESM reads the (measurement data) events from a number of CSV files available to the ESM (Figure 6.4.1) and sends them to the receiver as event streams.

6.2.2.4. Background Knowledge Base Manager

The Background Knowledge Base Manager (KB) provides access to the background knowledge available for the currently monitored system by providing a Background Knowledge Base Query Service. The service is used by the PM and FSPM components to retrieve background information based on the current Scenario Processing Template and the currently indicated or analyzed (Possible) Situation.

The KB is queried using SPARQL queries which are processed with the help of Eclipse RDF4J[2] [rdf]. The knowledge base contents are read from two Turtle [BBL11] files, one for specifying the domain specific schema (e.g. for the Smart Grid domain as illustrated in Figure 4.2.2 as Layer M1) and one for the actual information on the monitored system (As illustrated in Figure 4.2.2 as Layer M0).

6.2.2.5. Result Receiver

The Result Receiver (RR) component receives interim and final processing results from the FSPM component. To receive the events it provides a Result Receiver Service which is consumed by the FSPM. In the current prototypical implementation the RR writes the result events to a set of log files.

6.2.2.6. Scenario Specific Extensions

Aside from the components of the processing system, scenario-specific components may exist. Such components may provide domain specific methods like for example a geographical clustering function needed for the cloud tracking scenario (Subsection B.1.1). In general the prototype is completely independent of these scenario-specific extensions, they are only referenced by Scenario Processing Templates as part of their processing logic.

6.3. Run-Time View

In order to illustrate how the prototype implements the processing model, the following subsections discuss the realization of the 4 phases defined by the processing model. Afterwards Subsection 6.3.5 discusses the functionality of the Area Registration Manager component as it provides the central synchronization between multiple Focused Situation Processing Instances and can thus be seen as an essential part of the prototype and the

[2]Formerly known as Sesame

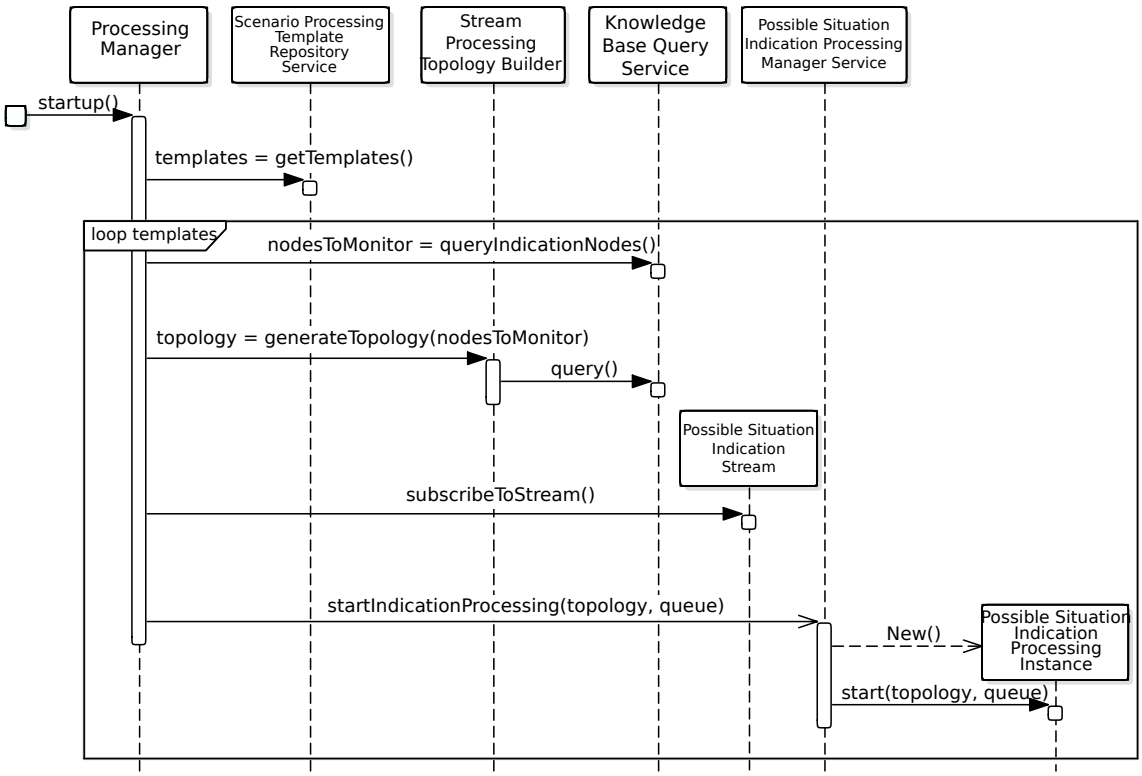


Figure 6.3.1.: Phase 0: Initialization of Scenario Processing Templates by the processing system.

realization of the processing model. Details on the supporting components is omitted as they do not directly realize the processing model defined by this work.

6.3.1. Phase 0: Possible Situation Indication Processing Initialization

The initial step to configure the processing system for a given Scenario Processing Template is the initialization of the Possible Situation Indication Processing to enable the processing system to detect Possible Situations. The initialization process is defined by the processing model as Phase 0 (Section 4.4) and in particular by Algorithm 1.

The processing system implements this algorithm in the Processing Manager (PM) which executes the Indication Nodes Query Function defined in the Scenario Processing Template with the help of the Knowledge Base Query Service. For the resulting set of nodes the PM executes the Indication Stream Processing Builder that was constructed from the Scenario Processing Template. The result of the builder call is a Stream Processing Topology. To execute this topology, the PM utilizes the Possible Situation Indication Processing Manager Service (PSIPM) and requests the instantiation of the given topology. In turn the PSIPM creates a new Possible Situation Indication Processing Instance which implements the actual event stream processing as discussed in the next subsection.

6.3.2. Phase 1: Possible Situation Indication Processing

The actual event stream processing to detect Possible Situation Indications is implemented in the Possible Situation Indication Processing Manager (PSIPM). The PSIPM encapsulates the processing for each Scenario Processing Template in a separate Possible Situation Indication Processing Instance. The Instance uses the event stream processing functionality of the Event Stream Processing component which deploys the provided Stream Processing Topology on the Drools Fusion rule engine and provides the necessary environment. The required event streams are obtained by requesting suitable subscriptions from the Event Stream Subscription Service.

Once the stream processing is started it can generate Possible Situation Indication Events which are published by the Possible Situation Indication Processing Instance to an event queue provided during the instantiation of this instance. The queued indication events are then handled by the Processing Manager.

6.3.3. Phase 2: Focused Situation Processing Initialization

Phase 2 of the processing model is implemented by the Processing Manager (PM). The PM receives the Possible Situation Indication Events from the PSIPM instances and classifies each of the events as specified in Section 4.5 (Algorithm 2). The resulting interactions with other components of the prototype are illustrated in Figure 6.3.2.

After the PM receives an indication event, it executes the Pre-Classification. If the Pre-Classification function classifies the event as a duplicate, the PM drops it without further consideration. If the event passed the Pre-Classification, the PM determines the initial time frame. Further it determines the potential Locked Area and potential Focus Area by executing the corresponding queries from the Scenario Processing Template against the Knowledge Base Query Service. Afterwards the PM needs to determine collisions with the Area Registrations of running Focused Situation Processing (FSP) Instances. To determine the possible collisions, the PM tries to acquire an Area Registration based on the determined initial Time Frame and the potential Locked and Focus Area.

Based on the results of the Area Registration request, the remaining classification is executed utilizing the classification rules specified in the Scenario Processing Template. Afterwards the resulting actions are executed: If the classification requires the assignment of the indication event to already running FSP Instances, the event is assigned to them by calling the FSPM together with the identifier of the FSP Instance which then forwards the event to this FSP Instance. In a similar way the PM can request the creation of a new FSP Instance by requesting it from the FSPM which in turn initializes and starts the new instance.

If the Area Registration request is successful but the classification of the indication event does not result in the start of a new FSP Instance, the PM releases the created Area Registration so that it can be deleted.

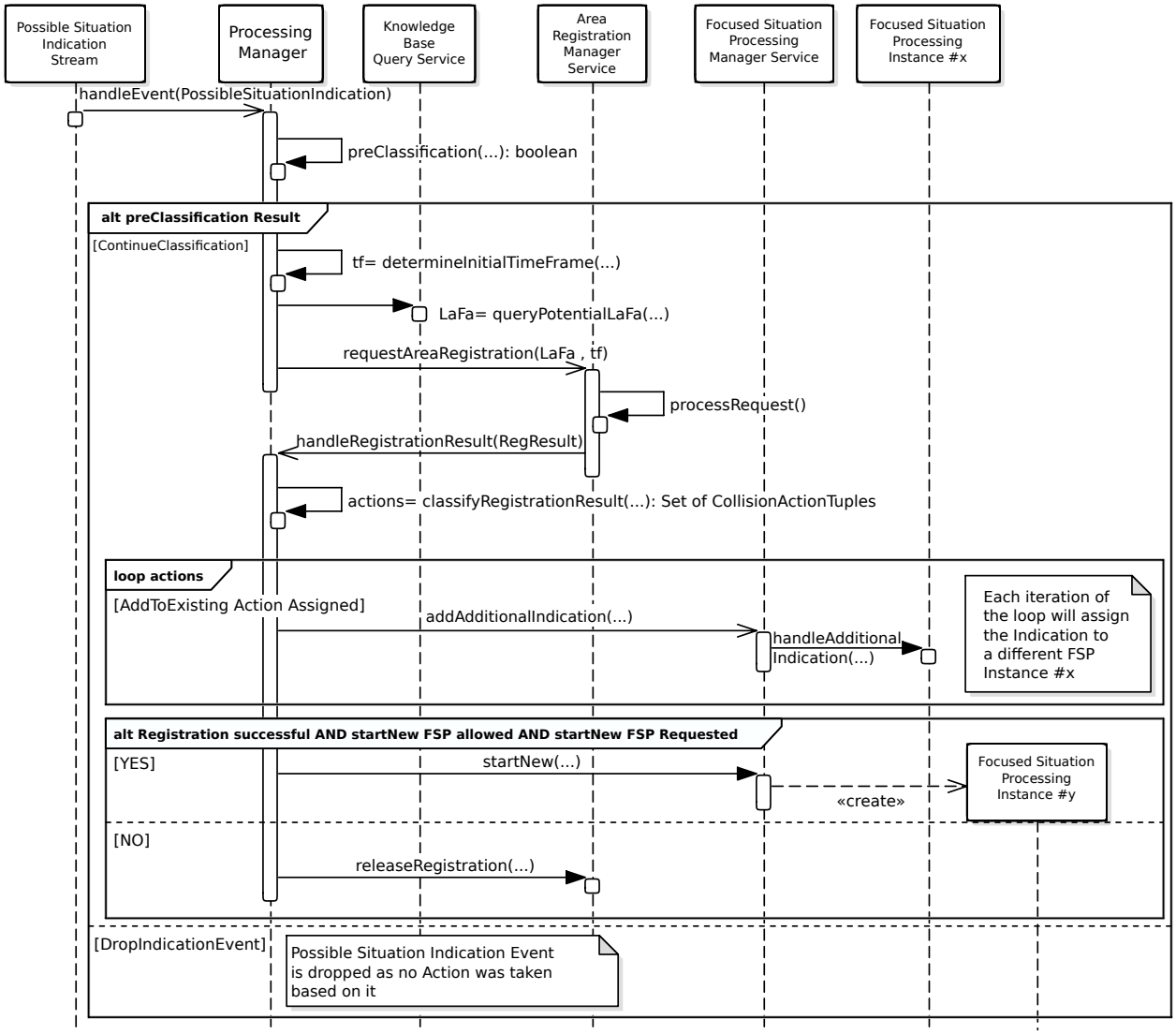


Figure 6.3.2.: Phase 2: Handling of raised Possible Situation Indication Events.

6.3.4. Phase 3: Focused Situation Processing

All processing regarding Phase 3 of the processing model is encapsulated in the Focused Situation Processing Manager component. The component manages each requested Focused Situation Processing (FSP) as a separate FSP Instance which encapsulates all information on the corresponding (possible) situation.

The component provides a Focused Situation Processing Management Service which is used by the Processing Manager to trigger the creation of new FSP Instances (Figure 6.3.2) and to assign additional indications.

A FSP Instance implements the Phase 3 Algorithm specified by the processing model (Algorithm 3). Figure 6.3.3 illustrates the processing flow of a FSP Instance and the resulting interactions with other components of the prototype.

The focused situation processing is defined by the processing model as an iterative process which is implemented as such by the prototype. As preparation for the iterative processing, the FSP Instance creates a FSP Context object and initializes it with the context initialization function defined in the Scenario Processing Template. Then, for each iteration, the FSP Instance first executes the scenario-specific Pre-Iteration function on the FSP Context followed by the execution of the FSP Iteration Stream Processing Builder[3] based on the resulting FSP Context in order to generate the stream processing topology for the current iteration. The FSP Instance then executes the stream processing which stores its results into the FSP Context. As defined by the algorithm, the results in the FSP Context are then provided to the scenario-specific Post-Processing function.

After the processing is finished, the FSP Instance evaluates all specified interim result publication rules. For any rule that is positively evaluated, the corresponding Interim Result Event is created from the contents of the FSP Context and forwarded to the Result Receiver. Afterwards all specified termination rules are evaluated against the FSP Context and the FSP Instance is terminated if requested by any of the rules. If the instance is terminated, the final result event is forwarded to the Result Receiver and if not requested otherwise the Area Registration Manager is informed that the Area Registration of the current iteration can be released.

If the processing is not terminated by any of the specified rules, the FSP Instance prepares for the next iteration by determining the next iterations time frame and by executing the next iteration Locked Area and Focus Area queries against the Knowledge Base Query Service. Based on the results the FSP Instance tries to acquire a new Area Registration for the next iteration.

While acquiring the initial Area Registration in Phase 2 is done asynchronously, here acquiring the Area Registration for the next Iteration is done synchronously as the corre-

[3]The FSP Iteration Stream Processing Builder was created by the Scenario Processing Template Repository Manager from the builder description contained in the Scenario Processing Template.

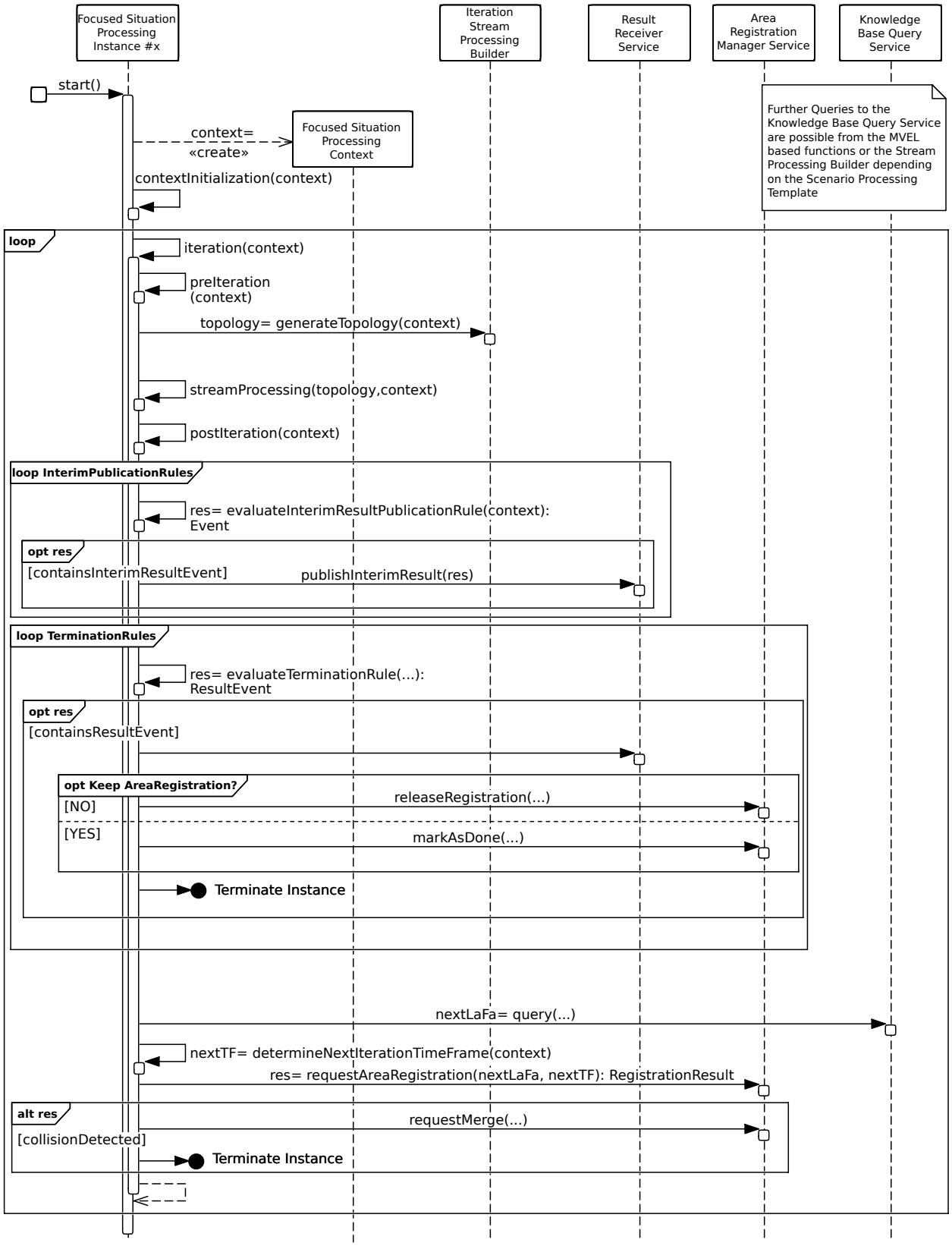


Figure 6.3.3.: Phase 3: Iterative Focused Situation Processing implemented by Focused Situation Processing Instances.

sponding processing instance can not continue with the processing until the registration was granted.

If the area registration request results in a collision with another FSP Instance, the colliding FSP Instance requests the merging of the two colliding instances from the Area Registration Manager which is responsible for the coordination between multiple FSP Instances as discussed in the next Subsection (6.3.5).

If the area registration request succeeds, the FSP Instance starts with the processing of the next iteration.

6.3.5. Area Registration Manager

The Area Registration Manager (ARM) acts as the central authority for granting Area Registrations. It thereby implements the Focus and Locked Area concept defined by the processing model (Subsection 4.2.3). Further it acts as central synchronization point between the Phase 2 and 3 processing (discussed in the following subsection) and is the coordinator for the FSP Instance merging (discussed in Subsection 6.3.5.2).

6.3.5.1. Synchronization between Phase 2 and Phase 3

As discussed in Subsection 4.5.10 the processing model requires the synchronization between the Phase 2 and Phase 3 processing as the classification from Phase 2 must *not* outrun any FSP Instance of Phase 3.

This synchronization requirement is realized by the ARM. The ARM grants and keeps track of the Area Registrations of all FSP Instances and thus has an overview over the current iteration time frames of each FSP Instance as they are part of the Area Registration. Further the Phase 2 processing also requests Area Registrations in preparation for its classification process.

To prevent the Phase 2 processing from outrunning the Phase 3 processing, the Area Registration Manager may delay the area registration requests originating from Phase 2, based on the requested time frame until all FSP Instances have at least reached the requested time frame. For this purpose the Phase 2 Area Registration request is implemented based on a callback as shown in Figure 6.3.2.

In order to implement this mechanism, the ARM needs to be informed if an FSP Instance has finished its processing for the time frame contained in its Area Registration. For the normal iterative processing flow of the FSP Instances, this notification is implicitly made when an FSP Instance requests a new Area Registration thus implicitly stating that it is finished with the processing of the previous one. However, as the ARM is not informed about life-cycle changes of the FSP Instances, FSP Instances explicitly need to mark Area Registrations as "done" if they terminate without releasing their last Area Registration[4].

[4]This behavior can be explicitly requested by the FSP Instance as a termination rule specifies if the current Area Registration should be kept or released.

6.3.5.2. Merge Processing Coordination

Aside from the synchronization between Phase 2 and 3, the ARM is also part of the coordination among FSP Instances as they request new Area Registrations.

If such an area registration request results in a collision between two or more FSP Instances, the processing model defines that the colliding FSP Instances have to be merged into one. To realize this merging, the ARM provides a synchronization between colliding instances as follows (Figure 6.3.4):

1. If an FSP Instance A detects a collision with another FSP Instance B, FSP Instance A requests a merge with Instance B from the ARM. The ARM then blocks the calling FSP Instance A till the merge has been completed.

2. in the meanwhile, FSP Instance B requests a new Area Registration from the ARM. As Instance A requested a merge, the area registration request is *not* immediately executed. Instead, the ARM executes the merge function defined in the Scenario Processing Template in order to allow the transfer of information from the to be terminated Instance A to Instance B.

3. Once the merge function was executed, the area registration request of Instance B is executed and the result returned to Instance B which will then continue with the processing. Furthermore, the blocking merge request of Instance A returns, thereby informing Instance A that the merge has been completed and Instance A is terminated afterwards.

6.4. Deployment

The current prototype is implemented as a single Java application that runs on a single machine. The prototype obtains its configuration as well as the event streams from a number of files (Subsection 6.4.1). The processing results are written into protocol files (Subsection 6.4.2). Aside from the prototype itself two additional components have been created (Figure 6.4.1) to allow testing the prototype:

Data Simulator The data simulator provides test event streams as well as relationship data to allow the evaluation of the processing system prototype as well as of the processing model.

Result Visualization The result visualization allows the visualization of the processing results generated by the prototype together with the simulated input data to ease the verification of the processing results and development of new Scenario Processing Templates.

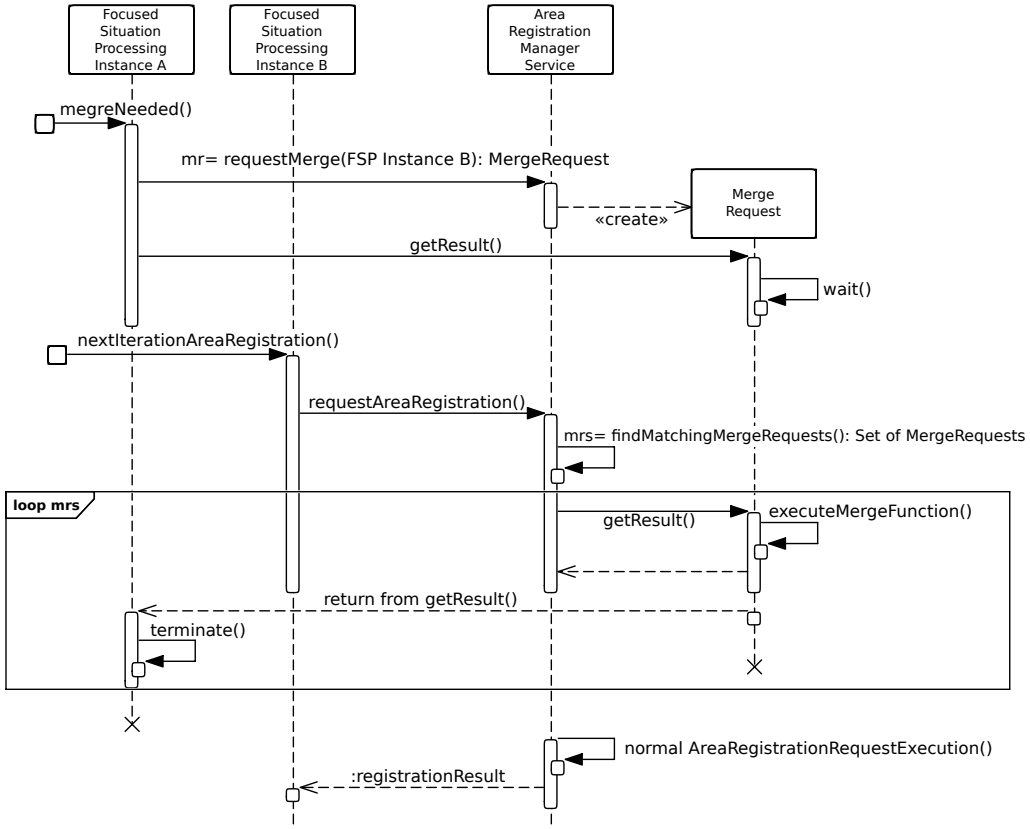


Figure 6.3.4.: Phase 3: Coordination of the merge between two FSP Instances by the Area Registration Manager.

6.4.1. Prototype Configuration and Input Data

The prototype depends on several data and configuration files:

Background Knowledge The prototype imports the background knowledge base contents from two Turtle files, one providing the domain specific schema (e.g. the Smart Grid Schema) and the other providing the actual background knowledge regarding the monitored system based on the domain specific schema (e.g. the available solar panels and their geographical positions).

Event Stream Contents The prototype loads the measurement data streams that are to be processed from a number of CSV files. The mapping of the CSV file contents to the event streams defined in the background knowledge is defined in a configuration file `eventStreamSources.cfg`. The configuration file further specifies which column of each of the specified CSV files contains the measurement time and which column the measurement value.

Scenario Processing Templates The prototype reads the Scenario Processing Templates from a number of SPTL files.

6.4.2. Prototype Processing Output

The prototype generates a number of protocol files which are located in `<runtime>/log/run_<timestamp>/`. The actual processing results are written into a separate protocol file for each started FSP Instance. Further several more protocols are written in order to document made area registration requests, the classification of raised Possible Situation Indications as well as the iterations done by the FSP Instances.

The generated protocols are used by the Result Visualization tool to visualize the processing results.

6.5. Conclusion

The prototype implements the designed processing model in a single Java application, which together with the Data Simulator and the Result Visualization allows testing the processing model. Further the prototype implements a parser and interpreter for the designed SPTL thus allowing to test the definition of scenario processing templates defined in SPTL.

The designed architecture of the prototype further demonstrates how the different phases of the processing model can be compartmentalized into separate components with a limited set of interactions and a central synchronization component (Area Registration Manager). Due to this decoupling of the components, a possibility for distributing the processing system for scalability is outlined.

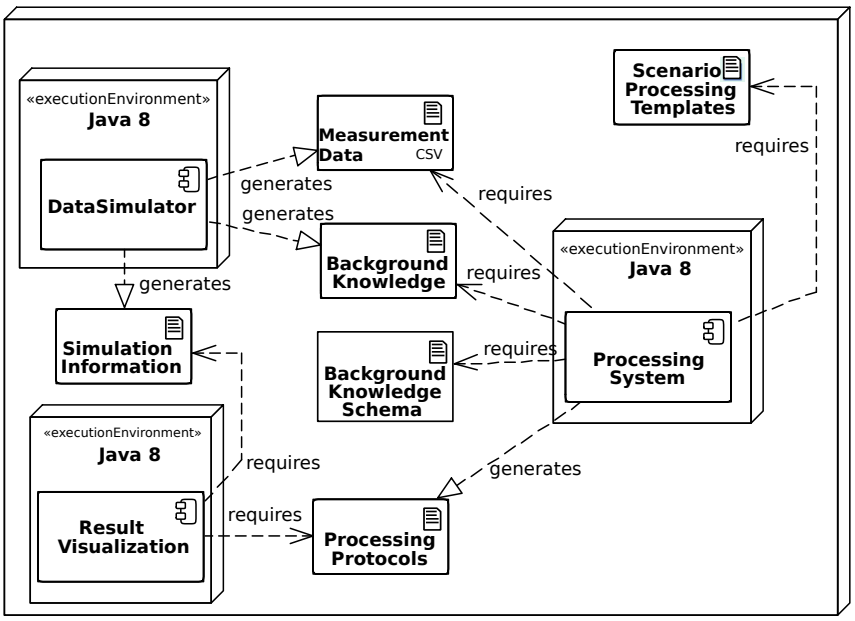


Figure 6.4.1.: Deployment view of the prototype with the surrounding systems, the Data Simulation and the Result Visualization and the produced and consumed data and configuration files.

The prototype together with the Data Simulator and the Result Visualization as shown in Figure 6.4.1 are the basis for the evaluation of the processing model and specification language discussed in the next chapter and Appendix C.

7. Evaluation

Contents

The goal of this work is to allow the development of situation-aware adaptive processing systems without the need to implement a specific processing system for each scenario (Subsection 1.4). The approach to achieve this goal is to define a situation-aware adaptive processing model (Chapter 4) together with a specification language (Chapter 5) which allows the specification of situation-aware adaptive processing tasks based on the defined model so that they can be executed by a generalized processing system.

The evaluation discussed in this chapter shows that the specification language can be used to define situation-aware adaptive processing tasks based on two example scenarios from two different application domains. Based on these scenarios several tests were conducted to demonstrate that the language is capable of defining all elements needed to parameterize a processing system for a given scenario. Further, the tests demonstrate the use of the designed situation-aware adaptive processing model as it is the foundation for the realization of the two scenarios and the conducted tests.

The chapter starts with a discussion on how the resulting processing model and language map to the defined research questions (Section 7.1). It then continues with a discussion of the aspects of the processing model and language that need to be tested in order to verify the solution's suitability (Section 7.2). The remainder of this chapter then discusses

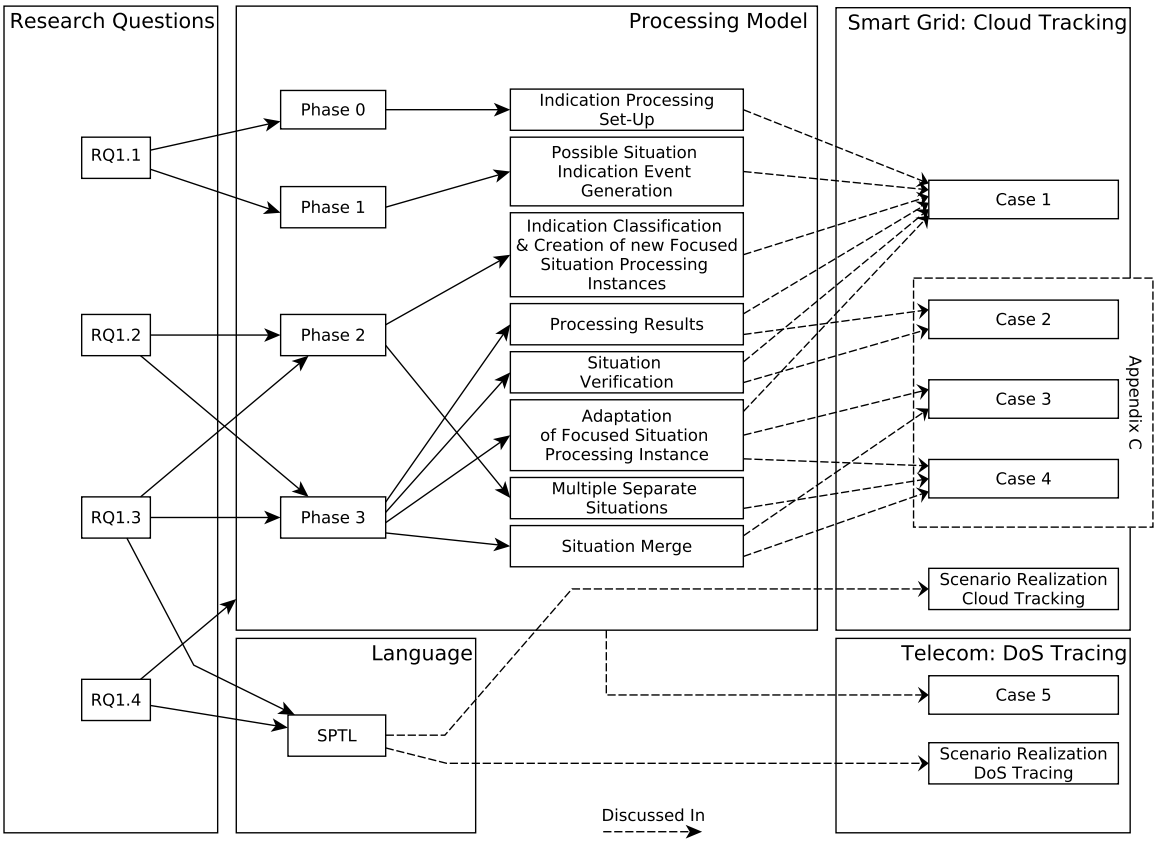


Figure 7.1.1.: Overview of the implemented test cases and the functionality demonstrated by each test case and the mapping of the research questions to the processing model and language.

the realization of the two application scenarios (Sections 7.3 and 7.5) and based on it, the two primary test cases 1 and 5 in detail as they demonstrate the central aspects of the processing model (Sections 7.4 and 7.6). The additional test cases 2 to 4 are discussed in detail in Appendix C. A general discussion on the limitations of the processing model is given at the end of this chapter in Section 7.8.

7.1. Mapping of the Evaluation to the Defined Research Questions

The defined research questions (Section 1.3.1) are answered by the designed processing model and language as discussed in the following paragraphs (Figure 7.1.1).

The overall research question that motivates this work is defined in Subsection 1.3.1 as follows:

RQ1: How to allow situation-aware adaptive processing without the need to implement a specialized solution for each scenario that requires a situation-aware adaptive processing?

In order to find an answer to this question, this work approached the problem by designing a situation-aware adaptive processing model and a language in order to allow the specification of situation-aware adaptive processing tasks which can be executed by a generic processing system. Tests conducted based on a created prototype demonstrate that the designed processing model and language provide the expected situation-aware adaptive processing mechanism without the need to create a specific implementation for each scenario (Scenario Realizations 1 & 2 and Test Cases 1 to 5).

The overall research question was subdivided into four sub-questions which are discussed and mapped to the conducted test cases in the following paragraphs:

RQ1.1: *How can a generalized solution provide a processing system suitable to handle large amounts of streaming data for a situation-aware adaptive processing?*

The designed processing model defines a separation of the rapid detection or indication of possible situations from the more time consuming possible situation verification and its further analysis in order to cope with large amounts of streaming data. This separation allows the usage of simple operations for detecting potential situations in a possibly large set of inbound event streams while the later more resource intensive analysis is only executed for potential situations and thus only for subsets of the overall streaming data. This two step mechanism is demonstrated in particular by the Test Cases 1 & 3.

Furthermore, the Possible Situation Indication Processing itself provides two mechanisms to better cope with large sets of inbound events:

a) All background information that is needed for the possible situation indication is retrieved before the stream processing starts (in Phase 0) as retrieving the background information is considered an expensive operation.

b) The processing model allows the parallel execution of the possible situation indication stream processing if supported by the scenario (Phase 1). The only required synchronization between the parallel processes is performed based on their processing result, the detected possible situation indications in Phase 2 which thus operates only on a reduced set of events.

The Phase 2 processing (Focused Situation Processing Initialization) further supports reducing the event load with its pre-classification mechanism (Subsection 4.5.1).

The following resource intensive Focused Situation Processing (FSP) in Phase 3 is also only executed for potential situations if the Phase 2 deemed it necessary. Furthermore, the Phase 3 FSP is designed with only one synchronization point per processing iteration with other FSP Instances[1] thereby also allowing for a mostly parallel execution of multiple Focused Situation Processing Instances.

[1] During the acquisition of the next iterations Area Registration.

In order to coordinate between multiple (possible) situations, the processing model defines the concept of Area Registrations (Subsection 4.2.3.1) which consist of two sets of nodes (the Locked Area and the Focus Area) together with a time frame during which a registration is valid. The set of created Area Registrations serves as the single synchronization point among all active FSP Instances (Phase 3) and the classification of new potential situations (Phase 2). Thus, it effectively limits the parallelization of any processing system implementing the designed model. The implementation of this synchronization mechanism can however be based on systems like for example Apache Redis [red] or Apache Zookeeper [Apai] which allow scalability for such data structures.

RQ1.2: *How can a generalized solution define the adaptation steps in a flexible and domain independent way?*

RQ1.3: *How can a generalized solution define a suitable semantic definition that clarifies the behavior during the whole situation-aware adaptive processing in particular the behavior of automatic adaptations?*

With regard to RQ 1.2 and RQ 1.3, the designed processing model defines when and how adaptations take place:

a) Adaptations take place to start a new FSP Instance if a Possible Situation Indication Event has been classified as New Possible Situation (Figure 4.5.1).

b) Furthermore, adaptations may take place for running FSP Instances *after* the completion of an iteration as part of the preparation of the next iteration (Subsection 4.6.2).

For both kinds of adaptations the processing model requires the scenario processing template to specify the tuple of Focus Area, Locked Area and Time Frame which is used to implement the adaptation and to provide the synchronization between multiple Focused Situation Processing Instances. Based on this the processing model defines the behavior of such adaptation steps.

These adaptation processes are demonstrated in particular by the Test Cases 1, 3 and 4 (Sections 7.4, C.2 and C.3). The definition of the scenario-specific aspects of the adaptation is demonstrated by the realization of the two application scenarios (Section 7.3 and 7.5). Both scenarios require the initial adaptation of the processing system to start new FSP Instances and the later adaptation of their running FSP Instances.

RQ1.4: *How can situation-aware adaptive processing tasks be specified for a generalized processing system?*

This work defines the Scenario Processing Template Language (SPTL) (Chapter 5) which allows the definition of Focused Situation Processing Tasks.

In order to demonstrate the suitability of the language, the language has been used to specify two scenarios (Section 7.3 and Section 7.5). The language itself is domain independent which is also demonstrated by the fact that each of the two realized scenarios originates in a different application domain.

In order to evaluate the suitability of the processing definition, the prototypical implementation uses these definitions for its processing for the five test cases conducted for the evaluation.

7.2. Evaluation Plan

The remainder of this chapter will evaluate the processing model and language by applying it to two scenarios. Each scenario is realized by describing the necessary scenario-specific processing steps in a Scenario Processing Template. Based on these templates and the processing system prototype, several tests are conducted to demonstrate and verify the different aspects of the processing model. The whole evaluation process is structured as follows:

7.2.1. Evaluation Part 1: Cloud Tracking Scenario

Realization of Cloud Tracking Scenario *(Section 7.3)*

Realization of the Smart Grid Cloud Tracking scenario as discussed in Section 2.1.1. The scenario realization demonstrates how the SPTL can be used.

Based on the created scenario template, four test cases where executed in order to demonstrate the processing models functionalities as well as to point out its limitation towards the tracking of non uniquely identified situations (Case 4). The test cases are structured as follows (Figure 7.1.1 provides an overview):

Case 1: Detection and Tracking of a Single Cloud *(Section 7.4)*

The first test case demonstrates the main functionality of the processing model to detect, verify and track a situation based on the example of a single cloud passing through an area with monitored solar panel installations. Thereby the test case discusses the following aspects of the processing model:

- The Stream Processing Topology Builder for setting-up the Possible Situation Indication and FSP Iteration stream processing topologies.
- The Possible Situation Indication Stream Processing and the generation of Possible Situation Indication Events.

- The Focused Situation Processing Initialization with regard to the Possible Situation Indication classification process and the initial adaptation of the processing system by starting a new FSP Instance.
- The Focused Situation Processing with regard to the iterative processing, including the verification of the possible situation, the adaptation of the FSP Instance over time as well as the publication of interim processing results and the termination of the FSP Instance.

While the initial test case demonstrates the positive case for the detection of a valid situation and its analysis, the second case demonstrates the negative case where Possible Situation Indication Events where raised for non-existing clouds.

Thereby, the test case demonstrates in particular:

Case 2: False Situations(Section C.1)

- The situation verification by a started FSP Instance.
- The generation of processing results, in this case the generation of False Situation events.

Case 3: Single Cloud Bigger Than the Initial Focus Area (Section C.2)

The third test case demonstrates the impact of „big" situations which are not fully covered by the initial Focus Area used to classify the raised Possible Situation Indications. As a result the test case *demonstrates the merging* of two FSP Instances which were created for the same situation (Cloud).

Thereby, the test case demonstrates in particular:

- The merging of multiple FSP Instances.
- The adaptation of FSP Instances in order to fully cover a situation.

The final test case based on the cloud tracing scenario, consists of two parts. In Part 1 it demonstrates the handling of multiple (two) separate situations by the processing model. Part 2 discusses the temporary overlap of these two situations. Based on this temporary overlap the limitation of the model to handle the ambiguity of tracked situation identities is demonstrated.

Thereby, the test case demonstrates in particular:

Case 4: Multiple Clouds with a Collision Due to a Temporary Overlap(Section C.3)

- The detection and tracking of multiple separate situations.
- The FSP Instance merging and splitting.
- The limitation of the processing model with regard to handling a temporary situation overlap due to the ambiguity of the situation identities.

7.2.2. Evaluation Part 2: Telco Denial of Service (DoS) Detection and Tracing

Realization of the DoS Detection and Tracing Scenario *(Section 7.5)*
Realization of the Telecommunications Network Monitoring for the Denial of Service Attack Scenario (Subsection 2.1.2.1) in order to demonstrate the applicability of the processing model and language to *another domain*.

Based on the scenario template, the fifth test case was conducted:

Case 5: Detection and Tracing of a DoS Attack *(Section 7.6)*
Test Case 5 is based on the realized DoS Detection and Tracing Scenario Template and demonstrates the application of the processing model to another application domain. In doing so, it further demonstrates the use of topology based background knowledge.

7.3. Cloud Tracking Scenario Realization

This section discusses the realization of the Cloud Tracking Scenario (Section 2.1.1) based on the developed processing model and Scenario Processing Template Language. The following Section 7.4 then discusses the resulting behavior of the processing model based on a test case and the implemented processing system prototype. Furthermore three additional test cases based on the Cloud Tracking Scenario are discussed in Appendix C.

7.3.1. Scenario Realization

The Cloud Tracking Scenario requires the processing system to monitor a potentially large number of solar panel installations for a drop in their energy production (Phase 1, Possible Situation Indication) in order to detect and indicate potential clouds. Based on this it needs to verify that the indication is valid and determine the clouds size and trajectory (Phase 3, Focused Situation Processing). The following subsections discuss the realization this process based on the three phases of the processing model.

The following descriptions are limited to only the relevant aspects of the processing description, a full Scenario Processing Template is given in Appendix B.1.

The processing is based on energy production measurement events provided by the Test Data Simulation (Subsection 7.3.2). For this simulation each measurement event contains the relative energy production of one solar panel installation. To simplify the scenario realization, the provided values are considered to be normalized as a relative value to the maximum production of the specific solar panel installation. Further the normalization values are considered to be independent of lower production in the morning an evening as the sun raises or sets.

7.3.1.1. Phase 0&1: Possible Situation Indication

In order to detect possible situations (clouds), single solar panels need to be monitored for a rapid decrease of their energy production as they become shaded by a cloud. This detection needs to be done for all solar panels where measurement data is available. The selection of these solar panels is specified in the processing template as a SPARQL query which selects the nodes that should be monitored and assigns the result to the pre-defined variable *$$indicationNodes* as shown in Listing 7.1 Line 2.

The stream processing needed for each solar panel that detects the rapid energy production decrease, is expressed as a Stream Processing Builder in Lines 4 to 20. The builder contains a single rule template as shown in Lines 6 to 18 which contains a placeholder (*$$pv*) for the actual solar panel to be monitored. The value for this placeholder is provided by the surrounding foreach statement (Line 5) which iterates over all nodes in *$$indicationNodes* which have been selected for the possible situation indication processing. When the builder is executed it will thus generate a single stream processing rule based on the given rule template for each node selected by the initial SPARQL query.

The stream processing rule template specifies that a window over two of the events received from the PVPowerProduced event stream of the monitored panel should be created. As aggregation function for the time frame, the custom „SuddenChangeDetector" aggregation function is used. The output of the aggregation function represents the difference between the maximum value and the minimum value of the measurements provided in the event stream. If the resulting delta is greater than 50[2] (Line 14), the stream processing rule body is executed. The rule body specifies that a new indication should be published for the monitored solar panel (Line 16). The raised indications are then handled by the next processing phase, the Focused Situation Processing Initialization.

Listing 7.1: Possible Situation Indication for Cloud Tracking

```
PossibleSituationIndication {                                                    1
  $$indicationNodes from sparql "?VALUE rdf:type smartgrid:device. ?VALUE        2
      smartgrid:providesMeasurement ?point. ?point rdf:type smartgrid:
      PVPowerProduced.";
                                                                                 3
  IndicationStreamProcessingBuilder{                                             4
      foreach $$indicationNodes as $$pv {                                        5
        rule [DROOLS_TEMPLATE]                                                   6
            when                                                                 7
                Number( $delta : doubleValue )                                   8
            from accumulate(                                                     9
              MeasurementEvent( $val:value )                                     10
              over window:length( 2 )                                            11
              from entry-point "$${{pv?PVPowerProduced}}",                       12
              SuddenChangeDetector( $val )                                       13
```

[2]The power production measurement values in the PVPowerProduced events is the solar panels output in percent in relation to the total possible output of this panel. Thus, the given difference threshold refers to a drop in production of 50%.

```
          ) eval($delta > 50)                                                    14
              then                                                               15
                publishIndication( "$${pv}" );                                   16
              end                                                                17
          [/DROOLS_TEMPLATE] publishes indications;                              18
    }                                                                            19
  }                                                                              20
};                                                                               21
```

7.3.1.2. Phase 2: Focused Situation Processing Initialization

Based on the raised possible situation indication events, Focused Situation Processing Instances need to be started, however *only if* a received indication event concerns a new possible cloud. As specified by the processing model, the Focused Situation Processing Initialization phase provides the general mechanism to classify indications in order to decide if they are to result into a new Focused Situation Processing Instance. As defined in Section 4.5 (Figure 4.5.1), the classification is based on the following steps which are discussed for this scenario in the following subsections:

1. De-Duplication
2. Potential Locked, Focus Area and Time Frame determination
3. Collision Detection & Collision Classification

Pre-Classification

The first part of the process is the pre-classification to filter out reoccurring events within a specified time frame. For this scenario, assuming that not another cloud needs to be detected for any given panel earlier than 5 minutes after a previous cloud shaded this panel, all indication events within 300 seconds after an indication was received for a given panel can be considered as duplicates and are thus filtered out.

As the general pre-classification mechanism is scenario independent, only the scenario-specific time frame needs to be specified in the template as shown in Listing 7.2 Line 2.

Potential Locked, Focus Area and Time Frame determination

For the classification of a raised possible situation indication, a potential Locked Area and Time Frame is needed (Section 4.5.2). Following Definition 4.3, the Locked Area has to represent the identity of the occurring situation. For a newly indicated possible situation, the nodes contained in the indication can be considered as representing the identity of the potential situation as these nodes represent the solar panel(s) currently shaded by the

potential cloud. Thus, the **potential Locked Area** can be set to the set of nodes from the possible situation indication event as defined in Line 4.

Further a **potential Focus Area** needs to be defined that may be used by the Focused Situation Processing Instance's first iteration, if the classification results in a new Instance being started. The initial Focus Area for this scenario is based on the selection of all solar panels in geographical proximity to the indicated nodes from the possible situation indication event. As such the initial Focus Area is defined by a SPARQL query based on the contents of *$$indicatedNodes* as shown in Line 6.

The **initial Time Frame** can be defined as a time frame starting at the time indicated by the possible situation indication event and ending 300 seconds later in order to cover a large enough time frame to verify that the solar panel's production stays low and was not just temporarily dropping due to some fluctuation (Line 17).

Based on the potential Locked Area and initial Time Frame, the collision detection and classification can take place.

Collision Detection & Collision Classification

The collision detection itself is defined by the processing model in a scenario independent way and needs no scenario-specific parametrization. However, the classification of the results is partly scenario-specific (see Section 4.5.4).

The template language provides two separate possibilities to specify the scenario-specific parts of the collision classification. The first option is to specify them as an MVEL function while the second option is based on a number of rules (Section 5.5.3). For this scenario, the classification is specified rule based as the additional functionality available through MVEL is not needed.

The classification rules need to be specified based on the collision grade between the potential Locked Area of the Possible Situation and the Locked Area and Focus Area of already running Focused Situation Processing Instances.

The Possible Situation Indication Processing (Phase 1) of the cloud tracking scenario and the following Locked Area determination will always result in a potential Locked Area which only contains a single node. Thus, only three cases need to be distinguished for the collision classification:

1. No overlap of the potential Locked Area at all.
2. Full overlap of the potential Locked Area with the Focus Area of an already running Focused Situation Processing Instance.
3. Full overlap of the potential Locked Area with the Locked Area of an already running Focused Situation Processing Instance.

For the first and last case, the processing model defines a fixed scenario independent behavior[3]. Only the second case is scenario-specific and needs to be expressed as a rule. For this case, the indication is considered to be related to the colliding Focused Situation Processing Instance and thus no new processing instance must be created. This behavior is specified by the collision action rule in Line 19[4].

The specification of the Collision Action rule concludes the specification needed for the Focused Situation Processing Initialization.

Listing 7.2: Focused Situation Processing Initialization for Cloud Tracking

```
FocusedProcessingInitialization {                                                    1
  duplicationThreshold 300s;                                                         2
                                                                                     3
  potentialLockedArea $$indicatedNodes;                                              4
                                                                                     5
  potentialFocusArea from sparql "$${{indicatedNodes}} smartgrid:hasLocation ?LOC1   6
      . ?VALUE smartgrid:hasLocation ?LOC2.
  ?LOC1  smartgrid:hasLat ?LAT1.                                                     7
  ?LOC2  smartgrid:hasLat ?LAT2.                                                     8
  ?LOC1  smartgrid:hasLon ?LON1.                                                     9
  ?LOC2  smartgrid:hasLon ?LON2.                                                     10
                                                                                     11
  FILTER ( ?LAT1+0.0035 > ?LAT2 ) .                                                  12
  FILTER ( ?LAT1-0.0035 < ?LAT2 ) .                                                  13
  FILTER ( ?LON1+0.0035 > ?LON2 ) .                                                  14
  FILTER ( ?LON1-0.0035 < ?LON2 ) . ";                                               15
                                                                                     16
  initialTimeFrame startsAt $$indicatedTime withDurationOf 300s ;                    17
                                                                                     18
  collisionAction preventNew if FA overlap == 100% ;                                 19
};                                                                                   20
```

7.3.1.3. Phase 3: Focused Situation Processing (FSP)

If the Phase 2 Focused Situation Processing Initialization classified an Indication as a *New Possible Situation* (Subsection 4.5.5), a new Phase 3 Focused Situation Processing (FSP) Instance is started by the processing system. For the Cloud Tracking Scenario, the FSP has to provide the following functionality:

1. Verify that the indicated possible situation really concerns a cloud.
2. Determine the position and size of the cloud.
3. Follow the cloud over time to allow determining its speed and trajectory.

[3]For the first case, the model defines the fixed scenario independent classification as *New Possible Situation*. For the third case the model defines the fixed scenario independent classification as *Additional Indication*.

[4]The specification of this rule is needed as the default behavior if no collision-handling rules are specified (defined in Subsection 5.5.3) is to trigger a new FSP Instance in this case.

The verification is performed by the very first FSP Iteration by identifying all shaded solar panels in the initial Focus Area and determining if a cluster of more than one solar panel can be found.

Once the situation was verified, determining the position and size as well as following the cloud when it moves, is realized by the following process (Figure 7.3.1):

Step 1: Generate and execute a Stream Processing Topology for the current Focus Area which checks for each node (Solar Panel) from the Focus Area if it is shaded during the current time frame. If a panel was shaded, it is added to the set *$$positiveNodes*.

Step 2: After the stream processing, all nodes from *$$positiveNodes* are clustered based on their geographical location. The biggest cluster resulting from this process is assumed to represent the currently tracked cloud. The nodes of *this* cluster are added to the set *$$verifiedNodes* (performed during the Post-Processing Step).

Step 3: If *$$verifiedNodes* contains less than two nodes, terminate the processing as no valid cloud could be found. For this case, two separate reasons can be distinguished:

a) If this was the first iteration, terminate the processing as a *FalseSituation* as the cloud was not found and thus the indicated possible situation could not be verified.

b) If this was the second or a later iteration, terminate the processing as *Cloud Lost* as the cloud was tracked for a number of iterations but now disappeared.

Step 4: Prepare the next iterations Locked and Focus Area and Time Frame. Here again two cases can be distinguished:

a) If the nodes in *$$verifiedNodes* are the same as in the current Locked Area, the Locked Area of this iteration was correctly representing the situation (cloud) and the processing for the current Time Frame is thus finished and interim results on the clouds position can be published.
For the next iteration: Generate the next consecutive Time Frame as the Time Frame for the next iteration and set the next iterations Locked and Focus Area to the same values as the current iteration.

b) If the nodes in *$$verifiedNodes* are different from the current Locked Area, the current iterations Locked Area was incorrect and the processing of the current Time Frame needs to be repeated with an updated Locked Area.
For the next iteration: Keep the current Time Frame, set the new Locked Area to *$$verifiedNodes* and generate the new Focus Area accordingly.

Step 5: Repeat the process from Step 1.

The remainder of this subsection discusses the realization of these steps in the scenario template.

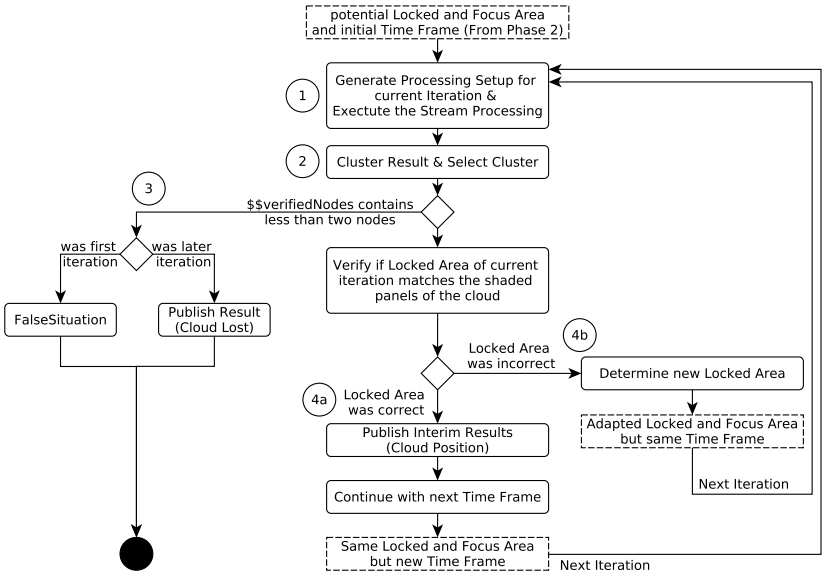


Figure 7.3.1.: Locked Area and Focus Area adaptation Process for the Cloud Tracking Scenario

Context Initialization and Pre-Iteration Processing

The cloud tracking scenario requires no special context initialization or pre-iteration processing for the normal processing flow. If however the current FSP Instance was merged with another FSP Instance during the last iteration, a pre-iteration processing is needed in order to incorporate processing results from the colliding FSP Instance. The following listing shows the needed context initialization and pre-iteration processing functions which are discussed in Section 7.3.1.3:

```
contextInitialization [MVEL]
  $$oldLAofMergedFPI = null;
[/MVEL];

  preIteration [MVEL]
  $$positiveNodes = new java.util.HashSet();
  $$nodesToConsider = $$focusArea;

  if($$oldLAofMergedFPI!=null){
    // remove the nodes that were already verified
    java.util.Iterator iter = $$oldLAofMergedFPI.iterator();
    while (iter.hasNext()) {
      $$nodesToConsider.remove(iter.next());
    }
    $$positiveNodes.addAll($$oldLAofMergedFPI);
    $$oldLAofMergedFPI = null;
  }
[/MVEL];
```

Stream Processing Topology Generation

For the actual Focused Situation Processing, each iteration's stream processing has to determine which nodes in the Focus Area are shaded. In order to determine this, the stream processing is defined as follows for each of the nodes in the set *$$nodesToConsider* which contains all nodes of the Focus Area except nodes that where already verified:

```
IterationStreamProcessingBuilder {
      foreach $$nodesToConsider as $$pv {
      rule [DROOLS_TEMPLATE]
          when
            Number( $average0 : doubleValue )
              from accumulate(
                MeasurementEvent( $val:value )
                over window:time( 300s  )
                from entry-point "$${{pv?PVPowerProduced}}",
                average ( $val )
                ) eval($average0 < 30.0 )
                then
            if(CONTEXT.noEarlyFiring("150s")){
                    CONTEXT.addToSet("$$positiveNodes","$${{pv}}");
            }
                end
        [/DROOLS_TEMPLATE] publishes no stream manipulates context;
    }
  };
```

The stream processing builder generates a single stream processing rule for each solar panel in *$$nodesToConsider*. The stream processing rule then checks if the average energy production over 300s is below 30% of the panels capacity. If the production is below this threshold, the panel is considered shaded and the corresponding node is added to the set *$$positiveNodes*.

After the stream processing is complete, the set *$$positiveNodes* contains all solar panels in the current Focus Area that have a low energy production during the current iterations Time Frame.

Iteration Post Processing

Based on the set *$$positiveNodes*, the post processing step determines geographical clusters were in the optimal case only one cluster is found which represents the currently tracked cloud. If however more than one cluster was found, more than one cloud is shading panels in the current Focus Area. In this case one of the clusters needs to be chosen as representing the current cloud. For this scenario, the largest cluster in terms of solar panel count is used[5]. All nodes of the selected cluster are then assigned to *$$verifiedNodes* so that they can be used by the following steps.

[5] An alternative strategy could be to choose the cluster closest to the last known position of the tracked cloud.

The clustering itself is performed by a domain specific function *findClusters* (Defined in B.1.1). The function also calculates the center of each cluster which is extracted by the post processing step and used as new position of the tracked cloud. The post processing function is thus defined as follows:

```
postIteration [MVEL]
  $$verifiedNodes = new java.util.HashSet();

  $$resLon = -1;
  $$resLat = -1;

  if( $$positiveNodes.size() > 0 ){
    clusters = es.schaaf.cloudTracking.GeoNodeClustering.findClusters( CONTEXT ,
        $$positiveNodes , 0.003 );
    // There must be at least one cluster.
    // The first one is always the biggest one.
    $$verifiedNodes.addAll(clusters[0].members);
    $$resLon = clusters[0].centerLon;
    $$resLat = clusters[0].centerLat;
  }

  $$positiveNodes.clear();
[/MVEL];
```

After the post-processing, the current position of the cloud is known by the processing system and can be published as an interim processing result thus making it available to external systems. However the position should only be published, if the current Locked Area correctly represented the cloud and contains more than a single node. For this purpose, the following publish rule can be defined:

```
publish result "$$verifiedNodes", "$$resLon", "$$resLat" when "$$verifiedNodes.
    size()>1 && $$verifiedNodes.equals( $$lockedArea)";
```

Termination

Furthermore, the termination of the Focused Situation Processing needs to be defined. For the termination, the following two cases need to be distinguished:

No cloud was detected

The *first* iteration of the Focused Situation Processing was not able to find a cluster of more than one shaded solar panel within the initial Focus Area. Thus, the possible situation was determined as a False Situation. For this case, the following termination rule is defined:

```
terminate if [MVEL] $$verifiedNodes.size() < 2 && $$iterationCounter < 2 [/
    MVEL] with result FalseSituation keep area registration if [MVEL]true[/
    MVEL];
```

As the indicated situation was determined as a False Situation, the termination rule requests that the initial Area Registration is being kept in order to mark the False Situation as such and prevent other FSP Instances from being created for it.

Cloud disappeared

The possible situation was successfully verified in the first iteration of the Focused Situation Processing and might have been tracked for a while but now disappeared for example by leaving the monitored area. In order to handle this case, the following rule can be defined:

```
terminate if [MVEL] $$verifiedNodes.size() < 2 && $$iterationCounter >= 2 [/MVEL] with result $$verifiedNodes, $$resLon, $$resLat keep area registration if [MVEL]false[/MVEL];
```

As the Focused Situation Processing terminated due to the fact that the cloud could not be found anymore during this last iteration, the rule releases the Area Registration as this last iteration was not correctly marking the situation anymore.

Next Iteration Preparation

If the Focused Situation Processing was not shut down by the termination rules, the Locked Area, Focus Area and Time Frame for the next iteration need to be determined. Here also two scenario-specific cases can be distinguished (Figure 7.3.1):

Current Iteration Locked Area was Correct:

The Locked Area of the current iteration was successfully verified as fitting the situation for the current iterations Time Frame. For this case, the next iteration can look at a new Time Frame, which is following the Time Frame to the current iteration. The assumption for the clouds position for the new Time Frame is, that it is still at the same position as for the current Time Frame. Thus, the Locked Area and Focus Area of the next iteration will be the same as for the current, only the Time Frame is advanced.

Current Iteration Locked Area was Incorrect:

The Locked Area of the current iteration was not correctly fitting the cloud for the current iterations Time Frame and thus needs to be adapted. For this case, the next iteration needs to cover the same Time Frame as the current iteration but with an updated Locked and Focus Area. The new Locked Area equals the nodes from the set *$$verifiedNodes* which are the result of the iteration post processing, the Focus Area then consists of the nodes from the geographical area around the Locked Area.

Thus, the next iteration Locked Area, Focus Area and Time Frame determination can be defined as follows:

```
nextIterationTimeFrame startsAt [MVEL] if( $$verifiedNodes.equals($$lockedArea) {
    $$endTime } else { $$startTime } [/MVEL] withDurationOf 300s ;

nextLockedArea $$verifiedNodes;

nextFocusArea from sparql " $${verifiedNodes} smartgrid:hasLocation ?LOC1. ?VALUE
    smartgrid:hasLocation ?LOC2.
  ?LOC1  smartgrid:hasLat ?LAT1.
  ?LOC2  smartgrid:hasLat ?LAT2.
  ?LOC1  smartgrid:hasLon ?LON1.
  ?LOC2  smartgrid:hasLon ?LON2.

  FILTER ( ?LAT1+0.0035 > ?LAT2 ) .
  FILTER ( ?LAT1-0.0035 < ?LAT2 ) .
  FILTER ( ?LON1+0.0035 > ?LON2 ) .
  FILTER ( ?LON1-0.0035 < ?LON2 ) .
  ";
```

Situation Merging

In order to support the merging of two FSP Instances that turned out to be investigating the same situation, in this case the same cloud, a merge function needs to be specified (See Sub-Section 5.6.9). The function is called once for the two processing instances that tried to acquire an overlapping Locked Area for an overlapping Time Frame. In such a case the processing model assumes that the two instances regard the same situation. Thus, the processing model terminates one of the instances (Instance „A") and lets the other continue with the processing (Instance „B") after A was merged into B.

To facilitate the merge, a merging function can retrieve information from the processing context of the terminated processing instance A and incorporate it into the processing context of Instance B.

For this scenario the merging function retrieves the most recent Locked Area from the FSP Instance A that is to be terminated and assigns it to the variable *$$oldLAofMergedFPI* in the context of the FSP Instance B that continues the situation processing. The merge function can thus be defined as follows:

```
mergeFunction [MVEL]
    if(CONTEXT_A.get("$$timeFrame").equals(CONTEXT_B.get("$$timeFrame"))){
      la = CONTEXT_A.get("$$lockedArea");
      CONTEXT_B.put("$$oldLAofMergedFPI",la);
    }
  [/MVEL];
```

The Locked Area is taken from FSP Instance A as all nodes in it have already been verified by this instance as being shaded. Therefore the FSP Instance B can directly incorporate the set if nodes into its ongoing processing without checking them once again. This exclusion is done during the preparation of the next iteration of the FSP Instance B in its Pre-Iteration function as shown in Section 7.3.1.3 where all nodes in $$*oldLAofMergedFPI*

are removed from the set of nodes which are considered by the following iteration stream processing.

Based on the defined Template, several tests where conducted following the evaluation plan in Section 7.2. The Tests and their results are discussed in Section 7.4 and Appendix C.

7.3.2. Test Data Simulation

For conducting the cloud tracking test cases based on the prototype processing system, test data was generated based for a grid of 10 x 10 evenly distributed solar panels (Figure 7.4.2). Energy production measurements where produced for every 30 seconds of the simulated time frame. Depending on the actual test case, one or more clouds moving over the solar panels where simulated with varying speeds, sizes and trajectories specific for the test case.

The simulation provides the measurement data as a number of CSV files which are loaded by the processing systems prototype. Further the simulator generates the background knowledge base contents based on the positions of the simulated solar panels as a Turtle file that is also used by the prototype.

To verify the cloud tracking process, the simulation also creates a log of the positions of the simulated clouds which is however *not* available to the processing system but only displayed in the result visualization in order to compare the reported cloud positions from the processing system with the actual positions from the simulation.

7.4. Case 1: Single Situation Detection and Tracking

The goal of the first test case is to determine the general capabilities of the designed processing model. The test case demonstrates the tracking of a single cloud across a field of solar panels based on the scenario processing template discussed in the previous section (Complete listing in Appendix B.1). The execution of this test case demonstrates that the processing models fulfills the following initially defined requirements (Section 2.3):

RQ1: *Support to set up a situation indication processing that can handle large amounts of streaming data.*

The generation of Stream Processing Topologies for the Possible Situation Indication is demonstrated by the test case during Phase 0 (Subsection 7.4.1). Several independent rules are generated that can be executed in parallel in order to handle large amounts of streaming data (on multiple machines) where only their processing results are gathered together in a single result stream.

RQ2: *Support to deduce and initiate an analysis processing for a detected situation, where the analysis processing is specific for the detected situation.*

The classification of the generated indications which leads to the creation of a new FSP Instance is demonstrated during Phase 2 (Subsection 7.4.3). Based on it the adaptation of the processing system to instantiate a new situation-specific FSP Instance for the indicated cloud is shown as well as the adaptation of the started FSP Instance to cover the whole cloud during Phase 3 (Subsection 7.4.4).

RQ3: *Support to handle changes of a currently investigated situation that require the adaptation of the processing of an ongoing situation-specific analysis based on interim results.*

For the running FSP Instance, the adaptation to follow the changes of the clouds position is demonstrated during Phase 3 (Subsection 7.4.4).

Furthermore, general functionalities like the Area Registration based collision detection and classification, the verification of an indicated possible situation, the publication of interim processing results and the termination of the FSP Instance after the situation was tracked are demonstrated.

The following sections discuss the execution of the Focused Situation Processing based on the generated test data for a single cloud based on the Scenario Processing Template (B.1) defined in the previous section.

7.4.1. Phase 0: Possible Situation Indication Processing Initialization

In order to set up the Possible Situation Indication Processing, the Indication Stream Processing Topology Builder from the Template discussed in the previous section (Listing B.1) needs to be executed. As a preparation, the set of nodes that need to be monitored for possible situation indicators is determined by executing the defined SPARQL query. For this template, the query returns all solar panels from the background knowledge base which provide measurement data. The nodes are assigned to the variable *$$indicationNodes*.

Based on the *$$indicationNodes* the Situation Indication Stream Processing Topology is generated, following the definition given in the Template in the „IndicationStreamProcessingBuilder" block. The builder generates one stream processing rule for each node in *$$indicationNodes*.

Each of the rules takes the event stream „PVPowerProduced" for the corresponding solar panel as input and directly generates Possible Situation Indication events if an indicator for a cloud (a sudden drop of the solar panels energy production) is found. Thus, the resulting Stream Processing Topology consists of n parallel stream processing rules for n solar panels all with their separate input stream and all emitting possible situation events to the Possible Situation Indication event stream (Figure 7.4.1).

Listing 7.3 gives an example for a such a processing rule generated by the builder function for the solar panel „tests:panel_89" which was assigned to the variable *$$pv*. The

Listing 7.3: Generated Stream Processing Rule for the Situation Indication by the corresponding builder function

```
rule rule_0                                                                    1
    when                                                                       2
        Number( $delta : doubleValue )                                         3
            from accumulate(                                                   4
            MeasurementEvent( $val:value )                                     5
            over window:length( 2 )                                            6
            from entry-point "dfcfbc80e5c1b1c32f4a5ad6023053bc",               7
            SuddenChangeDetector( $val )                                       8
        ) eval($delta > 50)                                                    9
    then                                                                       10
        __INDICATION_HANLDER__.handleIndication( "tests:panel_89" );           11
    end                                                                        12
```

listing shows that the entry-point was replaced by a unique identifier specific to the prototype. Based on this identifier, the prototype stream the measurement events to the rule. Further the generation of the Possible Situation Indication event has been replaced by an implementation specific code block to generate and publish the actual indication event, in this case for the solar panel „tests:panel_89".

7.4.2. Phase 1: Possible Situation Indication Processing

The Stream Processing Topology defined in Phase 0 is instantiated by the processing system and monitors all solar panels for a sudden drop of their energy production.

For the test data set, a single cloud enters the monitored area from the west and starts to shade the Solar Panels 4 and 5 at time 1438293900. The corresponding stream processing rules detect the drop in production and raise possible situation indication events (Table 7.4.1). For Panels 4 and 5 each one of the first indications is correctly raised for time 1438293900 when the first low measurement occurred as this results in the rapid change from 100% production to 0% (Line 1 & 2).

As the defined indication only looks at two events to detect a delta in the power production of at least 50%, only a single indication event is raised for each panel.

7.4.3. Phase 2: Focused Situation Processing Initialization

The indications raised by the Possible Situation Indication Processing (Phase 1) are classified in this phase. The results of the performed classification are listed in Table 7.4.1 in the column „P2 Classification". For the first Indication Event (Table 7.4.1 Line 1) the classification took place as follows:

Step 1: The Pre-Classification lets the event pass as it is the first event for this node (Panel 4).

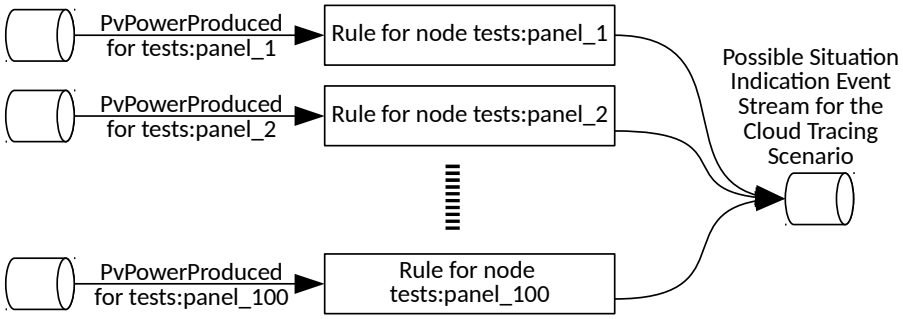


Figure 7.4.1.: Structure of the Stream Processing Topology generated for the Cloud Tracking Possible Situation Indication based on the Scenario Processing Template from Listing B.1.

#	Indicated Time	Indicated Nodes	Initial TimeFrame	Potential LockedArea	P2 Classification	Resulting Focused Situation Processing Instance
1	1438293900	tests:panel_4	1438293900-1438294200	tests:panel_4	New Possible Situation	FP__CloudTracking1__10c40795-b2f2-4b8a-a80d-8b7842e4db45 (#1)
2	1438293900	tests:panel_5	1438293900-1438294200	tests:panel_5	Ignored Indication	
3	1438296330	tests:panel_3	1438296330-1438296630	tests:panel_3	Ignored Ind. (LA Collision)	
4	1438296330	tests:panel_6	1438296330-1438296630	tests:panel_6	Ignored Ind. (LA Collision)	
5	1438299360	tests:panel_15	1438299360-1438299660	tests:panel_15	Ignored Ind. (LA Collision)	
6	1438299360	tests:panel_14	1438299360-1438299660	tests:panel_14	Ignored Ind. (LA Collision)	

Table 7.4.1.: Case 1: Excerpt of the first 6 Possible Situation Indication Events from the Possible Situation Indication Event Log.

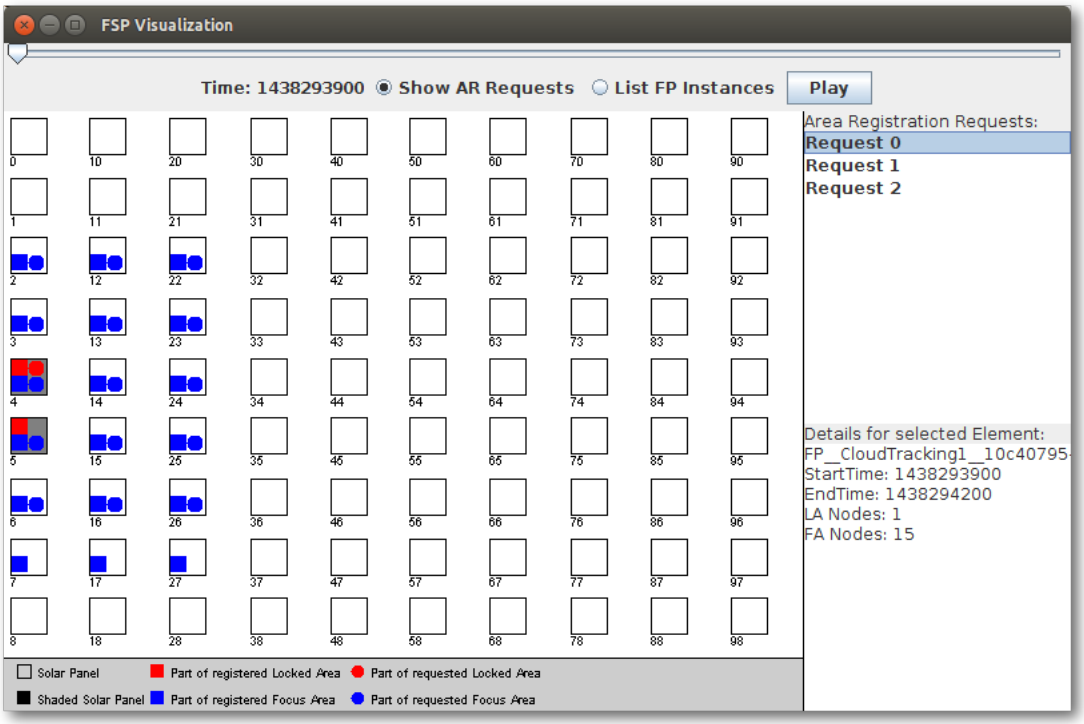


Figure 7.4.2.: Case 1: Area Registration Request caused by the first Possible Situation Indication Event (Participating nodes of the request shown as circles).

Step 2: The potential Locked and Focus Areas and initial Time Frame are determined (The potential Locked Area and initial Time Frame is listed in Table 7.4.1, the potential Focus Area is illustrated in Figure 7.4.2 as part of the Area Registration „Request 0").

Step 3: The processing system tries and succeeds in acquiring the potential Locked and Focus Area for the initial Time Frame without any collisions with other registrations as it is the very first registration.

Step 4: As no collisions with already running Focused Situation Processing Instances where detected, the Indication Event is classified as „New Possible Situation" and a new Focused Situation Processing Instance is created with the ID shown at the end of Line 1 in Table 7.4.1. *For later reference, the ID of this instance shall be „#1".*

After the first Possible Situation Indication Event was classified, the processing system classifies the second event as follows (Table 7.4.1 Line 2):

Step 1: The Pre-Classification lets the event pass as it is the first event for this node (Panel 5).

Step 2: The potential Locked and Focus Areas and initial Time Frame is determined (The potential Focus Area is illustrated in Figure 7.4.3 for Area Registration „Request 2")

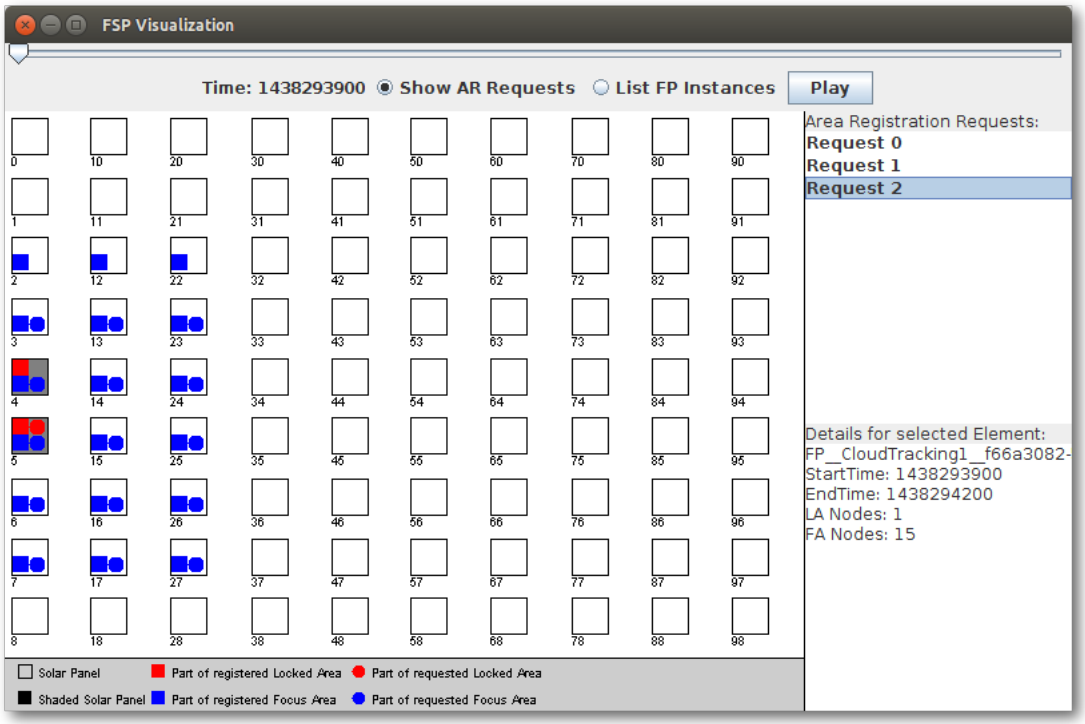


Figure 7.4.3.: Case 1: Area Registration Request caused by the second Possible Situation Indication Event which was rejected as it collides with the initial Area Registration of the Focused Situation Processing Instance #1 (Figure 7.4.2).

Step 3: The processing system tries and succeeds in acquiring the potential Locked and Focus Area for the initial Time Frame *but detected a collision* with the Focus Area of the already running Focused Situation Processing Instance #1.

Step 4: Based on the collision action rule specified in the template, the detected collision results in the classification of the Indication Event as „Ignored Indication" as its potential Locked Area is a subset of the Focus Area of the Focused Situation Processing Instance #1. The Indication Event is thus dropped by the processing system and the acquired Area Registration released.

The third Indication Event shown in Table 7.4.1 Line 3 is classified as follows:

Step 1: The Pre-Classification lets the event pass as it is the first event for this node (Panel 3).

Step 2: The potential Locked and Focus Areas and initial Time Frame is determined (Figure 7.4.4 as part of „Request 13").

Step 3: The processing system tries and *fails* to acquire the Area Registration as the running Focused Situation Processing Instance #1 already acquired the corresponding nodes as part of its Locked Area for the overlapping Time Frame 1438296300 to 1438296600 (Figure 7.4.4 shown as red rectangles).

Step 4: As the registration failed due to a collision, the Indication Event is classified

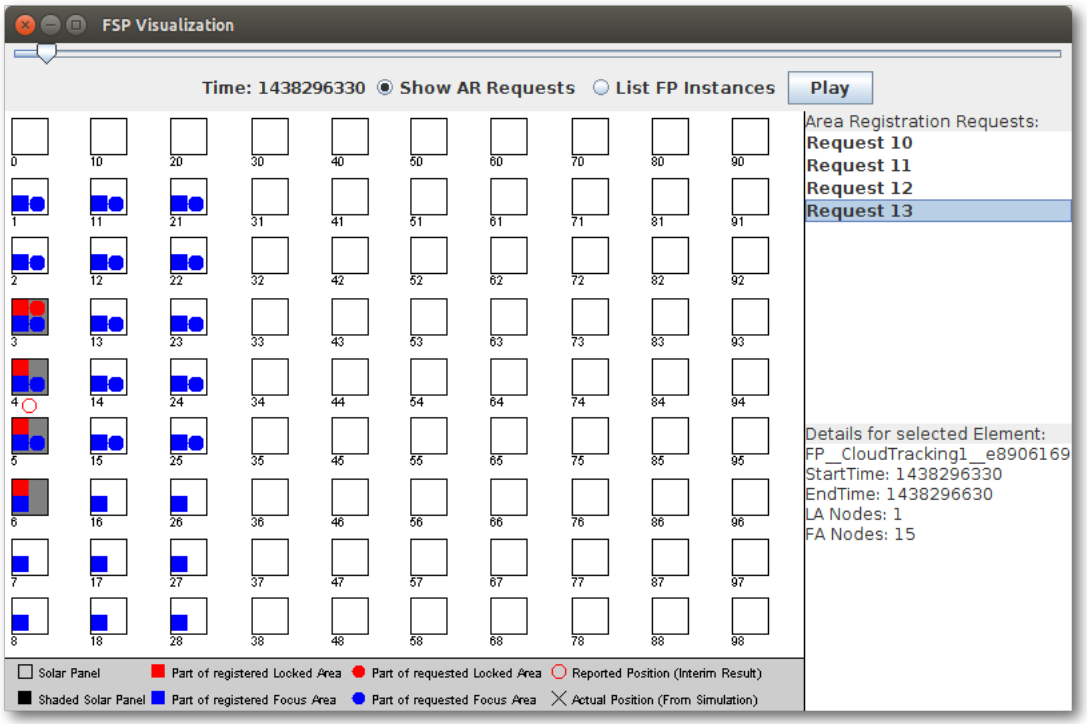


Figure 7.4.4.: Case 1: Area Registration Request caused by the third Possible Situation Indication Event caused by the newly shaded Solar Panel 3 (shown in circles). The registration request was denied as the requested Locked Area overlaps with the Locked Area of the Focused Situation Processing Instance #1 (shown as rectangles).

as „Ignored Indication" and dropped by the processing system.

All following Possible Situation Indication Events generated by the Phase 1 processing follow the same pattern as discussed for the third event.

7.4.4. Phase 3: Focused Situation Processing

For the current test case, a single situation (cloud) needs to be followed by the processing system. For this purpose a single FSP Instance was started by the processing system as the result of Phase 2. The following subsections discuss the first 11 iterations of this FSP Instance where the potential situation is first verified and then later the processing is adapted to follow the movement of the cloud.

7.4.4.1. Context Initialization

The specified context initialization function only initializes the context with the empty variable *$$oldLAofMergeFPI* in preparation for a possible later merge (Subsection 5.6.2).

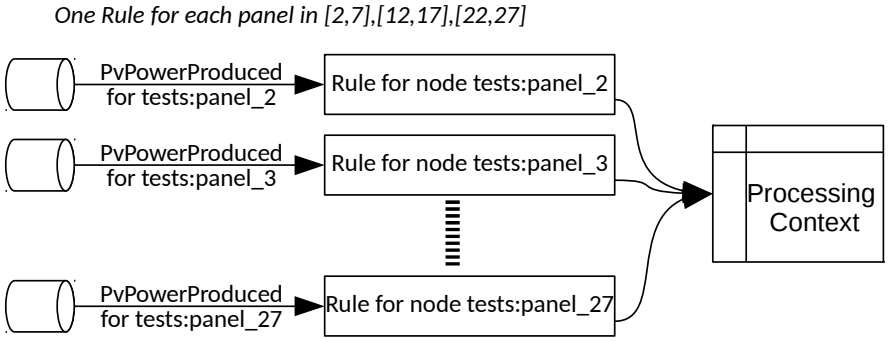


Figure 7.4.5.: Case 1: Stream Processing Topology generated for Iteration 0 of the Focused Situation Processing Instance #1.

7.4.4.2. Iteration 1

For the first iteration, the Focused Situation Processing Instance starts with the Locked Area, Focus Area and Time Frame determined during Phase 2 as shown in Figure 7.4.2 as „Request 0" (marked by the red and blue circles).

The FSP Instance starts with the first iteration of its processing by executing the *Pre-Iteration* function which defines a new variable *$$positiveNodes* and assigns an empty set to it. Further the set *$$nodesToConsider* is set to the contents of the Focus Area.

Based on the resulting processing context, the Iteration Stream Processing Builder function is executed to generate the stream processing topology for the first iteration:

The stream processing builder generates a single event stream processing rule for each node in the current Focus Area where each of these stream processing rules writes its results to the processing context. The stream processing rule calculates the average energy production for the given node (solar panel Installation) over a sliding time frame of 300s and if the production is below a certain threshold adds the node to the set of *$$positiveNodes*.

As the current Focus Area consists of 15 nodes, the builder created 15 separate stream processing rules as outlined in Figure 7.4.5.

After the stream processing topology was created, the stream processing took place. It identified the low energy production of nodes tests:panel_4 and tests:panel_5 and added them to the set of *$$positiveNodes*.

Once finished, the processing results were gathered by the *Post-Iteration* function. The function uses a domain specific function to cluster the results based on their geographical location and to calculate the center of each cluster (Subsection B.1.1). For the two nodes in *$$positiveNodes*, a single cluster was found which consisted of the two nodes tests:panel_4 and tests:panel_5. The post-processing function added the two nodes of the cluster to *$$verifiedNodes* and assigned the clusters position to *$$resLon* and *$$resLat*.

After this post processing step, the termination rules were evaluated but did not match

as *$$verifiedNodes* contained more than one node, so the processing could continue with another iteration.

As the focused processing was not terminated, the interim result publication rules were evaluated. As *$$verifiedNodes* did not contain the same set of nodes as the current iterations Locked Area, the processing state was not yet published as the calculated position was not considered final for the iteration's time frame.

With the evaluation of the interim publication results, all steps regarding the first iteration were completed and the processing system prepared for the next iteration by defining the next iteration's Locked and Focus Area as well as its Time Frame.

As the first iteration's Locked Area was found to be incorrect for the current iterations time frame (`$$verifiedNodes.equals($$lockedArea)`) the second iteration needed to repeat the same time frame (Subsection 7.3.1.3) but with an adapted Locked and Focus Area to check if the adapted version is correct. Thus, the second iteration's time frame was set to the same time frame as the first iteration.

The next Locked Area was set to the contents of *$$verifiedNodes* (which contained the two panels tests:panel_4 and tests:panel_5) and the new Focus Area was set to contain nodes surrounding the nodes of the new Locked Area. The area registration for the second iteration was then successfully requested by the processing system (shown in Figure 7.4.6 as „Request 1"). After the new registration was made, the second iteration began.

7.4.4.3. Iteration 2

The second iteration operated on the adapted Locked and Focus Area and followed the same pattern as the first iteration with the difference that the generated stream processing topology covered a larger set of nodes as the Focus Area size was increased. The result of the stream processing for this iteration was the same as for iteration 1 as no additional nodes were identified as being shaded. As such the Post-Processing function and the termination rules are executed in a similar way.

As however the Locked Area and the determined set *$$verifiedNodes* were the same for this iteration, the calculated cloud position is considered final for the this iteration's time frame and the interim result publication rule matched. Thus, an interim result was generated and published as shown in Line 1 of Table 7.4.2. The event contained the position of the cloud as *$$resLat* and *$$resLon* as well as the time frame of current iteration and the nodes that were detected as being affected by the cloud.

Furthermore, as the current Locked Area equaled the contents of *$$verifiedNodes* and thus correctly represented the situation for the current time frame, the next iteration's time frame was determined as the next consecutive time frame of 300s following the time frame of the current iteration.

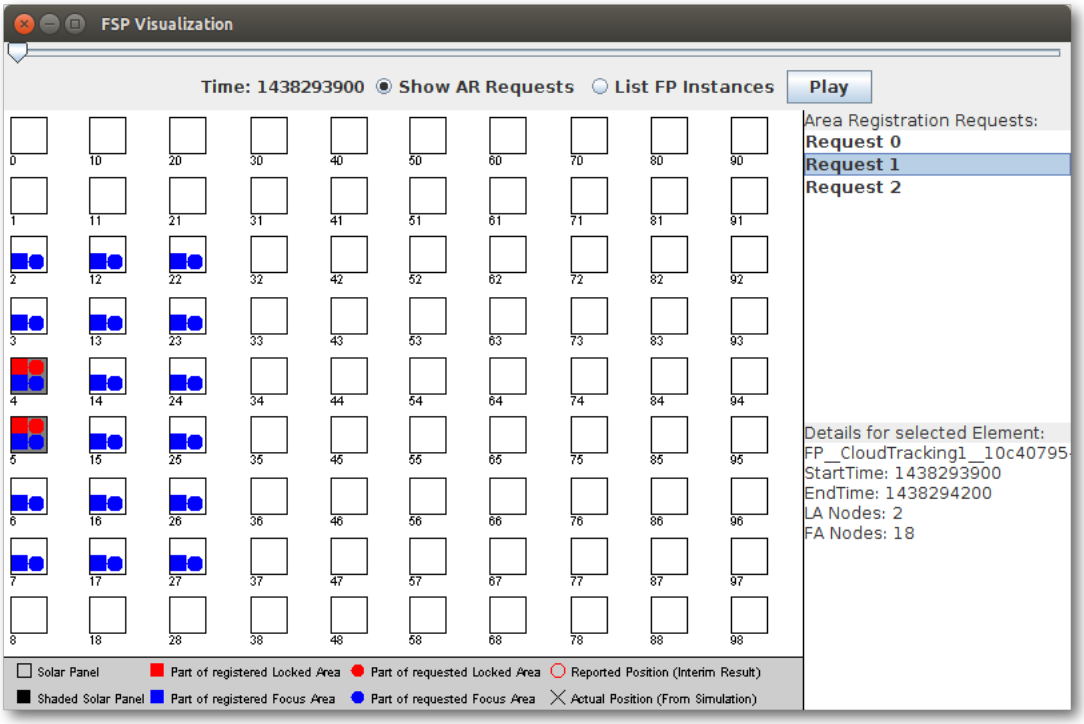


Figure 7.4.6.: Case 1: Request for the updated Area Registration for the Focused Situation Processing Instance #1 for the second iteration (shown in circles) which then covered the *two* shaded solar panels.

#	Event Type	Time	Event Contents
1	InterimResultEvent	1438294200	$$endTime = [1438294200], $$resLat = [47.356595], $$resLon = [7.889175445556641], $$startTime = [1438293900], $$verifiedNodes = [tests:panel_4;tests:panel_5]
2	InterimResultEvent	1438294500	$$endTime = [1438294500], $$resLat = [47.356595], $$resLon = [7.889175445556641], $$startTime = [1438294200], $$verifiedNodes = [tests:panel_4;tests:panel_5]
3	InterimResultEvent	1438294800	$$endTime = [1438294800], $$resLat = [47.356595], $$resLon = [7.889175445556641], $$startTime = [1438294500], $$verifiedNodes = [tests:panel_4;tests:panel_5]
	...	...	...

Table 7.4.2.: Case 1: Excerpt form the Interim Result Events log generated by the Focused Situation Processing Instance #1.

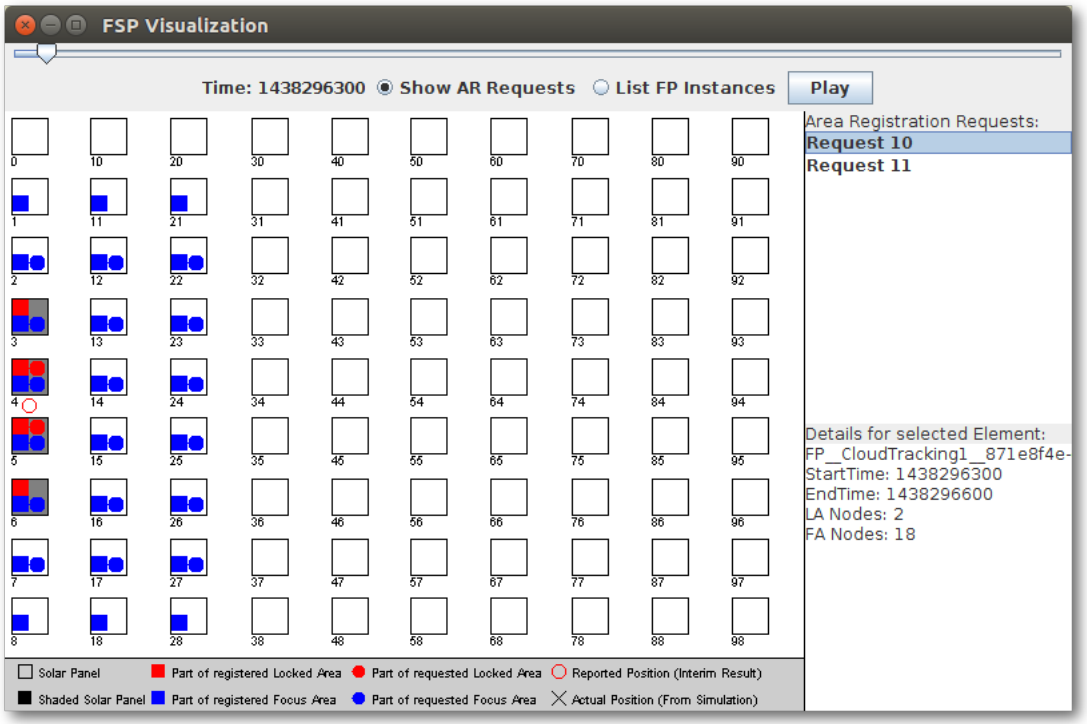


Figure 7.4.7.: Case 1: The Area Registration (circles) used for Iteration 10 of the Focused Situation Processing Instance #1

7.4.4.4. Iterations 3 to 9

The third iteration continued with the same Locked Area and Focus Area as Iteration 2 but with the next consecutive time frame. As the cloud was still shading the same solar panels during this iteration's time frame, no changes to its Area Registration were required and the next iteration time frame was determined as the next consecutive time frame. The next Locked and Focus Areas were set to the same set of nodes as for the current iteration.

Furthermore, the iteration emitted an interim result event to report the verified position of the cloud for the current time frame as shown in Figure 7.4.2 Line 3.

The iterations up to Iteration 9 followed the same pattern as the panels shaded by the cloud did not change during the time frame covered by those iterations.

7.4.4.5. Iteration 10

Within the time frame of Iteration 10 (1438296300 to 1438296600), the Panels 3 and 6 became shaded in addition to the original two panels. In the same way as for the very first iteration, the stream processing detected the newly shaded panels and added them to *$$positiveNodes* together with the so far shaded Panels 4 and 5.

Thus, the set *$$positiveNodes* contained four elements (Panels 3-6) at the end of the iteration stream processing. The post-processing step determined that the four panels

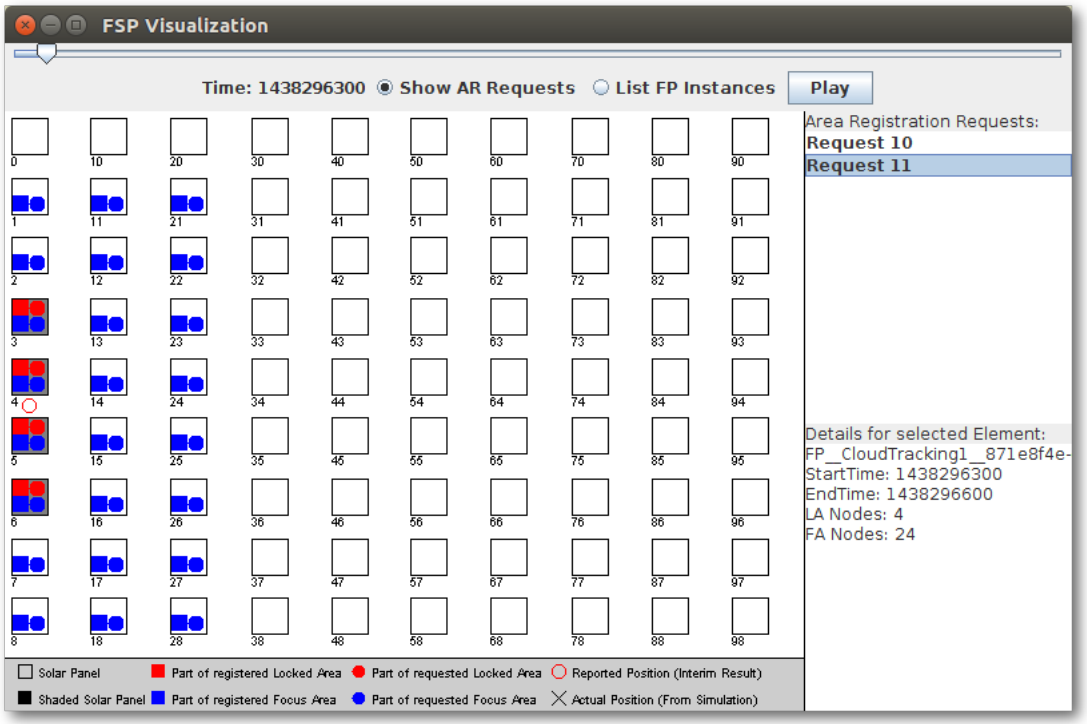


Figure 7.4.8.: Case 1: The Area Registration (circles) used for Iteration 11 of the Focused Situation Processing Instance #1.

form a single cluster and assigned all four nodes to *$$verifiedNodes*.

As again the set *$$verifiedNodes* did not equal the current Locked Area which only contained the Panels 4 and 5 (Figure 7.4.7, Request 10), the current iteration's time frame needed to be repeated by the next iteration with an adapted Locked Area which covered all four panels. Thus, the next iterations time frame was set to the time frame of the current iteration, the next iteration Locked Area was set to *$$verifiedNodes* and the next iteration Focus Area was determined in the specified manner based on the new Locked Area (Figure 7.4.8, Request 11). Once done, the next iteration started in order to verify if the adapted Locked Area was now correct for the current time frame.

7.4.4.6. Iteration 11

Iteration 11 followed once again the same pattern as Iterations 3 to 9 as it determined that the current Locked Area was correct and thus produced an interim result event and prepared for the following iteration-based on the next consecutive time frame with the same Locked Area.

This pattern once again continued until the shaded panels changed as discussed for Iteration 10.

7.4.5. Conclusions from the Test Results

The first test case *successfully demonstrated the central functionality of the processing model* by showing all processing phases defined by the processing model. In particular the test case demonstrated the setup of the indication processing (Phase 0) as well as the detection of potential situations (Phase 1). Based on it the test case showed the classification of potential situations as well as the adaptation of the processing system in order to start a new Focused Situation Processing Instance. Following the start-up, the verification of a potential situation was demonstrated as well as the tracking of the situation over time and with it the generation of interim processing results and the adaptation of a running FSP Instance to follow the situation.

Appendix C discusses three additional test cases that demonstrate special cases aside from the general processing flow based on the cloud tracking scenario discussed here. A summary of the test results is given as part of the overall conclusion for this chapter in Section 7.10.

7.5. Telecommunications Network Monitoring: Denial of Service Tracing

This section discusses the realization of the Denial of Service (DoS) detection and tracing Scenario (Section 2.1.2.1) based on the developed processing model and the corresponding Scenario Processing Template Language. The realized template is then used for the following Test Case 5 (Subsection 7.6).

The following discussions are focused on how the scenario is realized and thus omit detailed explanations of the related principles of the processing model and specification language as they were already discussed for the realization of the Cloud Tracking Scenario in Section 7.3.

A complete listing of the created scenario processing template is given in Appendix B.2.

7.5.1. Scenario Realization

The goal of this scenario is to detect DoS attacks on certain routers that for example connect data centers to the monitored network. Such routers are marked in the knowledge base as „dosMonitoredRouter". Further the DoS traffic shall be traced back to the point where it entered the monitored network so that it can be blocked there before entering the network.

The scenario realization approaches this task in the following way:

Possible Situation Indication *(Detailed in Subsection 7.5.1.1)*

Monitor the average package size for network interfaces of routers that are marked as „dosMonitoredRouter". If the average package size drops significantly, raise a possible situation indication for the interface as this indicates a possible DoS attack.

Focused Situation Processing Initialization *(Detailed in Subsection 7.5.1.2)*

For each raised indications, set the potential Locked Area to the network interface that was reported in the indication event. Thereby multiple indications raised for the same router are automatically assigned to an already started FSP Instance. If no such instance exists, a new FSP Instance is started for the indicated possible DoS attack.

Focused Situation Processing *(Detailed in Subsection 7.5.1.3)*

The analysis of a potential DoS attack is then realized in two separate steps:

Step 1: Verify the possible attack by checking if not only the average package size dropped but also that the average package count increased significantly.

Step 2: Iteratively trace the traffic of the DoS attack trough the network until a router at the network border was reached or the traffic pattern can't be found anymore.

Listing 7.4: Possible Situation Indication for DoS detection

```
PossibleSituationIndication {
  // select all interfaces of routers that are flagged for DoS monitoring
  $$indicationNodes from sparql "?VALUE   rdf:type        telco:interface.
                   ?VALUE   fsp:providesMeasurement ?point.
                   ?point   rdf:type        telco:trafficIn .
                   ?router  telco:hasInterface    ?VALUE .
                   ?router  rdf:type        telco:dosMonitoredRouter .
                   ";

  IndicationStreamProcessingBuilder{
      foreach $$indicationNodes as $$interface {
        rule [DROOLS_TEMPLATE]
          when
              // package size dropped
              $b : SuddenChange( consideredEvents == 6 , delta < -700 )
              from accumulate(
                  $meB : MeasurementEvent( )
                over window:length( 6 )
                from entry-point "$${{interface?packageSizeAvgIn}}",
                ExtendedSuddenChangeDetector ( $meB )
                )
          then
            publishIndication( "$${{interface}}" );
          end
        [/DROOLS_TEMPLATE] publishes indications;
      }
    }
};
```

7.5.1.1. Possible Situation Indication

The Situation Indication Processing monitors all network interfaces of routers that are marked as „dosMonitoredRouter" in the knowledge base (Listing 7.4 Line 4). If for any of the monitored interfaces a significant drop in the measured average package size is detected this is considered as an indication for a DoS attack and a Possible Situation Indication Event is raised for the network interface (Listing 7.4 Line 13).

In order to detect such a drop, the processing system considers six consecutive measurement events. As the measurements are generated every 10 seconds, this resembles a time frame of 60 seconds. The used „ExtendedSuddenChangeDetector" splits this time frame into the first and second half, calculates for each half the average value and then the delta between these two values. If this delta is large enough, the indication event is raised.

7.5.1.2. Focused Situation Processing Initialization

The raised Possible Situation Indication Events are classified in Phase 2. During this phase, first the Pre-Classification mechanism is used to filter out duplicate indications. For the current scenario newly raised indications within 600 seconds (Listing 7.5) after the first indication was raised for the same network interface, are ignored assuming that no new DoS attack needs to be detected that starts within less than 10 minutes after the first one was reported.

Listing 7.5: Focused Situation Processing Initialization for the DoS detection

```
1
2  FocusedProcessingInitialization {
3    duplicationThreshold 600s;
4
5    potentialLockedArea $$indicatedNodes;
6    potentialFocusArea $$indicatedNodes;
7
8    initialTimeFrame startsAt [MVEL]$$indicatedTime - 30[/MVEL] withDurationOf 60s;
9    collisionAction preventNew if FA overlap == 100%;
10 }
```

For the classification of the remaining indication events, the Phase 2 processing needs to determine the initial time frame as well as the potential Locked and Focus Areas that result from the indication.

The potential Locked Area needs to represent the identity of the possible situation. For the DoS case, a situation can be identified by the network interface of a certain router that is under attack. As such, the potential Locked Area resulting from a raised Situation Indication Event is the node (network interface) contained in the indication event (Listing 7.5 Line 5).

The potential Focus Area needs to contain all nodes that are relevant for the first iteration of the potentially stated FSP Instance. The first iteration of a started FSP Instance will analyze the indicated interface in further detail in order to verify the potential attack and to determine the magnitude of the attack. As such, the potential Focus Area is also set to the node contained in the indication event (Listing 7.5 Line 6).

The initial time frame is chosen to overlap with the time frame used by the possible situation indication processing by 30 seconds but also contains the following 30 seconds (Listing 7.5 Line 8). Thus the first iteration of the new FSP Instance can verify that the drop of the average package size continues after the initial drop was detected by the Possible Situation Indication Processing.

With regard to the classification of potential collisions, only the case of a complete collision with the Focus Area of a running FSP Instance needs to be considered. The only other possible case for this scenario is a complete Locked Area collision where the handling is defined by the processing model.

If the potential Locked Area of a raised indication is part of the Focus Area of an already active FSP Instance, the indication is considered related to the DoS attack traced by this FSP Instance as its Focus Area is adapted by the Focused Situation Processing to represent the path of the DoS traffic through the network.

7.5.1.3. Focused Situation Processing

The Focused Situation Processing for the DoS Scenario implements two steps:

Step 1: *Verify the indicated possible situation:*

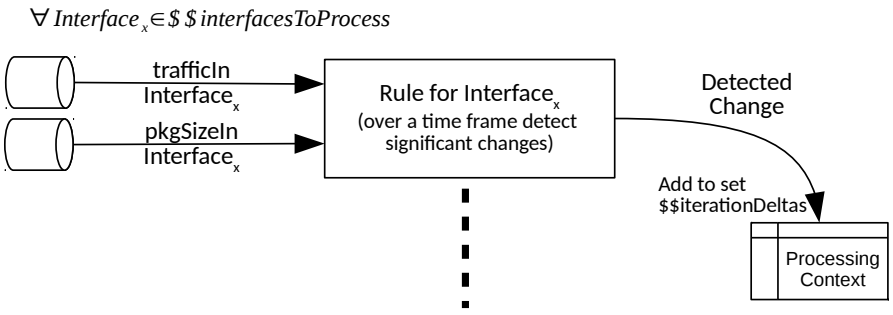


Figure 7.5.1.: Illustration of the iteration stream processing implemented by the Focused Situation Processing for the DoS scenario. The defined Stream Processing Builder (Listing 7.6) generates a rule for every network interface that needs to be analyzed. The rule is triggered if a significant delta in the average packet size *and* package count measurements of the current interface is detected. In this case the rule adds the calculated deltas for the given interface to the set *$$iterationDeltas* in the Focused Situation Processing Context.

The first iteration verifies the indicated possible DoS Attack by checking that not only the average package size dropped significantly but *also* the package count increased. Further both the package size and the package count deltas are determined and stored in the processing context so they can be used for the tracking of the DoS traffic.

Step 2: *Trace the traffic of the DoS Attack trough the monitored network:*
The following iterations implement the tracing by hopping from one router to the next with each new iteration of the FSP Instance. This process continues until the traffic leaves the network or can not be traced anymore as it becomes indistinguishable from the normal traffic.

For both steps, the Iteration Stream Processing Builder (Listing 7.6) generates a Stream Processing Topology that determines the package size and package count differences for each network interface contained in the set *$$interfacesToProcess* and writes them to the FSP Context by adding the delta values to the set *$$iterationDeltas* (Figure 7.5.1).

The set *$$interfacesToProcess* is the subset of the current iterations Focus Area where all network interfaces have been removed which are already verified as being part of the attack's path trough the network. The Focus Area contains those verified interfaces in order to mark the path through the network so that other Possible Situation Indications can be linked to an already investigated DoS trace.

The different handling of the two steps, possible situation verification and the later tracing, is implemented in the Post-Processing function that is executed for each iteration after the stream processing finished. The structure of the contained functionality is as follows (illustrated in Figure 7.5.2):

Listing 7.6: Iteration Stream Processing Builder of the Focused Situation Processing Template for the DoS detection and tracing

```
IterationStreamProcessingBuilder {
      foreach $$interfacesToProcess as $$interface {
        rule [DROOLS_TEMPLATE]
        when
          // package count increased
          $a : SuddenChange( consideredEvents == 6  , delta > 300 )
            from accumulate(
                $meA : MeasurementEvent( )
              over window:length( 6 )
              from entry-point "$${{interface?trafficIn}}",
              ExtendedSuddenChangeDetector ( $meA )
              )
            // package size dropped
            $b : SuddenChange( consideredEvents == 6 , delta < -300 )
            from accumulate(
                $meB : MeasurementEvent( )
              over window:length( 6 )
              from entry-point "$${{interface?packageSizeAvgIn}}",
              ExtendedSuddenChangeDetector ( $meB )
              )
              then
            CONTEXT.addToSet("$$iterationDeltas",new InterfaceDelta("$${{interface
                }}",$a.getDelta(),$b.getDelta()));
              end
        [/DROOLS_TEMPLATE] publishes no stream manipulates context;
      }
    };
```

For the first iteration (Step 1: Verification): *(Listing 7.7 Lines 2ff)*

If the stream processing executed for the first iteration was able to detect a significant enough change in the package count and package size measurements of the indicated interface, the determined differences are stored in the processing context as *$$deltaCount* and *$$deltaSize* so that the following iterations can use these values to find similar traffic deltas on other interfaces. Further the interface is added to the set of *$$verifiedInterfaces* which holds all interfaces that are part of the already verified traffic path through the network. Moreover, the interface are added to the set *$$iterationVerifiedInterfaces* which holds all interfaces newly verified by the *current* iteration so that they can be used as the starting point for finding the interfaces to look at during the next iteration.

If the iteration stream processing was not able to find a large enough delta on the indicated interface, the set *$$iterationVerifiedInterfaces* will remain empty which triggers the termination of the FSP Instance as a False Situation as the indicated possible DoS attack could not be verified.

For all following iterations (Step 2: Tracing): *(Listing 7.7 Lines 13ff)*

The traffic is traced through the network in several iterations. In every iteration, the post processing function looks at all interfaces from the current Focus Area that where identified by the current iterations stream processing as possibly affected by

the investigated DoS attack (All interfaces part of the set *$$iterationDeltas*). From the set of those interfaces, the Post-Processing selects as many nodes as needed to get the total amount of traffic change caused by the DoS attack. The selection starts with the interface that shows the highest measurement differences as this interface is considered the most likely origin of the DoS traffic (Listing 7.7 Lines 33ff).
If enough interfaces are found to account for the overall traffic delta, all the selected interfaces together are then considered as the source of the DoS attack and added to the set of *$$verifiedInterfaces*. Further the selected interfaces are assigned to the set *$$iterationVerifiedInterfaces* so that they are used to generate the Focus Area for the next iteration.
If however it is not possible to find enough interfaces with delta values above the threshold to reach the total DoS traffic amount, the path is considered as lost and the flag *$$pathLost* is set which later triggers the matching termination rule (Listing 7.7 Lines 47).

After the selection of the interfaces is complete, the Focus Area for the next iteration is determined. The next Focus Area has to contain for each newly verified interface (from the set *$$iterationVerifiedInterfaces*) all interfaces of the router which is linked to this verified interface.
Furthermore the set of verified interfaces *$$verifiedInterfaces* is added to the Focus Area in order to mark the so far determined traffic path so that other possible situation indications can be correlated with this FSP Instance. Based on the resulting Focus Area, the processing continues with the next iteration.

Termination of the Trace

If the Focus Area of an iteration only contains already verified interfaces (*$$focusArea* = *$$verifiedInterfaces*) and no new interfaces which could be the source of the DoS attack, the Stream Processing Builder generates an empty rule set. Due to the empty rule set, no stream processing happens during this iteration and the set *$$iterationDeltas* remains empty. In this case, the Post-Processing function only determines if the trace of the DoS traffic is complete (traffic was traced to Edge Routers) or if the trace failed as the trail of the DoS traffic was lost during the last iteration. Based on the results, the Post-Processing function sets an appropriate flag (*$$traceFinished* or *$$trailLost*) which later triggers a termination rule matching the determined outcome. If the trace was completed, the post-processing function determines the origin routers which are reported as part of the final processing result, stating the origin of the DoS attack.

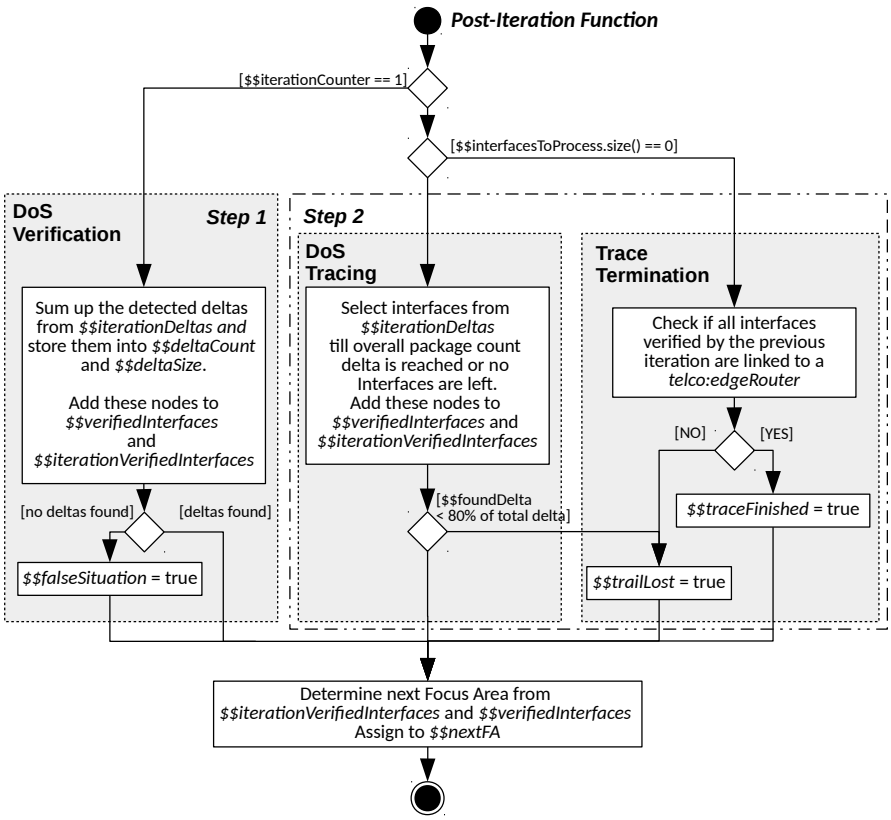


Figure 7.5.2.: Structure of the Focused Situation Processing Post-Iteration function for the DoS tracing.

Listing 7.7: Post Iteration Function of the Focused Situation Processing template for the DoS scenario

```
postIteration [MVEL]
  if( $$iterationCounter == 1 ) {
    // in the first iteration sum up the deltas of the attacked interfaces
      foreach( i : $$iterationDeltas){
        $$deltaCount   += i.getDeltaTraffic();
        $$deltaSize    += i.getDeltaSize();
        $$iterationVerifiedInterfaces.add(i.getInterface());
        $$verifiedInterfaces.add(i.getInterface());
      }
      if($$iterationVerifiedInterfaces.size()==0){
        $$falseSituation = true;
      }
    }else{
      if ($$interfacesToProcess.size() == 0 ) {
        // if there where no interfaces to process,
        // check if trace is complete or if we lost the path
        allBorderRouters = true;

        foreach( i : $$lastIterationVerifiedInterfaces){
          res = CONTEXT.querySet(" SELECT DISTINCT ?VALUE WHERE { " + i + " telco:
              hasLink ?LNK . ?srcInterface telco:hasLink ?LNK . ?VALUE telco:
              hasInterface ?srcInterface . ?VALUE rdf:type telco:edgeRouter }");
          if(res.size() == 0){
              allBorderRouters = false;
          }else{ // res contains the Router where the attack is comming from
            $$originRouters.addAll(res);
          }
        }

        if( !allBorderRouters ) {
          $$trailLost = true;
        }else{
          $$pathComplete = true;
        }
      }else{ // otherwise continue with the trace
        $$lastIterationVerifiedInterfaces.clear();
        $$lastIterationVerifiedInterfaces.addAll($$iterationVerifiedInterfaces);
        $$iterationVerifiedInterfaces.clear();

        trafficSum = 0;
        java.util.Collections.sort($$iterationDeltas);
        foreach( i : $$iterationDeltas){
          if(trafficSum < $$deltaCount){
              $$iterationVerifiedInterfaces.add(i.getInterface());
              $$verifiedInterfaces.add(i.getInterface());
              trafficSum += i.getDeltaTraffic();
        }}
        // if less then 80% of the package count was found on the considered
            interfaces the trace is stopped as the path can't be traced anymore
        if(trafficSum < ($$deltaCount * 0.8) ){
          $trailLost = true;
          $$message = "Path was lost during iteration " + $$iterationCounter + "
              as less then 80% of the traffic could be found";
        }
      }
    }

    // build the next Focus Area manually so there is more control
    $$nextFA = new java.util.HashSet();
    // query all interfaces of routers reachable from the interfaces verified in
        the current iteration
    foreach ( n : $$iterationVerifiedInterfaces ){
      $$nextFA.addAll(CONTEXT.querySet(" SELECT DISTINCT ?VALUE WHERE { " + n + "
          telco:hasLink ?LNK . ?srcInterface telco:hasLink ?LNK . ?srcNode telco:
          hasInterface ?srcInterface . ?srcNode telco:hasInterface ?VALUE }"));
    }
    // also add all verified interfaces to mark our path
    $$nextFA.addAll($$verifiedInterfaces);
};
```

Listing 7.8: Interim result publication, termination rules and statements for preparing the next iteration of the Focused Situation Processing template for the DoS scenario

```
// publish our current tracking state if the processing is not yet finished
publish interim result $$verifiedInterfaces when [MVEL] !$$pathComplete && !
    $$trailLost [/MVEL];

// terminate if no DoS could be verified
terminate if [MVEL] $$falseSituation [/MVEL]
  with result FalseSituation keep area registration if [MVEL]false[/MVEL];

// terminate if we traced the path
terminate if [MVEL] $$pathComplete [/MVEL]
  with result $$verifiedInterfaces, $$originRouters keep area registration if [
      MVEL]true[/MVEL];

// terminate if we can't follow the path any further
terminate if [MVEL] $$trailLost [/MVEL]
  with result $$message, $$verifiedInterfaces keep area registration if [MVEL]true
      [/MVEL];

  // For DoS tracing the Time Frame is fixed so we can track the path
nextIterationTimeFrame startsAt $$startTime withDurationOf 60s ;

// lockedArea is fixed to the nodes that are under attack
nextLockedArea $$lockedArea;

// focusArea was determined in a custom way during pre-Processing
nextFocusArea $$nextFA;

// as the Locked Area never moves, no merge possible
mergeFunction [MVEL][/MVEL];
```

7.5.2. DoS Test Data Simulation

The simulation is based on the telecommunications network shown as part of the scenario discussions in Subsection 2.1.2.1 (Figure 2.1.3). The network consists of two routers which connect a data center to the network of a telecommunications provider (Router 1 and 13), several routers representing the provider's network and three routers connecting the provider's network with other providers (Routers 10, 11 and 15). Furthermore, in the corresponding background knowledge, Routers 1 and 13 are marked to be monitored for possible DoS Attacks by the property *telco:dosMonitoredRouter*. Routers 10,11 and 15 in turn are marked as *telco:edgeRouter* to mark their role as interconnecting routers with the networks of other providers.

For the following test case (Case 5), a DoS attack was simulated based on the discussed network (Figure 7.6.1). The attack is simulated by a burst of very small network packages against one of the border routers of a data center (Router 13). In the simulation, the attack starts at 1438293710 (100s after the simulation starts) and originates from the Router 10. The DoS traffic is routed through the simulated network via the Routers 9,8,6 and 5 to reach Router 13.

In order to simulate normal network traffic, in which the DoS attack needs to be detected and traced, several other nodes in the network generate network traffic by sending packages to other Routers which automatically reply.

The simulation provides the measurement data as a number of CSV files which are loaded by the processing systems prototype. Further the simulation generates the background knowledge base contents based on the simulated network topology as a Turtle file that is also loaded by the prototype.

7.6. Case 5: DoS Tracing

This section discusses the test of the Scenario Processing Template developed for the detection and tracing of a DoS attack. The test is based on a simulated data set which contains a simulated DoS attack against Router 13 originating from Router 10 as discussed in the previous section.

The following subsections discuss the execution of the processing template based on the developed prototype. The discussions are structured along the processing phased defined by the processing model. Furthermore the discussions focus on aspects specific to this application domain and scenario as the general execution of the processing model is already discussed in detail for the previous test cases.

#	Indicated Time	Indicated Nodes	Initial TimeFrame	Potential LockedArea	P2 Classification	Resulting Focused Situation Processing Instance
1	1438293720	tests:interface_13_5	1438293690-1438293751	tests:interface_13_5	New Possible Situation	FP__DosTracing__d20f923f-8c85-4b97-b55f-1067daeb6dcd
2	1438293730	tests:interface_13_5	-	-	Duplicate Indication	-
3	1438293740	tests:interface_13_5	-	-	Duplicate Indication	-

Table 7.6.1.: Case 5: Generated Possible Situation Indication Events

7.6.1. Phase 0: Possible Situation Indication Processing Initialization

The Phase 0 processing used the specified indication nodes query to select two interfaces to monitor for potential DoS Attacks, one interface of each of the Routers 1 and 13. For each interface a single stream processing rule is generated. The resulting structure of the Stream Processing Topology is similar in structure to the Indication Stream Processing topology of the Cloud Tracking Scenario (Figure 7.4.1). Due to the similarity, the generation process and resulting topology is not discussed in further detail.

7.6.2. Phase 1: Possible Situation Indication Processing

The generated stream processing topology was instantiated by the processing system. The following stream processing resulted in the generation of three Possible Situation Indication Events starting with the first one at time 1438293720 which is 10 seconds after the simulated attack started for the interface *tests:interface_13_5* of Router 13 as shown in Table 7.6.1. The following two events were the result of the used sliding window in which the drop in average package size continues to be evident thus causing additional indications.

7.6.3. Phase 2: Focused Situation Processing Initialization

The first Possible Situation Indication Event was correctly classified as concerning a new possible situation and the creation of a new Focused Situation Processing Instance (Refereed to as Instance #1) was triggered (Table 7.6.1 Line 1). The potential Focus Area and Locked Area was set to the interface contained in the indication and the initial time frame started 30s before the indicated time and ended 30s after, thus allowing the created FSP Instance to verify the DoS attack based on the measurement data from the indicated interface.

The following two events (Lines 2 and 3) were raised for the same interface shortly after the initial indication event and were therefore classified as duplicate indications in accordance with the specified time threshold of 600s (Listing 3 Line 3).

7.6.4. Phase 3: Focused Situation Processing

During the first iteration, the FSP Instance successfully verified the possible situation as shown by the generated Interim Result Event (Table 7.6.3, Event #1) which confirms the

#	Time Frame	Requesting Focused Situation Processing Instance	Requested Locked Area	Requested Focus Area	Registration Granted?
1	1438293690-1438293751	FP__DosTracing__d20f923f-8c85-4b97-b55f-1067daeb6dcd	tests:interface_13_5;	tests:interface_13_5;	Yes
2	1438293690-1438293751	FP__DosTracing__d20f923f-8c85-4b97-b55f-1067daeb6dcd	tests:interface_13_5;	tests:interface_5_13;tests:interface_5_17;tests:interface_5_2;tests:interface_5_6;tests:interface_13_5;	Yes
3	1438293690-1438293751	FP__DosTracing__d20f923f-8c85-4b97-b55f-1067daeb6dcd	tests:interface_13_5;	tests:interface_6_8;tests:interface_5_13;tests:interface_5_17;tests:interface_6_15;tests:interface_5_2;tests:interface_6_3;tests:interface_5_6;tests:interface_6_5;tests:interface_6_7;tests:interface_13_5;	Yes
4	1438293690-1438293751	FP__DosTracing__d20f923f-8c85-4b97-b55f-1067daeb6dcd	tests:interface_13_5;	tests:interface_6_8;tests:interface_8_6;tests:interface_8_9;tests:interface_8_14;tests:interface_6_15;tests:interface_6_3;tests:interface_6_5;tests:interface_5_6;tests:interface_8_4;tests:interface_6_7;tests:interface	Yes
5	1438293659 - 1438293720	FP__DosTracing__d20f923f-8c85-4b97-b55f-1067daeb6dcd	tests:interface_13_5;	tests:interface_8_6;tests:interface_6_8;tests:interface_8_9;tests:interface_9_8;tests:interface_8_14;tests:interface_9_16;tests:interface_9_10;tests:interface_5_6;tests:interface_8_4;tests:interface_9_4;tests:interfac	Yes
6	1438293690-1438293751	FP__DosTracing__d20f923f-8c85-4b97-b55f-1067daeb6dcd	tests:interface_13_5;	tests:interface_10_9;tests:interface_6_8;tests:interface_9_8;tests:interface_8_9;tests:interface_9_16;tests:interface_9_10;tests:interface_5_6;tests:interface_9_4;tests:interface_13_5;	Yes
7	1438293690-1438293751	FP__DosTracing__d20f923f-8c85-4b97-b55f-1067daeb6dcd	tests:interface_13_5;	tests:interface_6_8;tests:interface_8_9;tests:interface_9_10;tests:interface_5_6;tests:interface_13_5;	Yes

Table 7.6.2.: Case 5: Area Registration Requests generated as part of the tracking process by the created FSP Instance.

interface *tests:interface_13_5* of Router 13 as being an interface of the DoS attacks traffic path.

Based on the verification, the FSP Instance started with the tracing the DoS attack trough the network in its second iteration. The trace was completed after five iterations as shown by the Area Registration Requests (Table 7.6.2 Lines 2 to 6). For the first tracing step (Iteration 2), the FSP Instance focused on all interfaces of the Router 5 (*tests:interface_5_**) and the already verified interface *tests:interface_13_5* as shown by the Area Registration Request #2. This first tracing step added the network interface *tests:interface_5_6* to the list of verified interfaces as shown by the generated Interim Result Event (Table 7.6.3, Event #2). For the third iteration the same process was repeated for all network interfaces of Router 6 where the processing resulted in the addition of the interface *tests:interface_6_8* (Table 7.6.3, Event #3).

This process was continued until the FSP Instance reached Router 10 where no additional interface was added to the list of verified interfaces anymore, which completed the trace. The FSP Instance concluded its processing with one additional iteration for which the registered Focus Area represented the complete traced path of the DoS traffic trough the network (Area Registration Request #7). The FSP then terminated with a Final Result Event which correctly stated the traced traffic path in *$$verifiedInterfaces* as well as the routers which where the origin of the attack in *$$originRouters* (Table 7.6.3, Event #6). Figure 7.6.1 illustrates the resulting path by showing the Focus Area from the final Area Registration in blue rectangles.

In total the FSP Instance needed 7 iterations to verify the situation and to trace it back to its origin. The first iteration for the verification, the iterations 2 to 6 to trace the traffic to its origin and the final iteration to update the Area to correctly represent the traced path.

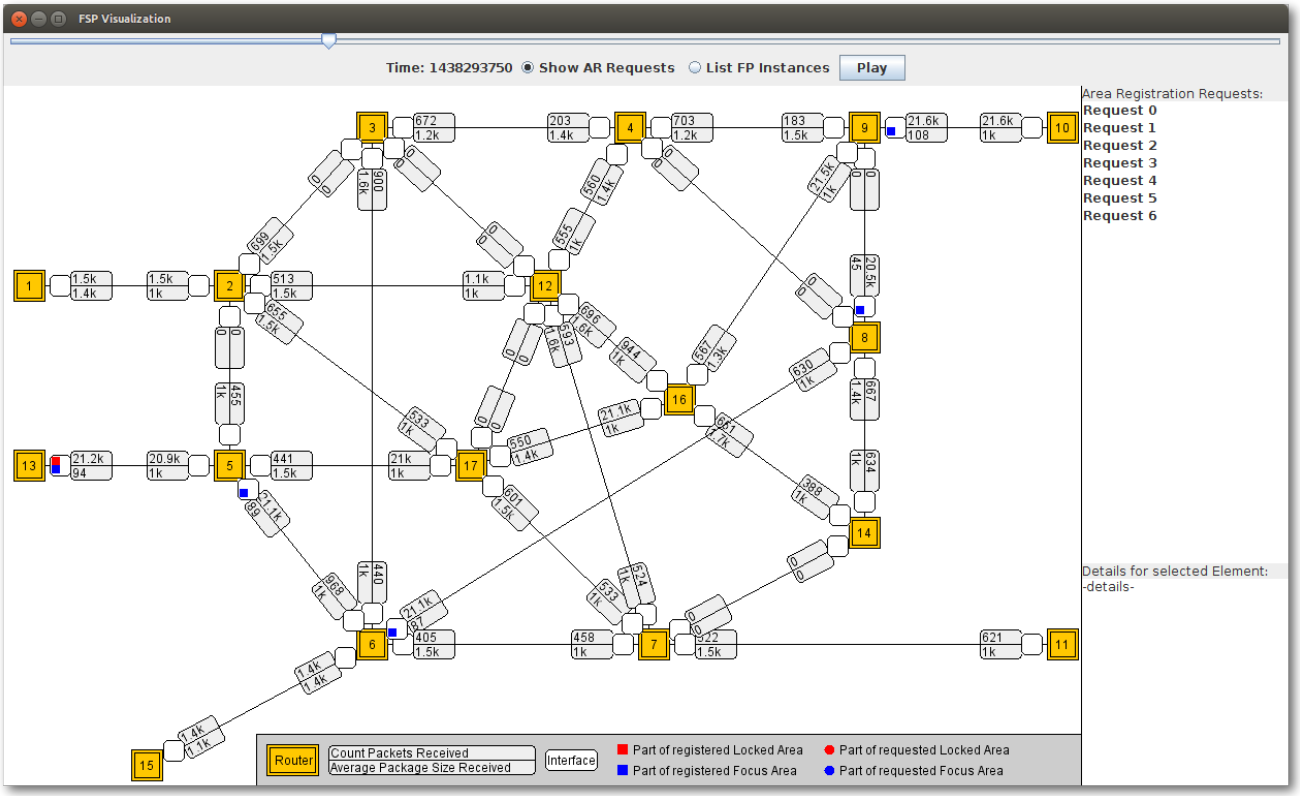


Figure 7.6.1.: Case 5: Visualization of the simulated telecommunications network with an active DoS attack against Router 13. The red rectangle shows the Locked Area of the FSP Instance which traced the DoS attack to its origin as shown by the blue Rectangles which represent the final Focus Area acquired by the FSP Instance once it had completed the trace.

#	Event Type	Time	Contents
1	InterimResultEvent	1438293751	$$verifiedInterfaces = [tests:interface_13_5]
2	InterimResultEvent	1438293751	$$verifiedInterfaces = [tests:interface_13_5;tests:interface_5_6]
3	InterimResultEvent	1438293751	$$verifiedInterfaces = [tests:interface_13_5;tests:interface_5_6;tests:interface_6_8]
4	InterimResultEvent	1438293751	$$verifiedInterfaces = [tests:interface_13_5;tests:interface_5_6;tests:interface_6_8;tests:interface_8_9]
5	InterimResultEvent	1438293751	$$verifiedInterfaces = [tests:interface_13_5;tests:interface_5_6;tests:interface_6_8;tests:interface_8_9;tests:interface_9_10]
6	FinalResultEvent	1438293751	$$verifiedInterfaces = [tests:interface_13_5;tests:interface_5_6;tests:interface_6_8;tests:interface_8_9;tests:interface_9_10] $$originRouters = [[tests:node_10]]

Table 7.6.3.: Case 5: Result Events generated by the FSP Instance reporting the interfaces along the DoS traffic path.

7.6.5. Conclusions from Case 5

The processing system correctly identified the DoS attack and traced the traffic through the network to its origin as illustrated by Figure 7.6.1. The test case thereby demonstrated the use of the processing model based on a processing template defined in SPTL for a different application domain than the so far used Smart Grid domain.

7.7. Synchronization Required by the Processing Model

The processing model is aimed at the parallel execution of the stream processing tasks in order to achieve scalability over potentially large numbers of event streams. This section discusses the amount of synchronization required by the processing model as this effectively limits the amount of parallelization possible (Also see the discussion of the answer to RQ1.1 on page 159).

The indication stream processing in Phase 1 needs no coordination within this phase and can from the perspective of the processing model be executed in parallel for each separate event stream. However, the resulting possible situation indications need to be synchronized with each other and with already active FSP Instances during the Phase 2 processing. Moreover, in Phase 3, the active FSP Instances need to be synchronized after the completion of each iteration. The synchronization for both processing phases is realized through the Area Registration mechanism as discussed in Section 4.5.10 for Phase 2 and Section 4.6.6 for Phase 3. these synchronization requirements effectively limit the number of parallel (potential) situations that can be handled by the system (See Limitation L3). The synchronization requirements have however no limiting impact on the number of event streams that need to be monitored for potential situations.

While multiple FSP Instances can be executed in parallel based on the discussed synchronization mechanisms, the processing of any single situation can not be distributed onto multiple machines as the processing model does not define how such a distributed processing of a single situation has to be synchronized between multiple machines (See Limitation L2).

7.8. Limitations of the Processing Model and Language

The evaluation of the processing model demonstrated, that it behaves as expected. However, the processing model itself has certain limitations which are discussed in the following paragraphs as a foundation for the later discussion of possible future work in Section 8.2:

L1: Suitable Possible Situation Indication Functions Required
The processing model relies on the existence of „suitable" indication functions that can be used to identify potential situations. Such an indication function is considered suitable if it can detect potential situations from a fixed set of event streams

with an acceptable rate of false positives & false negatives for the given scenario. Furthermore, the indicator must only require a limited amount of processing time to allow the possible situation indication processing to keep up with the rate of inbound events that need to be screened for possible situations.

L2: Limited Size of *Single* Situations

The processing model uses a situation-specific processing context for each investigated (possible) situation. This context can be used freely by the scenario-specific processing logic. This however limits the processing of a single situation to the scope of a single machine as no synchronization mechanism for accessing the processing context in a distributed context is specified by the model. Thus, the maximum size of a situation is effectively limited by the capacity of the used machines.

L3: Limited Number of *Parallel* (Possible) Situations

The processing model uses its Area Registration mechanism for the coordination between multiple parallel (possible) situations. While the processing of the (possible) situations can be executed in parallel on multiple machines, the Area Registration mechanism acts as the *central* synchronization mechanism between them. Thus, the set of active Area Registrations needs to be managed by a single machine which effectively limits the number situations that can be processed in parallel as each (possible) situation requires at least one Area Registration.

L4: Only Static Background Knowledge Supported

The processing model does not define how changes in the background knowledge can be incorporated into the ongoing processing, for example into an ongoing tracing process of a cloud. Thus, the processing model is currently only suitable for static background knowledge.

L5: Limited Continuity of Situation Identities Under Certain Circumstances

The processing model is providing a situation-*specific* processing which implies situation-specific processing results. Thus, every processed situation has its own identity while being processed (represented by the responsible Focused Situation Processing Instance). For the two following cases however, the identity of a situation can change, leading to the assignment of a new identity:

Merging of Situations: *(Discussed in Test Case 3, Section C.2)*

When two situations A, B merge into one situation, the processing system gives no guarantees which of the two situation identities is used for the situation after the merge. Thus, the selection of the identity is dependent on the actual implementation of the processing system and the timing during execution.

Collision of Situations: *(Discussed in Test Case 4, Section C.3)*

As mentioned in the previous paragraph, when two situations A, B collide with

each other, the processing model handles this collision by merging the two situations into one, e.g. *A*. This for example happens if two clouds move close enough together so that the processing system can no longer separate them, while the actual clouds do not merge into one. When these two actual situations at a later point in time become distinguishable again by the processing system, the processing system will only follow one of these situations with the identity *A*. The other situation is detected as a *new* situation with a *new identity C*. Thus, after the merge and separation of the *two* actual situations, *three* situations have been reported by the processing system. Further the processing model gives no guarantees, which one of the two actual situations retains the identity and which one becomes a new identity.

L6: Limited to one Situation Type per Template with no Interaction between Different Templates

The processing model does not define any interactions between different situation types defined in different Scenario Processing Templates. For example if one template defines the tracking of clouds and another describes the tracking of storm fronts, no processing model conform interaction between the two templates is possible.

Furthermore, the Area Registration mechanism defined by the processing model does not define a mechanism to register different kinds of Area Registrations in order to separate registrations based on the type of situations they represent. This effectively prevents the implementation of two different types of situations like the cloud and storm front tracking in a single template as the Area Registrations of the different situation kinds can not be distinguished.

7.9. Preconditions for the Application of the Processing Model

Based on the previous discussion of the limitations of the processing model, this subsection discusses the preconditions that need to be fulfilled by a scenario to allow the application of the designed processing model:

1. Possibility to subdivide the the processing task of a given scenario into the processing phases of the processing model by defining a matching *indication function* as well as *Locked Area and Focus Area selection functions* together with a *collision handling function*.

2. *Suitable background knowledge* for the given scenario must exist and *must be linked to the available event streams* in oder to allow the generation of stream processing topologies for the indication processing as well as the situation specific processing. Further the scenario must be based on static background knowledge only (See Limitation L4).

3. For the Locked Area and Focus Area mechanism, a *scenario specific neighborhood criteria* between nodes in the knowledge base is needed to allow to determine which nodes are part of a new Locked Area or Focus Area like for example a geographical relation as demonstrated by the Cloud Tracking Scenario or a topological relation as demonstrated by the DoS Scenario.

4. Any situation that is to be processed needs to have a unique identity based on the defined Locked Area from the Area Registration mechanism to allow the processing model to process multiple situations separately for a given scenario. Further situations need to stay separable over time based or a loss of the situation identities needs to be acceptable by the scenario as discussed for Limitation L5.

5. The scenario must only require one kind of situation as the Area Registration mechanism does not allow to distinguish between different kinds of situations as discussed for Limitation L6.

6. The processing load of any single situation that can be expected for a scenario must not exceed the capabilities of a single machine of a used processing system as the situation specific processing of any one situation needs to take place on a single machine (See Limitation L2).

7.10. Conclusions

The realization of the Cloud Tracking Scenario and the DoS Detection and Tracing Scenario successfully demonstrated the use of the SPTL for the definition of situation-aware adaptive processing tasks for two distinct scenarios from two separate application domains. Based on the realized scenarios, several tests where successfully conducted:

The first four[6] cloud tracking based test cases demonstrated the use of the processing model as well as special functionality like the merging of situations. The final fifth test case demonstrated the use of the processing model for a different application domain thus demonstrating the domain independence of the SPTL and the processing model. Further it demonstrated the use of non geographical background knowledge and static Focused Situation Processing Time Frames.

Aside from the verification of the processing model and SPTL functionalities, Test Case 4 (Appendix C.3) also pointed out the limitation of the processing model towards the tracing of multiple parallel situations with temporarily ambiguous situation identities.

[6]The Test Cases 2 to 5 are documented in Appendix C.

8. Conclusions and Outlook

Contents

8.1. Summary and Resulting Conclusions

This work defines a situation-aware adaptive event stream processing model and scenario specification language. The processing model and language allow the specification of stream processing tasks which support an automatic scenario-specific adaptation of their processing logic based on detected situations in order to analyze and follow detected situations. The **motivation** for this work lies in the missing support of current state of the art Event Stream Processing (ESP) systems for such a „situation-aware adaptive Event Stream Processing". Scenarios that can benefit from this a processing model are scenarios that require

- the identification of situations of potential interest in a large set of streaming data while
- the verification and analysis of the situation requires an in-depth analysis based on a situation-specific *subset* of the overall streaming data.

One such **scenario** is the detection and tracing of solar energy production drops caused by clouds shading solar panels as they pass by. The scenario requires a stream processing system to handle large amounts of streaming data (energy production measurements from a great number of solar panel installations) to detect a cloud (possible situation). However, the later verification of the detected cloud as well as its tracking only requires a small *situation-specific* subset of the overall streaming data, namely the measurements from the solar panels of the affected area. The scenario thus requires a situation-aware adaptation of its processing setup in order to focus on a detected cloud. Further the cloud will change its position over time resulting in a different set of affected solar panels which again requires the processing system to adapt its processing setup based on changes in the tracked situation.

Aside from this scenario, two additional scenarios from the application area of telecommunication network monitoring were discussed which pose similar challenges.

8.1.1. Gap in the State of the Art

Based on the analysis of the three scenarios, a set of six **characteristic properties** was defined (Section 2.2). Based on these properties, the analysis of the current **State of the Art** has revealed that the *existing classes of event stream processing systems and approaches are not on their own capable of a situation-aware adaptive event stream processing.*

Even though distributed Data Stream Management Systems (DSMS) provide a lot of the functionality that is needed, they lack support for the automatic query adaptations based on a higher level model as they have no support for such a model (Section 3.2.4). Other approaches, targeted at adaptive DSMS optimization, exist like systems that employ statistics based optimizations of their query graphs. Other approaches introduce new operators which allow for an adaptive partitioning or query plan execution. Such approaches allow for a certain degree of adaptiveness of DSMS. Yet, they are focused on the optimization of the already deployed queries to handle fluctuations in the incoming data stream sizes or wrong initial assumptions. These approaches also do not provide a higher level model that allows the targeted scenario-specific adaptation of the queries themselves based on the detection of a possible or changing situation.

8.1.2. Problem Statement

This leads to the **problem** that a lack of support for scenarios that require a situation-aware adaptive event stream processing is given. As a result, for each new scenario, a new processing system needs to be designed, implemented and maintained. Such a processing system is specific for this scenario's required adaptivity. It is therefore the **aim** of this work to ease the development of such situation-aware adaptive processing systems.

This work **approach**es the problem by defining *a situation-aware adaptive stream processing model* together with a matching *scenario definition language* to allow the definition of situation-aware adaptive processing scenarios for a *scenario independent processing system.* The **requirements** for the definition of the model and language are the result of an analysis of three distinct scenarios from two application domains which all require a situation-aware adaptive event stream processing.

8.1.3. Contribution

Based on the approach, this work has two main contributions:

1. The **Processing Model** defines situation-aware adaptive event stream processing in an implementation independent way. It consists of *three main phases.* The *first phase* is aimed at the rapid detection of possible situations in large sets of event

streams. The *second phase* is responsible for the decision whether an indicated possible situation needs to be investigated by a Focused Situation Processing Instance. *Phase three* then constitutes the situation focused analysis performed for a specific situation. Further the model defines a *mechanism to provide identities to detected (possible) situations* by assigning Locked Areas based on an Area Registration mechanism.

2. Based on the model the *Scenario Processing Template* ***Language*** *(SPTL)* was defined which allows the specification of situation-aware adaptive processing tasks in the form of scenario processing templates. These templates are combined with the information available from a background knowledge to configure a processing system. Further, the templates are used together with information on detected (possible) situations to adapt the processing system in a scenario and situation-specific way.

8.1.4. Evaluation

A **prototype** was created which implements the defined processing model and can execute situation-aware adaptive processing scenarios defined in the SPTL. Thus, it implements a parser and interpreter for the SPTL.

Even though the prototype is not created as a distributed scalable system, its architecture outlines how the several components needed for implementing the processing model can be tailored to have a high cohesion of the components with only service or event based interactions with other components as it is required for a distributed version.

For the **evaluation** of the developed model and language, two scenarios have been realized in SPTL as templates and were tested on the processing system prototype. The *realization of the two scenarios* (Cloud Tracking and Denial of Service Tracing) demonstrates the usability of the defined language to express complex scenarios. Further it demonstrates the independence of the SPTL from the application domain as the scenarios originate from *two separate domains* with different requirements[1].

Based on the prototype and realized scenarios, *five test cases* have been performed and the results analyzed. The first test case (Detection and tracking of a single Cloud) successfully demonstrates the main aspects of the processing model as well as the capability of the corresponding scenario template to provide the required information for executing a situation-aware adaptive processing. The following three test cases demonstrate additional functionality of the processing model like the handling of false situations or the merging of focused situation processing tasks.

While the first four cases are based on the Cloud Tracking scenario, the last test case demonstrates the application of the processing model for the telecommunication network

[1] For example with regard to the kind of the needed background knowledge which, for the Cloud Tracking scenario, is of a geographic nature while being of a topological nature for the Denial of Service Tracing scenario.

DoS tracing scenario, and thus for a separate application domain.

The evaluation demonstrates that the language and processing model fulfill the defined requirements by providing an application domain and scenario independent mechanism to define and execute situation-aware processing tasks on a generic processing system.

Even though the scenario realization is still a complex task that requires planning on how the processing should take place like for example, when and how to select new nodes for the processing (Focus Area), the templates only concern scenario-specific aspects. Thus, the scenario developer does not need to tackle technical problems on how to obtain the needed event streams or how to integrate background knowledge into parts of the process. Most importantly, the complete process of detecting potential situations (Phase 1) to their classification (Phase 2) and to verify and analyze possible situations (Phase 3), is provided and only needs to be configured for a given scenario.

8.2. Outlook

Based on the results of this work and the discussed limitations (Section 7.8), further research topics can be identified in the following areas: Adding additional functionality, adding increased usability, further enhancing scalability and providing additional evaluation of the model and language.

The following discussions also offer approaches to address the Limitations L3 to L6 outlined in Section 7.8. The limitations L1 & L2 are however not addressed as they are considered a design aspect of the model.

Additional Functionality:

1. The third test case (Multiple FSP Instances for One Cloud) discusses that the processing model does not define which one of two merging situations retains its identity and which one looses its identity (Limitation L5). A possible approach for avoiding this limitation is to allow a scenario-specific function to determine which of the two merging situations should be merged into the other one.
2. The fourth test case (Temporary Situation Merge) demonstrates the limitation of the processing model in handling temporarily colliding situations (Limitation L5). More generally the current model has only limited capability to handle situations with temporary non-unique situation identities. This limitation could be mitigated for example by adding the capability of retaining a situation's identity based on additional scenario-specific knowledge (e.g. the clouds trajectory and speed to calculate likely further occurrences of the same situation) even if the situation is temporary lost or is indistinguishable from another situation. Further, the capability to reuse a situation's identity after it is detected again could be added to the model.

3. The current processing model does not provide any handling for changes of the background knowledge during run-time (Limitation L4). In order to support changes of the background knowledge, additional research on how to incorporate the changes into active FSP Instances would be needed. One possible approach could be to synchronously change the background knowledge for all active FSP Instances as well as the Phase 2 processing in order to avoid inconsistencies. This step would also need to include the handling of Area Registration updates due to the changed background knowledge and with it also the handling of collisions.
4. As discussed for Limitation L6, the processing model does not support more than one kind of situation per processing template. This limitation is caused by the Area Registration mechanism which does not provide the means to distinguish between two different situation kinds. The limitation could be mitigated by extending the Area Registration mechanism to support multiple types of Locked Areas in order to support more than one kind of situation in one template. This would however require further research towards the handling of collisions between different situation kinds.

Enhancing the Scalability:

With regard to enhancing the scalability of the processing model towards larger numbers of parallel situations (Limitation L3), the processing model could allow some parallelization of the Area Registration Mechanism. Such parallelization could be achieved by partitioning the Area Registration mechanisms co-domain in a scenario-specific way. This would require further research towards the definition of a scenario-specific partitioning scheme.

Increasing Usability:

The usability of the designed specification language may be improved by providing some basic tooling for the definition of Scenario Processing Templates as well as the debugging of their execution, for example an Eclipse based IDE could simplify the realization of new scenarios. Furthermore, providing some abstraction from the used stream processing rule language (Drools) could ease the development of templates and allow the use of other stream processing engines with their own languages.

Extending the Evaluation:

An implementation of the processing model and language suitable for a large scale distributed deployment would allow determining the practical scalability limitations of the chosen Area Registration based synchronization mechanism. Further, the realization of other scenarios from additional domains would further verify the general applicability of the results.

A. Scenario Processing Template Language

This chapter contains the complete specification of the language designed by this work as it has been discussed in Chapter 5. It further specifies additional details regarding the handling of the embedded languages.

A.1. EBNF Representation

⟨*ScenarioProcessingTemplate*⟩ ::= ⟨*TemplatePreamble*⟩
⟨*PossibleSituationIndication*⟩
⟨*FocusedProcessingInitialization*⟩
⟨*FocusedSituationProcessing*⟩

Template Preamble:

⟨*TemplatePreamble*⟩ ::= ⟨*TemplateName*⟩ ⟨*DroolsPrefix*⟩? ⟨*SPARQLPrefix*⟩?

⟨*TemplateName*⟩ ::= 'name' ⟨*STRING*⟩ ';'

⟨*DroolsPrefix*⟩ ::= 'drools prefix' ⟨*STRING*⟩ ';'

⟨*SPARQLPrefix*⟩ ::= 'sparql prefix' ⟨*STRING*⟩ ';'

Possible Situation Indication:

⟨*PossibleSituationIndication*⟩ ::= 'PossibleSituationIndication' '{'
⟨*IndicationNodesQueryFunction*⟩
⟨*IndicationStreamProcesssingBuilder*⟩
'}'

⟨*IndicationNodesQueryFunction*⟩ ::= '$$indicationNodes' ⟨*SPARQL*⟩ ';'

⟨*IndicationStreamProcesssingBuilder*⟩ ::= 'IndicationStreamProcessingBuilder' '{'
⟨*StreamProcessingBuilder*⟩
'}'

Focused Situation Processing Initialization:

⟨*FocusedProcessingInitialization*⟩ ::= 'FocusedProcessingInitialization' '{'
⟨*IndicationPreClassificationThreshold*⟩
⟨*PotentialLockedFocusAreaInitialTimeFrameQueryFunction*⟩
⟨*PartialLockedAreaCollisionActionAssignmentFunction*⟩
⟨*FocusAreaCollisionActionAssignmentFunction*⟩
'}'

⟨*IndicationPreClassificationThreshold*⟩ ::= 'duplicationThreshold' ⟨*TIME_DURATION*⟩ ';'

⟨*PotentialLockedFocusAreaInitialTimeFrameQueryFunction*⟩ ::= ⟨*PotentialLockedArea*⟩
⟨*PotentialFocusArea*⟩
⟨*InitialTimeFrame*⟩

⟨*PotentialLockedArea*⟩ ::= 'potentialLockedArea' (⟨*VAR*⟩ | ⟨*SPARQL*⟩) ';'

⟨*PotentialFocusArea*⟩ ::= 'potentialFocusArea' (⟨*VAR*⟩ | ⟨*SPARQL*⟩) ';'

⟨*InitialTimeFrame*⟩ ::= 'initialTimeFrame' 'startsAt' (⟨*VAR*⟩ | ⟨*MVEL*⟩)
'withDurationOf' (⟨*TIME_DURATION*⟩ | ⟨*MVEL*⟩) ';'

⟨*PartialLockedAreaCollisionActionAssignmentFunction*⟩ ::= (⟨*PartialLAcollisionFunction*⟩
| ⟨*CollisionRules*⟩)

⟨*FocusAreaCollisionActionAssignmentFunction*⟩ ::= (⟨*FaCollisionFunction*⟩
| ⟨*CollisionRules*⟩)

⟨*PartialLAcollisionFunction*⟩ ::= 'partialLACollision' ⟨*MVEL*⟩ ';'

⟨*FaCollisionFunction*⟩ ::= 'FACollision' ⟨*MVEL*⟩ ';'

⟨*CollisionRules*⟩ ::= (⟨*CollisionRule*⟩)*

⟨*CollisionRule*⟩ ::= 'collisionAction' ⟨*CollisionAction*⟩ (',' ⟨*CollisionAction*⟩)*
'if' ⟨*Condition*⟩ ('and' ⟨*Condition*⟩)* ';'

⟨*Condition*⟩ ::= ('LA'|'FA') 'overlap' ('⟨' | '⟩' | '==' | '⟨=' | '⟩=')
⟨*PERCENTAGE*⟩

⟨*CollisionAction*⟩ ::= 'startNew','addToExisting','noAction','preventNew',
'stopActionExecution'

Focused Situation Processing:

⟨*FocusedSituationProcessing*⟩ ::= 'FocusedSituationProcessing' '{'
⟨*FocusedSituationProcessingInitializationFunction*⟩
⟨*PreIterationProcessingFunction*⟩
⟨*IterationStreamProcessingBuilder*⟩
⟨*PostIterationProcessingFunction*⟩
⟨*InterimResultEventGenerationFunction*⟩
⟨*FocusedSituationProcessingTerminationConditionAndTerminationResult*⟩
⟨*IterationLockedAreaFocusAreaTimeFrameQueryFunction*⟩
⟨*FocusedSituationProcessingCollisionHandlingFunction*⟩
'}'

⟨*FocusedSituationProcessingInitializationFunction*⟩ ::= 'contextInitialization' ⟨*MVEL*⟩ ';'

⟨*PreIterationProcessingFunction*⟩ ::= 'preIterationProcessing' ⟨*MVEL*⟩ ';'

⟨*IterationStreamProcessingBuilder*⟩ ::= 'IterationStreamProcessingBuilder' '{'
⟨*StreamProcessingBuilder*⟩ '}'

⟨*PostIterationProcessingFunction*⟩ ::= 'postIterationProcessing' ⟨*MVEL*⟩ ';'

⟨*InterimResultEventGenerationFunction*⟩ ::= ⟨*publishRuleDef*⟩*

⟨*publishRuleDef*⟩ ::= 'publish' 'result' ⟨*vars*⟩ 'when' ⟨*MVEL*⟩ ';'

⟨*FocusedSituationProcessingTerminationConditionAndTerminationResult*⟩ ::= ⟨*TerminationRule*⟩+

```
⟨TerminationRule⟩ ::= 'terminate' 'if' ⟨MVEL⟩
    'with' 'result' ( 'FalseSituation' | ⟨vars⟩ )
    'keep' 'area' 'registration' 'if' ⟨MVEL⟩

⟨IterationLockedAreaFocusAreaTimeFrameQueryFunction⟩ ::= ⟨NextIterationLockedArea⟩
    ⟨NextIterationFocusArea⟩ ⟨NextTimeFrame⟩

⟨NextIterationLockedArea⟩ ::= 'nextFocusArea' ( ⟨VAR⟩ | ⟨SPARQL⟩ ) ';'

⟨NextIterationFocusArea⟩ ::= 'nextLockedArea' ( ⟨VAR⟩ | ⟨SPARQL⟩ ) ';'

⟨NextTimeFrame⟩ ::= 'nextIterationTimeFrame' 'startsAt' ( ⟨VAR⟩ | ⟨MVEL⟩ )
    'withDurationOf' ( ⟨TIME_DURATION⟩ | ⟨MVEL⟩ ) ';'

⟨FocusedSituationProcessingCollisionHandlingFunction⟩ ::= 'mergeFunction' ⟨MVEL⟩
```

Stream Processing Builder:

```
⟨StreamProcessingBuilder⟩ ::= ⟨ProcOperation⟩+

⟨ProcOperation⟩ ::= ⟨BackgroundKnowledgeQuery⟩
    | ⟨ForEach⟩
    | ⟨ForEachGroup⟩
    | ⟨PublishStatement⟩
    | ⟨SetOperation⟩
    | ⟨Conditional⟩

⟨BackgroundKnowledgeQuery⟩ ::= ⟨VAR⟩ ⟨SPARQL⟩ ';'

⟨ForEach⟩ ::= 'foreach' ⟨VAR⟩ 'as' ⟨VAR⟩ '{' ⟨ProcOperation⟩+ '}'

⟨ForEachGroup⟩ ::= 'foreach' ⟨VAR⟩ 'as' ⟨VAR⟩ 'group by' ⟨STRING⟩
    '{' ⟨ProcOperation⟩+ '}'

⟨StreamProcessingRule⟩ ::= 'rule' ⟨DROOLS⟩ 'publishes' (
    'indications'
    | (
    'stream' ( ⟨VAR⟩ '.'  ⟨ID⟩ | ⟨ID⟩ )
    | 'no' 'stream'
    ) ('manipulates' 'context')?
    )  ';'

⟨SetOperation⟩ ::= ⟨VAR⟩ '=' ⟨VAR⟩ ( '+' | '&&' | '-' ) ⟨VAR⟩ ';'

⟨Conditional⟩ ::= 'if' ⟨MVEL⟩ '{' ⟨ProcOperation⟩* '}' ( 'else' '{' ⟨ProcOperation⟩* '}')?
```

General Elements:

```
⟨SPARQL⟩ ::= 'from' 'sparql' ⟨STRING⟩

⟨MVEL⟩ ::= '[MVEL]'.*?'[/MVEL]'

⟨DROOLS⟩ ::= '[DROOLS_TEMPLATE]'.*?'[/DROOLS_TEMPLATE]'

⟨VAR⟩ ::= '$$' [a-zA-Z0-9_]+

⟨vars⟩ ::= ⟨VAR⟩ ( ',' ⟨vars⟩ )?

⟨PERCENTAGE⟩ ::= ('100'|'1'..'9''0'..'9'|'0'..'9')('.'('0'..'9')+)?'%'
```

⟨*TIME_DURATION*⟩ ::= ⟨*INT*⟩ ('h'|'m'|'s')

Comments:

⟨*SL_COMMENT*⟩ ::= '//' .*? '\n'

⟨*ML_COMMENT*⟩ ::= '/*' .+? '*/'

A.2. Java Interfaces Available from MVEL

A.2.1. CollisionTuple

```
package es.schaaf.fsp.model;

public interface CollisionTuple extends Comparable<CollisionTuple> {
  /**
   * Get the collision grade with the Locked Area as absolute value (count of
   * overlapping nodes)
   */
  public int getGradeLa();

  /**
   * Get the collision grade with the Focus Area as absolute value (count of
   * overlapping nodes)
   */
  public int getGradeFa();

  /**
   * Get the colliding Area Registration
   */
  public AreaRegistration getAreaRegistration();
}
```

A.2.1.1. Enum CollisionAction

```
package es.schaaf.fsp.model;

public enum CollisionAction {
  AddToExisting, NoAction;
}
```

A.2.2. AreaRegistration

```
package es.schaaf.fsp.model;

public interface AreaRegistration {
  /**
   * Get the {@link FocusedProcessingIdentifier} of the FSP Instance that owns
   * the area registration
   */
  public FocusedProcessingIdentifier getFpId();

  /**
   * Get {@link TimeFrame} for the area registration
   */
  public TimeFrame getTimeFrame();

  /**
   * Get the Locked Area for the registration
   */
  public Area getLockedArea();

  /**
```

```
    * Get the Focus Area for the registration
    */
   public Area getFocusArea();
}
```

A.2.3. TimeFrame

```
package es.schaaf.fsp.model;

public interface TimeFrame {
   /**
    * get start end time in seconds since 1970
    */
   long getStartTime();

   /**
    * get the end time in seconds since 1970
    */
   long getEndTime();

   /**
    * Returns the number of seconds the time frames overlap, 0 if they don't
    * overlap
    */
   int overlaps(TimeFrame tf);

   /**
    * Returns true if the given time frame is included in this
    */
   boolean includes(TimeFrame tf);
}
```

A.2.4. Area

```
package es.schaaf.fsp.model;

import java.util.Set;

public interface Area extends Set<String> {
   /**
    * Returns the number of nodes the given area overlaps with this area
    */
   int overlaps(Area la);
}
```

A.2.5. Event

```
package es.schaaf.fsp.model.events;

public interface Event {
   /**
    * get the time of the event in seconds since 1970
    * */
   long getTime();
}
```

B. Implemented Processing Specifications

B.1. Cloud Tracking Scenario

The Listing B.1 shows the complete Scenario Processing Template used for realizing the cloud tracking as discussed in Section 7.3 and used for the tests in Section 7.4 and Appendix C.

```
Name "Cloud Tracking";

SPARQL prefix "
PREFIX tests:<smartGrid://smartgrid/tests#>
";

// load scenario specific accumulation function into Drools
DROOLS prefix [DROOLS_TEMPLATE]
import accumulate es.schaaf.cloudTracking.SuddenChangeDetector
    SuddenChangeDetector;
[/DROOLS_TEMPLATE];


PossibleSituationIndication {
  $$indicationNodes from sparql "?VALUE rdf:type smartgrid:device. ?VALUE
      smartgrid:providesMeasurement ?point. ?point rdf:type smartgrid:
      PVPowerProduced.";

  IndicationStreamProcessingBuilder{
          foreach $$indicationNodes as $$pv {
                rule [DROOLS_TEMPLATE]
                    when
                        Number( $delta : doubleValue )
                        from accumulate(
                            MeasurementEvent( $val:value )
                            over window:length( 2 )
                            from entry-point "$${{pv?PVPowerProduced}}",
                            SuddenChangeDetector( $val )
                        ) eval($delta > 50)
                    then
                        publishIndication( "$${{pv}}" );
                    end
                [/DROOLS_TEMPLATE] publishes indications;
            }
        }
};


FocusedProcessingInitialization {
    duplicationThreshold 300s;

    potentialLockedArea $$indicatedNodes;

    potentialFocusArea from sparql "
        $${{indicatedNodes}} smartgrid:hasLocation ?LOC1.
        ?VALUE smartgrid:hasLocation ?LOC2.
        ?LOC1  smartgrid:hasLat ?LAT1.
        ?LOC2  smartgrid:hasLat ?LAT2.
        ?LOC1  smartgrid:hasLon ?LON1.
        ?LOC2  smartgrid:hasLon ?LON2.
```

```
        FILTER ( ?LAT1+0.0041 > ?LAT2 ) .
        FILTER ( ?LAT1-0.0041 < ?LAT2 ) .
        FILTER ( ?LON1+0.0041 > ?LON2 ) .
        FILTER ( ?LON1-0.0041 < ?LON2 ) . ";

    initialTimeFrame startsAt $$indicatedTime withDurationOf 300s ;

    collisionAction preventNew if FA overlap == 100% ;
};

FocusedSituationProcessing {

    contextInitialization [MVEL]
        $$oldLAofMergedFPI = null;
    [/MVEL];

    preIteration [MVEL]
        $$positiveNodes = new java.util.HashSet();

        $$nodesToConsider = $$focusArea;

        if($$oldLAofMergedFPI!=null){
            // remove the nodes that where already verified
            java.util.Iterator iter = $$oldLAofMergedFPI.iterator();
            while (iter.hasNext()) {
                $$nodesToConsider.remove(iter.next());
            }
            $$positiveNodes.addAll($$oldLAofMergedFPI);
            $$oldLAofMergedFPI = null;
        }
    [/MVEL];

    IterationStreamProcessingBuilder {
        foreach $$nodesToConsider as $$pv {
            rule [DROOLS_TEMPLATE]
                when
                    Number( $average0 : doubleValue )
                    from accumulate(
                        MeasurementEvent( $val:value )
                        over window:time( 300s )
                        from entry-point "$${{pv?PVPowerProduced}}",
                        average ( $val )
                    ) eval($average0 < 30.0 )
                then
                    if(CONTEXT.noEarlyFiring("150s")){
                        CONTEXT.addToSet("$$positiveNodes","$${{pv}}");
                    }
                end
            [/DROOLS_TEMPLATE] publishes no stream manipulates context;
        }
    };

    postIteration [MVEL]
        $$verifiedNodes = new java.util.HashSet();

        $$resLon = -1;
        $$resLat = -1;

        if( $$positiveNodes.size() > 0 ){
            clusters = es.schaaf.cloudTracking.GeoNodeClustering.findClusters(
                CONTEXT , $$positiveNodes , 0.003 );
            // there must be at least one cluster.
            // The first one is always the biggest one.
            $$verifiedNodes.addAll(clusters[0].members);
            $$resLon = clusters[0].centerLon;
            $$resLat = clusters[0].centerLat;
        }
```

```
        $$positiveNodes.clear();

  [/MVEL];


    publish interim result $$verifiedNodes, $$resLon, $$resLat, $$startTime,
        $$endTime when [MVEL] $$verifiedNodes.size() > 1 && $$verifiedNodes.equals
        ($$lockedArea) [/MVEL];

    terminate if [MVEL] $$verifiedNodes.size() < 2 && $$iterationCounter < 2 [/
        MVEL] with result FalseSituation keep area registration if [MVEL]true[/
        MVEL];

    terminate if [MVEL] $$verifiedNodes.size() < 2 && $$iterationCounter >= 2 [/
        MVEL] with result $$verifiedNodes, $$resLon, $$resLat keep area
        registration if [MVEL]true[/MVEL];


    nextIterationTimeFrame startsAt [MVEL] if( $$verifiedNodes.equals($$lockedArea
        )) { $$endTime } else { $$startTime } [/MVEL] withDurationOf 300s ;

    nextLockedArea $$verifiedNodes;

    nextFocusArea from sparql "
        $${{verifiedNodes}} smartgrid:hasLocation ?LOC1.
        ?VALUE smartgrid:hasLocation ?LOC2.
        ?LOC1  smartgrid:hasLat ?LAT1.
        ?LOC2  smartgrid:hasLat ?LAT2.
        ?LOC1  smartgrid:hasLon ?LON1.
        ?LOC2  smartgrid:hasLon ?LON2.
        FILTER ( ?LAT1+0.0041 > ?LAT2 ) .
        FILTER ( ?LAT1-0.0041 < ?LAT2 ) .
        FILTER ( ?LON1+0.0041 > ?LON2 ) .
        FILTER ( ?LON1-0.0041 < ?LON2 ) .";

    mergeFunction [MVEL]
        if(CONTEXT_A.get("$$timeFrame").equals(CONTEXT_B.get("$$timeFrame"))){
            la = CONTEXT_A.get("$$lockedArea");
            CONTEXT_B.put("$$oldLAofMergedFPI",la);
        }
    [/MVEL];
};
```

B.1.1. Domain Specific Function

The cloud tracing scenario realization uses a domain specific clustering function in order to determine geographical clusters of solar panels based on their position available through the background knowledge. The function is realized as follows:

```
package es.schaaf.cloudTracking;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import es.schaaf.fsp.knowledgeBase.QueryFailedException;
import es.schaaf.fsp.model.MVELProcessingContext;

public class GeoNodeClustering {

  /**
```

```
 * Determines clusters of nodes based on the given maximal distance for
 * nodes that belong to the same cluster. Returns an array of
 * {@link Cluster} where the results are sorted descending by the cluster
 * size.
 *
 * @param CTX
 *            as the current MVEL processing context to access the
 *            background knowledge
 * @param nodes
 *            as the set of nodes that shall be clustered
 * @param as
 *            the maximum distance for nodes in a cluster
 */
public static Cluster[] findClusters(MVELProcessingContext CTX, Set<String>
    nodes, double distanceThreshold)
    throws QueryFailedException {

  List<ClusterPoint> clusterPoints = new ArrayList<>(nodes.size());

  int clusterCounter = 0;
  for (String s : nodes) {
    clusterPoints.add(new ClusterPoint(s, clusterCounter++, getGeoPosition(CTX,
        s)));
  }

  boolean changedSomething = false;

  Set<Integer> clusters = new HashSet<>();
  for (ClusterPoint clusterPoint : clusterPoints) {
    clusters.add(clusterPoint.clusterID);
  }

  for (Integer c : clusters) {

    do {
      changedSomething = false;
      for (ClusterPoint clusterPoint : clusterPoints) {
        int clusterID = clusterPoint.clusterID;

        if (clusterID != c)
          continue;

        double smallestDistance = Double.MAX_VALUE;
        ClusterPoint smallestDistanceTo = null;

        for (ClusterPoint clusterPoint2 : clusterPoints) {
          if (clusterPoint == clusterPoint2)
            continue;
          if (clusterPoint2.clusterID == clusterID)
            continue;

          double distance = clusterPoint.distance(clusterPoint2);

          if (distance < smallestDistance) {
            smallestDistance = distance;
            smallestDistanceTo = clusterPoint2;
          }
        }

        if (smallestDistance < distanceThreshold) {
          changedSomething = true;
          smallestDistanceTo.clusterID = clusterID;
        }

      }
    } while (changedSomething);
  }
```

```
  clusters = new HashSet<>();
  for (ClusterPoint clusterPoint : clusterPoints) {
    clusters.add(clusterPoint.clusterID);
  }

  Cluster[] resultingClusters = new Cluster[clusters.size()];

  clusterCounter = 0;
  for (Integer c : clusters) {
    Set<String> m = new HashSet<>();
    double lat = 0, lon = 0;

    for (ClusterPoint cp : clusterPoints) {
      if (cp.clusterID == c) {
        m.add(cp.id);
        lat += cp.geoPosition.m_lat;
        lon += cp.geoPosition.m_lon;
      }
    }
    lat /= m.size();
    lon /= m.size();

    Cluster cl = new Cluster(c, m, lat, lon);

    resultingClusters[clusterCounter++] = cl;
  }

  // sort the clusters by size
  Arrays.sort(resultingClusters, new BiggestClustersFirstComparator());

  return resultingClusters;
}

/*****************************************************************
 * Utils
 *****************************************************************/

private static Map<String, GeoPosition> m_positionCache = new HashMap<>();

/**
 * retrieve the position of the given solar panel from the background
 * knowledge base. NOTE: Uses a local cache of positions to reduce calls to
 * the knowledge base.
 */
private static GeoPosition getGeoPosition(MVELProcessingContext kb, String s)
    throws QueryFailedException {

  GeoPosition cached = m_positionCache.get(s);
  if (cached != null)
    return cached;

  String q = "select ?VALUE where { " + s + " smartgrid:hasLocation ?LOC . ?LOC
      smartgrid:hasLon ?VALUE. }";
  double lon = Double.parseDouble((String) kb.queryScalar(q));
  q = "select ?VALUE where { " + s + " smartgrid:hasLocation ?LOC . ?LOC
      smartgrid:hasLat ?VALUE. }";
  double lat = Double.parseDouble((String) kb.queryScalar(q));

  cached = new GeoPosition(lon, lat);
  m_positionCache.put(s, cached);
  return cached;
}

private static class BiggestClustersFirstComparator implements Comparator<
    Cluster> {
  @Override
  public int compare(Cluster o1, Cluster o2) {
    return o2.members.size() - o1.members.size();
```

```
    }
  }

  private static class GeoPosition {
    public final double m_lat;
    public final double m_lon;

    public GeoPosition(double lon, double lat) {
      m_lon = lon;
      m_lat = lat;
    }

    public double distance(GeoPosition g) {
      return Math.sqrt(Math.pow(m_lon - g.m_lon, 2) + Math.pow(m_lat - g.m_lat, 2)
          );
    }
  }

  private static class ClusterPoint {
    public int clusterID;
    public final String id;
    public final GeoPosition geoPosition;
    public Set<Integer> previousIDs = new HashSet<>();

    public ClusterPoint(String s, int i, GeoPosition g) {
      id = s;
      clusterID = i;
      previousIDs.add(i);
      geoPosition = g;
    }

    public double distance(ClusterPoint p2) {
      return geoPosition.distance(p2.geoPosition);
    }
  }
}
```

```
package es.schaaf.cloudTracking;

import java.util.Set;

public class Cluster {
  public final int id;
  public final Set<String> members;
  public final double centerLat;
  public final double centerLon;

  public Cluster(Integer c,Set<String> m,double cLat,double cLon) {
    id = c;
    members = m;
    centerLat = cLat;
    centerLon = cLon;
  }
}
```

B.2. Telecommunication Scenario

The Listing B.2 shows the complete Scenario Processing Template used for realizing the DoS Detection and Tracing Scenario as discussed in Section 7.5 and used for the test in Section 7.6.

```
Name "Dos Tracing";

SPARQL prefix "
PREFIX tests:<myCompany://tests#>
";

// load scenario specific accumulation function into Drools
DROOLS prefix [DROOLS_TEMPLATE]
import accumulate es.schaaf.dos.ExtendedSuddenChangeDetector
    ExtendedSuddenChangeDetector;
import            es.schaaf.dos.SuddenChange;
import            es.schaaf.dos.InterfaceDelta;
[/DROOLS_TEMPLATE];

PossibleSituationIndication {
  // select all interfaces of routers that are flagged for DoS monitoring
  $$indicationNodes from sparql "?VALUE   rdf:type        telco:interface.
                   ?VALUE   fsp:providesMeasurement ?point.
                   ?point   rdf:type        telco:trafficIn .
                   ?router  telco:hasInterface    ?VALUE .
                   ?router  rdf:type        telco:dosMonitoredRouter .
                   ";

    IndicationStreamProcessingBuilder{
        foreach $$indicationNodes as $$interface {
            rule [DROOLS_TEMPLATE]
                when
                    // package size dropped
                    $b : SuddenChange( consideredEvents == 6 , delta < -700 )
                    from accumulate(
                        $meB : MeasurementEvent( )
                        over window:length( 6 )
                        from entry-point "$${{interface?packageSizeAvgIn}}",
                        ExtendedSuddenChangeDetector ( $meB )
                    )
                then
                    publishIndication( "$${{interface}}" );
                end
            [/DROOLS_TEMPLATE] publishes indications;
        }
    }
};

FocusedProcessingInitialization {
    duplicationThreshold 600s;

    potentialLockedArea $$indicatedNodes;
    potentialFocusArea  $$indicatedNodes;

    initialTimeFrame startsAt [MVEL] $$indicatedTime - 30 [/MVEL] withDurationOf
        60s ;

    // if LA is part of already active FSP Instance, it is likely that the
        indication
    // is related to the already active instance
    collisionAction preventNew if LA overlap == 100% ;
};


FocusedSituationProcessing {
```

```
contextInitialization [MVEL]
  // flags used to terminate the processing
  $$pathComplete = false;
  $$trailLost    = false;
  $$message      = "";
  $$falseSituation = false;

  // if the trace was a success: contains the routers that are the origin of the DoS
  $$originRouters = new java.util.HashSet();

  // will contain all interfaces along the path
  $$verifiedInterfaces = new java.util.ArrayList();
  // contains all interfaces that have been verified by the current iteration
  $$iterationVerifiedInterfaces = new java.util.HashSet();
  // same for the previous iteration
  $$lastIterationVerifiedInterfaces = new java.util.ArrayList();

  // contains the results of the current iteration stream processing, the
  // deltas of package count and package size per interface if deltas are above threshold
  $$iterationDeltas = new java.util.ArrayList();

  // will contain the delta of package count and package size determined for
  // the attacked node during the first iteration
  $$deltaCount  = 0;
  $$deltaSize   = 0;
[/MVEL];

preIteration [MVEL]
  // prepare a new iteration...
  $$iterationDeltas.clear();

  // for the iteration stream processing we only need to look at
  // interfaces from the focus area that we don't consider as part
  // of the verified path yet
  $$interfacesToProcess = new java.util.HashSet($$focusArea);
  $$interfacesToProcess.removeAll($$verifiedInterfaces);

[/MVEL];

  IterationStreamProcessingBuilder {
      foreach $$interfacesToProcess as $$interface {
          rule [DROOLS_TEMPLATE]
              when
                  // package count increased
                  $a : SuddenChange(  consideredEvents == 6  , delta > 300 )
                  from accumulate(
                      $meA : MeasurementEvent( )
                      over window:length( 6 )
                      from entry-point "$${{interface?trafficIn}}",
                      ExtendedSuddenChangeDetector ( $meA )
                  )

                  // package size dropped
                  $b : SuddenChange(  consideredEvents == 6 , delta < -300 )
                  from accumulate(
                      $meB : MeasurementEvent( )
                      over window:length( 6 )
                      from entry-point "$${{interface?packageSizeAvgIn}}",
                      ExtendedSuddenChangeDetector ( $meB )
                  )
              then
                  CONTEXT.addToSet("$$iterationDeltas",new InterfaceDelta("$${{
                      interface}}",$a.getDelta(),$b.getDelta()));
              end
          [/DROOLS_TEMPLATE] publishes no stream manipulates context;
      }
```

```
};

postIteration [MVEL]

if( $$iterationCounter == 1 ) {
  // in the first iteration sum up and store the detected deltas of the
      attacked node(s)
  foreach( i : $$iterationDeltas){
    $$deltaCount   += i.getDeltaTraffic();
    $$deltaSize    += i.getDeltaSize();
    $$iterationVerifiedInterfaces.add(i.getInterface());
    $$verifiedInterfaces.add(i.getInterface());
  }

  if($$iterationVerifiedInterfaces.size()==0){
    $$falseSituation = true;
  }
}else{
  if ($$interfacesToProcess.size() == 0 ) {
    // if there where no interfaces to process,
    // check if trace is complete or if we lost the path


      // check if the last step connected only to border routers
      allBorderRouters = true;

      foreach( i : $$lastIterationVerifiedInterfaces){
        res = CONTEXT.querySet(" SELECT DISTINCT ?VALUE WHERE { " + i + "
            telco:hasLink ?LNK . ?srcInterface telco:hasLink ?LNK . ?VALUE
            telco:hasInterface ?srcInterface . ?VALUE rdf:type telco:
            edgeRouter }");

        if(res.size() == 0){
          allBorderRouters = false;
        }else{
          // res contains the Router where the attack is comming from..
          $$originRouters.addAll(res);
        }
      }


      // if not only there are still other routers, we didn't complete
      // the trace but lost the trail
      if( !allBorderRouters ) {
        $$trailLost = true;
      }else{
        $$pathComplete = true;
      }

  }else{
    // otherwise continue with the trace

    $$lastIterationVerifiedInterfaces.clear();
    $$lastIterationVerifiedInterfaces.addAll($$iterationVerifiedInterfaces);
    $$iterationVerifiedInterfaces.clear();

    trafficSum = 0;
    java.util.Collections.sort($$iterationDeltas);
    foreach( i : $$iterationDeltas){
      if(trafficSum < $$deltaCount){
          $$iterationVerifiedInterfaces.add(i.getInterface());
          $$verifiedInterfaces.add(i.getInterface());
          trafficSum += i.getDeltaTraffic();
      }
    }

    // if less then 80% of the package count was found on the considered
        interfaces
```

```
        // the trace is stopped as the path can't be traced anymore
        if(trafficSum < ($$deltaCount * 0.8) ){
          $trailLost = true;
          $$message = "Path was lost during iteration " + $$iterationCounter + "
              as less then 80% of the traffic could be found";
        }
      }
    }

    // we want to have more control over the next Focus Area so we build it
        ourselfs
    $$nextFA = new java.util.HashSet();
    // query all interfaces of routers reachable from the interfaces verified in
        the current iteration
    foreach ( n : $$iterationVerifiedInterfaces ){
      $$nextFA.addAll(CONTEXT.querySet(" SELECT DISTINCT ?VALUE WHERE { " + n + "
          telco:hasLink ?LNK . ?srcInterface telco:hasLink ?LNK . ?srcNode telco:
          hasInterface ?srcInterface . ?srcNode telco:hasInterface ?VALUE }"));
    }
    // also add all verified interfaces to mark our path
    $$nextFA.addAll($$verifiedInterfaces);
    [/MVEL];

    // publish our current tracking state if the processing is not yet finished
    publish interim result $$verifiedInterfaces when [MVEL]
        $$iterationVerifiedInterfaces.size() > 0 [/MVEL];

    // terminate if no DoS could be verified
    terminate if [MVEL] $$falseSituation [/MVEL]
        with result FalseSituation keep area registration if [MVEL]false[/MVEL];

    // terminate if we traced the path
    terminate if [MVEL] $$pathComplete [/MVEL]
        with result $$verifiedInterfaces, $$originRouters keep area registration
            if [MVEL]true[/MVEL];

    // terminate if we can't follow the path any further
    terminate if [MVEL] $$trailLost [/MVEL]
        with result $$message, $$verifiedInterfaces keep area registration if [
            MVEL]true[/MVEL];

    // For DoS tracing the Time window is fixed so we can track the path
    nextIterationTimeFrame startsAt $$startTime withDurationOf 60s ;

    // lockedArea is fixed to the nodes that are under attack
    nextLockedArea $$lockedArea;

    // focusArea was determined in a custom way during pre-Processing
    nextFocusArea $$nextFA;

    // as the Locked Area never moves, no merge possible
    mergeFunction [MVEL][/MVEL];
};
```

C. Further Tests

Chapter 7 discusses the evaluation of the processing model and language based on the realization of two scenarios as well as the discussion of two test cases based on these scenarios. In addition to these test cases, this chapter discusses three additional cases which focus on more specific cases and how the processing model handles them:

- The handling of False Situations (Test Case 2)
- The handling of multiple Focused Situation Processing Instances for the *same* actual situation and their required merging (Test Case 3)
- The handling of temporary collisions of two distinct situations and the resulting loss of the situation identities (Test Case 4).

C.1. Case 2: False Situations

In addition to the detection and tracing of valid situations (Test Case 1 on page 174), the negative case is demonstrated by this test case. Based on the scenario definition, a cloud needs to cover more than one solar panel to distinguish clouds from temporary failing solar panel installations (Section 2.1.1).

In order to demonstrate the classification of incorrect indications as false situations, a data set was generated were two solar panels report a low energy production shortly after the processing begins. The two panels where chosen so that together they do not form a single cluster of shaded panels. Further the two panels where chosen so that the potential Focus Areas generated for each panel based on the possible situation indication will include the other solar panel that reports low energy production[1]. Due to this overlap, a started Focused Situation Processing Instance will in its first iteration see more than one solar panel as shaded and needs to determine if these shaded panels warrant for a cloud or if they are too separated to be considered.

The following subsections discuss the processing done by the processing system for this case.

C.1.1. Phase 0: Possible Situation Indication Processing Initialization

The indication processing is set up in the same way as for Test Case 1 and is thus not discussed again (See Subsection 7.4.1).

C.1.2. Phase 1: Possible Situation Indication Processing

The Stream Processing Topology defined in Phase 0 was instantiated in Phase 1 and generated two separate Possible Situation Indication Events (Table C.1.1), one for the

[1]The case where the failing solar panels are even further apart is not explicitly considered as it is a simplification of the here considered case.

#	Indicated Time	Indicated Nodes	Initial TimeFrame	Potential LockedArea	P2 Classification	Resulting Focused Situation Processing Instance
1	1438296630	tests:panel_22	1438296630-1438296930	tests:panel_22	New Possible Situation	FP__CloudTracking1__a660dfed-1b15-4723-b481-e4632ebf86e7
2	1438296630	tests:panel_44	1438296630-1438296930	tests:panel_44	Ignored Indication	

Table C.1.1.: Case 2: Generated Indication Events.

solar panel 22 and one for the solar panel 44. Both indications were generated for the same indication time as both solar panels simultaneously stop producing. Both possible situation indication events were forwarded to the Phase 2 processing.

C.1.3. Phase 2: Focused Situation Processing Initialization

As the behavior is similar to the Phase 2 processing already discussed for Test Case 1 (Subsection 7.4.3) only a brief summary is given.

The system first processed the indication for solar panel 22. Its classification resulted in the creation of a new Focused Situation Processing Instance as shown in Table C.1.1. The newly created instance is referred to as #1 for the remainder of the test case.

The Possible Situation Indication Event received for solar panel 44 is processed afterwards. As however the generated potential Locked Area for this indication (consisting of the solar panel 44) is a subset of the Focus Area of the already created Focused Situation Processing Instance #1 (Figure C.1.2), the Indication Event was classified as *Ignored* and no further processing was triggered based on it.

As only two Possible Situation Indication Events where generated, the Focused Situation Processing Initialization is finished.

C.1.4. Phase 3: Focused Situation Processing

A single Focused Situation Processing Instance #1 was started based on the Phase 2 classification.

The Instance #1 generates and executed the stream processing as discussed for Test Case 1 (Subsection 7.4.4). The result of the stream processing is the set *$$positiveNodes* which contained the two shaded solar panels 22 & 44 as they are both not producing energy and are part of the current iterations Focus Area.

The *postIteration* processing step tried to cluster the two nodes which results in the generation of *two separate clusters* as the solar panels are too far apart to be part of the same cluster. The *postIteration* processing assigned the cluster with the highest number of nodes to *$$verifiedNodes*. As both clusters had the same size, no specific precedence is defined. As a result of the post processing, the set *$$verifiedNodes* contained one single solar panel.

The Iteration is terminated as the identified cluster only contained a single node. As the FSP Instance was still in the first iteration, the termination rule for terminating the processing if no cloud was found (Listing B.1 Line 123) terminated the processing with the *FalseSituation* result as shown in Table C.1.2.

#	Event Type	Time	Event Contents
1	FinalResultEvent	1438296930	FalseSituaton

Table C.1.2.: Case 2: Result event generated by the Focused Situation Processing Instance #1.

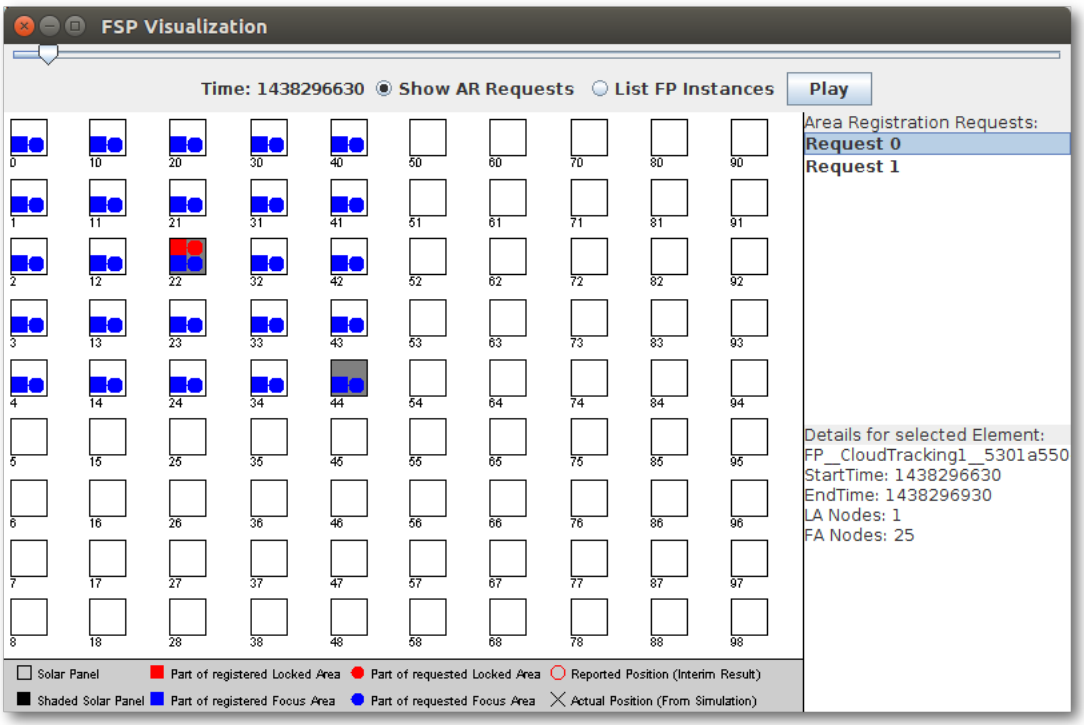


Figure C.1.1.: Case 2: The Area Registration (circles) resulting from the Possible Situation Indication Event.

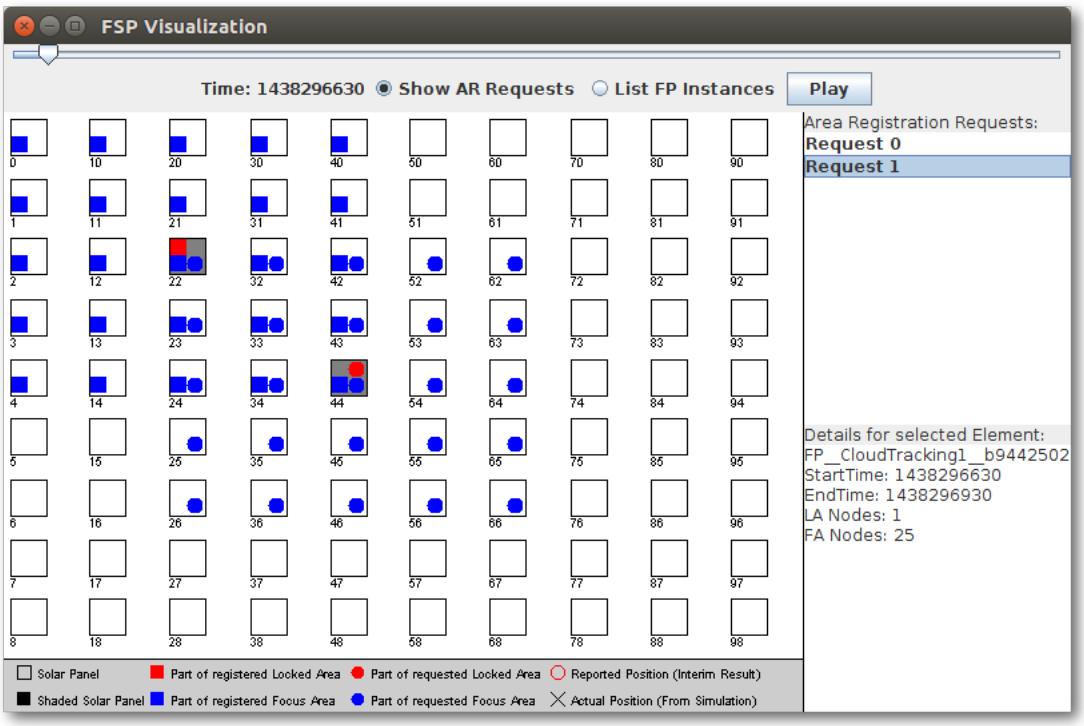


Figure C.1.2.: Case 2: The Area Registration Request 1 (circles) which resulted from the second Possible Situation Indication Event which was generated for the solar panel 44. The potential Locked Area of this registration is a subset of the Focus Area of the running Focused Situation Processing Instance #1 shown as the blue rectangles.

C.1.5. Conclusions from the Test Results

This second test case *successfully demonstrated the possible situation verification process* by correctly identifying possible situation indications caused by fluctuations in the solar energy production as False Situations.

C.2. Case 3: Multiple Focused Situation Processing Instances for One Cloud

For this test case, a single cloud was simulated like for Test Case 1. However, for this test case the clouds size was increased and the cloud was simulated with a rectangular shape so that the cloud covers a rectangular field of 6x6 Solar Panels. Due to the shape change and the increase size, the cloud immediately shades 6 solar panels when it first enters the monitored area (Panels 2 to 7) as shown in Figure C.2.1.

Due to the great distance between the outer edges of the shaded area (Panel 2 on the top edge and Panel 7 on the bottom edge) the initial Focus Area generated during Phase 2 does not cover the whole shaded area. Due to this, the Phase 2 Possible Situation Indication Classification was not able to correctly assign all indication events generated for the cloud to the *same* Focused Situation Processing Instance. As a result multiple (in this case two) Focused Situation Processes were started for the same Situation.

The processing model handles such a case by merging the two processing instances into one, which is demonstrated by this test case:

The following sections discuss the handling of this case in detail for each of the processing steps.

C.2.1. Phase 0: Possible Situation Indication Processing Initialization

The indication processing was set up in the same way as for Test Case 1 and is thus not discussed here (See Subsection 7.4.1).

C.2.2. Phase 1: Possible Situation Indication Processing

As the cloud entered the monitored area by simultaneously shading the first six panels on the left hand side of the monitored area (Figure C.2.1), the stream processing generated six Possible Situation Indication Events as shown in Table C.2.1 in Lines 1-6. Over time as the cloud moved further to the right, the Phase 1 processing raised further indications for the newly shaded solar panels as shown for the next set of panels shaded by the cloud in Lines 7 to 12. As the cloud moved further over the monitored area, further indications where raised in the same way.

C.2.3. Phase 2: Focused Situation Processing Initialization

As the behavior is similar to the Phase 2 processing already discussed for Test Case 1 (Subsection 7.4.3) only a brief summary is given:

1. The very first Possible Situation Indication Event that was raised for Panel 3 resulted in the instantiation of the first Focused Situation Processing Instance here referred to as „#1” (Table C.2.1 Line 1).

#	Indicated Time	Indicated Nodes	Initial TimeFrame	Potential LockedArea	P2 Classification	Resulting Focused Situation Processing Instance
1	1438293780	tests:panel_3	1438293780-1438294080	tests:panel_3	New Possible Situation	FP__CloudTracking1__81844638-ecad-4c59-843e-c7bd7bbf1bb3
2	1438293780	tests:panel_2	1438293780-1438294080	tests:panel_2	Ignored Indication	
3	1438293780	tests:panel_6	1438293780-1438294080	tests:panel_6	New Possible Situation	FP__CloudTracking1__9bd3f406-eb85-4127-813d-4e4a52104134
4	1438293780	tests:panel_7	1438293780-1438294080	tests:panel_7	Ignored Indication	
5	1438293780	tests:panel_5	1438293780-1438294080	tests:panel_5	Ignored Indication	
6	1438293780	tests:panel_4	1438293780-1438294080	tests:panel_4	Ignored Indication	
7	1438299240	tests:panel_17	1438299240-1438299540	tests:panel_17	Ignored Ind. (LA Collision)	
8	1438299240	tests:panel_15	1438299240-1438299540	tests:panel_15	Ignored Ind. (LA Collision)	
9	1438299240	tests:panel_13	1438299240-1438299540	tests:panel_13	Ignored Ind. (LA Collision)	
10	1438299240	tests:panel_12	1438299240-1438299540	tests:panel_12	Ignored Ind. (LA Collision)	
11	1438299240	tests:panel_14	1438299240-1438299540	tests:panel_14	Ignored Ind. (LA Collision)	
12	1438299240	tests:panel_16	1438299240-1438299540	tests:panel_16	Ignored Ind. (LA Collision)	
...	...	...	...	...	...	...

Table C.2.1.: Case 3: Generated Indication Events.

2. The second Possible Situation Indication Event concerned the Panel 2. As the resulting potential Locked Area is a subset of the registered Focus Area of the already started Focused Situation Processing Instance #1, the event is ignored (Line 2).

3. The third Possible Situation Indication Event concerned the Panel 6. In this case the resulting potential Locked Area was *not* a subset of the already started Focused Situation Processing Instance #1. Therefore, an additional Focused Situation Processing Instance, here referred to as #2 was started for the same cloud (Line 3).

The following three Possible Situation Indication Events (Lines 4-6) were classified in the same way as the second Possible Situation Indication Event.

All remaining Possible Situation Indication Events were classified as Ignored and dropped due to their collision with the Locked Area of the Focused Situation Processing Instance[2] that was already tracking the cloud.

C.2.4. Phase 3: Focused Situation Processing

Two Focused Situation Indication Processing Instances where started (#1 and #2) based on the Phase 2 classification (Table C.2.1 Lines 1 & 3). From the perspective of the processing system these two FSP Instances concerned separate situations and both processing instances where executed as such. However, as the instances in fact concern the same actual situation (Cloud), the processing system detected a collision of their Locked Areas and initiated a merge of the two instances. In the process the FSP Instance #2 was terminated

[2]The Possible Situation Indication Events collide only with *one* Focused Situation Processing Instance as one of the two initially started instance is merged into the other one as they concern the same situation (See the following Section C.2.4).

while the FSP Instance #1 was allowed to continue. The following sub-sections discuss this process in detail.

Note: The following discussions of the initial iterations of FSP Instances #1 and #2 all concern the same time frame (Table C.2.2 Column 2).

C.2.4.1. Iteration 1 of Instance #1

The first iteration was executed based on the Area Registration created during the Phase 2 classification as shown in Table C.2.2 Line 1. The processing determined in the first iteration that the nodes 2, 4 and 5 are also affected by the investigated situation. As such FSP Instance #1 requested an update of its Area Registration where the requested Area Registration contained the nodes 2 to 5 as the Locked Area. As no collision was detected, the Area Registration was granted (Table C.2.2 Line 3) and the FSP Instance #1 continues with its second iteration.

C.2.4.2. Iteration 1 of Instance #2

The first iteration of the FSP Instance #2 was also executed based on the Area Registration created during the Phase 2 classification as shown in Table C.2.2 Line 2. The processing determined that the nodes 4, 5 and 7 are also affected by the investigated situation. As such the FSP Instance #2 requested an update of its Area Registration for the current time frame where the requested new Area Registration contained the nodes 4 to 7 as the Locked Area.

However, as the already updated Area Registration of FSP Instance #1 collided with the requested Area Registration of FSP Instance #2, the Area Registration was *not* granted (Table C.2.2 Line 4). This collision triggered the *merge* processing between the two FSP Instances #1 and #2.

As the Area Registration request of FSP Instance #2 was rejected, this instance was immediately stopped. Afterwards the merge process waited for the next Area Registration Request of FSP Instance #1 during which the FSP Instance #1 was also paused while the merge is executed.

C.2.4.3. Iteration 2 of Instance #1

The second iteration of FSP Instance #2 was based on the updated Area Registration from the first iteration. The processing determined that the nodes 6 and 7 were also affected and thus requested an update of its Area Registration where the requested Locked Area contained the nodes 2 to 7 (Table C.2.2 Line 5).

As discussed in the previous subsection, the FSP Instance #2 collided with the FSP Instance #1. Due to this collision the FSP Instance #1 was temporarily paused once it requested the new Area Registration. While paused the merge function (Subsection 7.3.1.3) was executed.

The merge function copied the requested Locked Area from the FSP Instance #2 to its own processing context into the variable *$$oldLAofMergedFPI*. Afterwards all Area Registrations of the FSP Instance #2 for the current time frame were released and the merge process was finished.

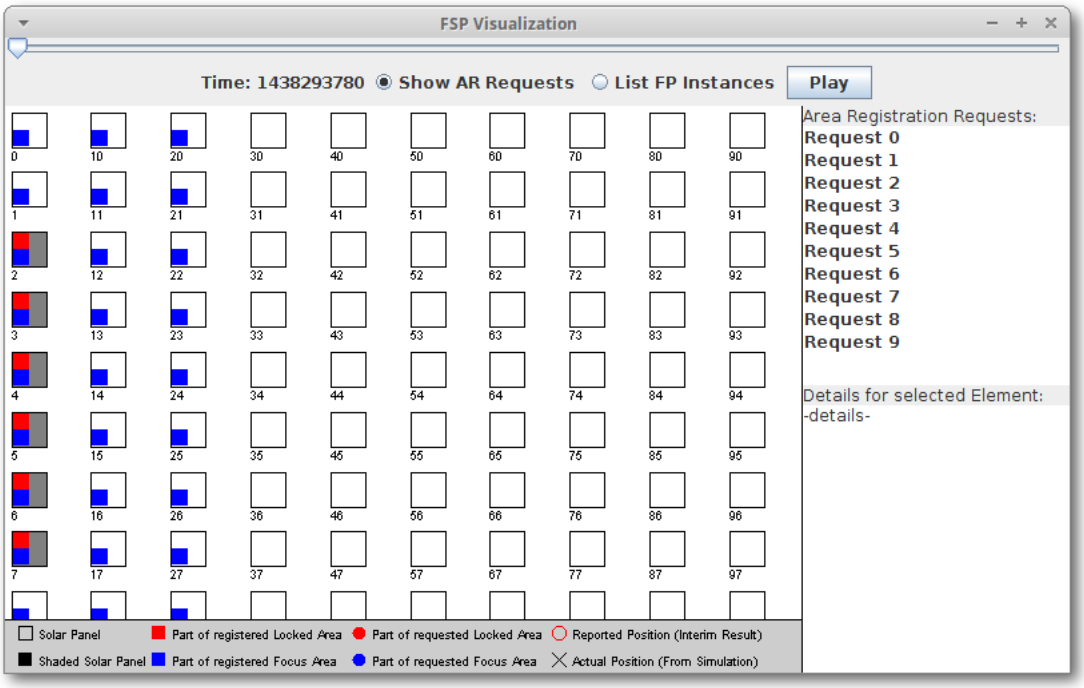


Figure C.2.1.: Case 3: Locked and Focus Area resulting from the merge of the two initially started FSP Instances and the later growth of the Locked and Focus Area of the surviving FSP Instance.

As the merge process released the last valid Area Registration of FSP Instance #2, which contained the node 6 as Locked Area, the Area Registration Request from FSP Instance #1 was granted as no collision wss detected (Table C.2.2 Line 5).

As the Area Registration was successful, the FSP Instance #1 continued with the next iteration.

C.2.4.4. Iteration 3 of Instance #1

The third iteration of FSP Instance #1 is based on the updated Area Registration from iteration 2 and based on the results of the merge that took place at the end of iteration 2.

As the merge processing set the variable *$$oldLAofMergedFPI* based on the requested Locked Area of FSP Instance #2, all nodes contained in this set have already been verified as being shaded. Thus the contained nodes were not considered anymore during the iteration stream processing but instead directly added to the set *$$positiveNodes* so that they were taken into account during the iterations post processing. Otherwise, the processing of this iteration happened in the same way as the other iterations.

As the affected nodes matched the Locked Area of the current Area Registration, the registration was *not* updated but instead the processing continued with the next consecutive time frame as discussed for Test Case 1. From here on the processing continued in the same way as already discussed for Test Case 1.

#	Time Frame	Requesting Focused Situation Processing Instance	Requested Locked Area	Requested Focus Area	Registration Granted?
1	1438293780 - 1438294080	FP__CloudTracking1__ 81844638-ecad-4c59-843e-c7bd7bbf1bb3 #1	tests:panel_3;	tests:panel_12; tests:panel_23; tests:panel_11; tests:panel_22; tests:panel_14; tests:panel_25; tests:panel_13; tests:panel_24; tests:panel_1; tests:panel_2; tests:panel_21; tests:panel_3; tests:panel_15; tests:panel_4; tests:panel_5;	Yes
2	1438293780 - 1438294080	FP__CloudTracking1__ 9bd3f406-eb85-4127-813d-4e4a5210413 #2	tests:panel_6;	tests:panel_14; tests:panel_25; tests:panel_24; tests:panel_16; tests:panel_27; tests:panel_15; tests:panel_26; tests:panel_18; tests:panel_17; tests:panel_28; tests:panel_4; tests:panel_5; tests:panel_6; tests:panel_7; tests:panel_8;	Yes
3	1438293780 - 1438294080	FP__CloudTracking1__ 81844638-ecad-4c59-843e-c7bd7bbf1bb3 #1	tests:panel_2; tests:panel_3; tests:panel_4; tests:panel_5;	tests:panel_23; tests:panel_22; tests:panel_25; tests:panel_24; tests:panel_21; tests:panel_20; tests:panel_27; tests:panel_26; tests:panel_4; tests:panel_5; tests:panel_6; tests:panel_7; tests:panel_12; tests:panel_11; tests:panel_14; tests:panel_13; tests:panel_0; tests:panel_1; tests:panel_10; tests:panel_2; tests:panel_3; tests:panel_16; tests:panel_15; tests:panel_17;	Yes
4	1438293780 - 1438294080	FP__CloudTracking1__ 9bd3f406-eb85-4127-813d-4e4a52104134 #2	tests:panel_4; tests:panel_5; tests:panel_6; tests:panel_7;	tests:panel_23; tests:panel_22; tests:panel_25; tests:panel_24; tests:panel_27; tests:panel_26; tests:panel_29; tests:panel_28; tests:panel_4; tests:panel_5; tests:panel_6; tests:panel_7; tests:panel_8; tests:panel_9; tests:panel_12; tests:panel_14; tests:panel_13; tests:panel_2; tests:panel_3; tests:panel_19; tests:panel_16; tests:panel_15; tests:panel_18; tests:panel_17;	No Collision with: FP__CloudTracking1__ 81844638-ecad-4c59-843e-c7bd7bbf1bb3
5	1438293780 - 1438294080	FP__CloudTracking1__ 81844638-ecad-4c59-843e-c7bd7bbf1bb3 #1	tests:panel_2; tests:panel_3; tests:panel_4; tests:panel_5; tests:panel_6; tests:panel_7;	tests:panel_23; tests:panel_22; tests:panel_25; tests:panel_24; tests:panel_21; tests:panel_20; tests:panel_27; tests:panel_26; tests:panel_29; tests:panel_28; tests:panel_4; tests:panel_5; tests:panel_6; tests:panel_7; tests:panel_8; tests:panel_9; tests:panel_12; tests:panel_11; tests:panel_14; tests:panel_13; tests:panel_0; tests:panel_1; tests:panel_10; tests:panel_2; tests:panel_3; tests:panel_19; tests:panel_16; tests:panel_15; tests:panel_18; tests:panel_17;	Yes

Table C.2.2.: Case 3: Area Registration Requests and their Outcome from the two started Focused Situation Processing Instance for the initial Time Frame.

C.2.5. Conclusions

The test case demonstrated the merge of two FSP Instances created for the same situation. With this it also demonstrated the impact of too small initial Focus Areas which prevented the Phase 2 classification to correctly correlate all Possible Situation Indications raised for a single situation thus causing the creation of two separate FSP Instances which then need to be merged during the Phase 3 processing.

C.3. Case 4: Temporary Situation Collision

This final cloud tracking test case demonstrates two separate aspects of the processing model:

Part 1: The detection and tracking of *multiple independent situations*.

Part 2: a *temporary overlap of two situations* which prevents the processing system to distinguish them.

While the first part demonstrates normal model functionality, the second part shows a limitation of the processing model.

C.3.1. Part 1: Two Separate Situations

The first part of this test case discusses the normal behavior of the processing model when detecting and tracking more than one situation.

#	Indicated Time	Indicated Nodes	Initial TimeFrame	Potential LockedArea	P2 Classification	Resulting Focused Situation Processing Instance
1	1438294230	tests:panel_4	1438294230-1438294530	tests:panel_4	New Possible Situation	FP__CloudTracking1__2f931009-0ba7-45bb-893a-3a39add6c860
2	1438294230	tests:panel_5	1438294230-1438294530	tests:panel_5	Ignored Indication	
3	1438294560	tests:panel_50	1438294560-1438294860	tests:panel_50	New Possible Situation	FP__CloudTracking1__08d45f6c-f8c3-47b0-b1cc-e2b0597f2eca
4	1438294560	tests:panel_40	1438294560-1438294860	tests:panel_40	Ignored Indication	
...	...	...	...	...	...	...
33	1438343340	tests:panel_94	1438343340-1438343640	tests:panel_94	New Possible Situation	FP__CloudTracking1__76416d89-9f19-4667-90bc-250e51555d84
34	1438343340	tests:panel_95	1438343340-1438343640	tests:panel_95	Ignored Indication	
35	1438343640	tests:panel_49	1438343640-1438343940	tests:panel_49	Ignored Indication	
36	1438343640	tests:panel_59	1438343640-1438343940	tests:panel_59	Ignored Indication	

Table C.3.1.: Case 4: Generated Possible Situation Indication Events (only the initial 4 events and the final 4 are shown).

C.3.1.1. Phase 0: Possible Situation Indication Processing Initialization

The indication processing is set up in the same way as for Test Case 1 and is thus not discussed here (See Subsection 7.4.1).

C.3.1.2. Phase 1 & 2: Possible Situation Indication Processing and Focused Situation Processing Initialization

The first cloud entered the monitored area at 1438294230 from the west by shading the solar panels 4 and 5 causing two individual Possible Situation Indication Events (Table C.3.1 Lines 1 & 2). In the same way as discussed for Test Case 1, the first indication event was classified as „New Possible Situation" causing the creation of a new Focused Situation Processing Instance (later refereed to as #1) while the second indication was ignored.

A little later at 1438294560 the second cloud entered the monitored area from the north by shading the solar panels 40 and 50 resulting in two separate Possible Situation Indication Events (Table C.3.1 Lines 3 & 4). As the generated potential Locked Areas for both indications did not collide with the Area Registration of the FSP Instance #1, the classification took place in the same way as for the two initial indications resulting in the creation of another Focused Situation Proceeding Instance (later refereed to as #2).

Over time as the clouds moved further into the monitored area, the P1 processing raised further indications for the newly shaded solar panels in the same way as discussed for Test Case 1.

The classification of the Possible Situation Indication Events 33ff are discussed in Part 2 (Subsection C.3.2) as they are the result of the temporary situation merge.

C.3.1.3. Phase 3: Focused Situation Processing

Two FSP Instances where started by Phase 2. Each followed a separate cloud, Instance #1 the cloud that entered the monitored area from the west, Instance #2 the cloud that entered from the north.

#	Time Frame	Requesting Focused Situation Processing Instance	Requested Locked Area	Requested Focus Area	Registration Granted?
0	1438294230 - 1438294530	FP__CloudTracking1__2f931009-0ba7-45bb-893a-3a39add6c860 #1	tests:panel_4	tests:panel_12 tests:panel_23 tests:panel_22 tests:panel_14 tests:panel_25 tests:panel_13 tests:panel_24 tests:panel_2 tests:panel_3 tests:panel_16 tests:panel_15 tests:panel_26 tests:panel_4 tests:panel_5 tests:panel_6	Yes
2	1438294230 - 1438294530	FP__CloudTracking1__2f931009-0ba7-45bb-893a-3a39add6c860 #1	tests:panel_4 tests:panel_5	tests:panel_12 tests:panel_23 tests:panel_22 tests:panel_14 tests:panel_25 tests:panel_13 tests:panel_24 tests:panel_2 tests:panel_3 tests:panel_16 tests:panel_27 tests:panel_15 tests:panel_26 tests:panel_17 tests:panel_4 tests:panel_5 tests:panel_6 tests:panel_7	Yes
4	1438294560 - 1438294860	FP__CloudTracking1__08d45f6c-f8c3-47b0-b1cc-e2b0597f2eca #2	tests:panel_50	tests:panel_30 tests:panel_41 tests:panel_52 tests:panel_40 tests:panel_51 tests:panel_62 tests:panel_32 tests:panel_31 tests:panel_42 tests:panel_70 tests:panel_50 tests:panel_61 tests:panel_72 tests:panel_60 tests:panel_71	Yes
6	1438294560 - 1438294860	FP__CloudTracking1__08d45f6c-f8c3-47b0-b1cc-e2b0597f2eca #2	tests:panel_40 tests:panel_50	tests:panel_22 tests:panel_30 tests:panel_41 tests:panel_52 tests:panel_40 tests:panel_51 tests:panel_62 tests:panel_21 tests:panel_32 tests:panel_20 tests:panel_31 tests:panel_42 tests:panel_70 tests:panel_50 tests:panel_61 tests:panel_72 tests:panel_60 tests:panel_71	Yes
...	...	...	...	...	...
131	1438310760 - 1438311060	FP__CloudTracking1__08d45f6c-f8c3-47b0-b1cc-e2b0597f2eca #2	tests:panel_52 tests:panel_41 tests:panel_51 tests:panel_42	tests:panel_23 tests:panel_44 tests:panel_22 tests:panel_24 tests:panel_41 tests:panel_63 tests:panel_40 tests:panel_62 tests:panel_43 tests:panel_21 tests:panel_42 tests:panel_64 tests:panel_20 tests:panel_61 tests:panel_60 tests:panel_34 tests:panel_33 tests:panel_30 tests:panel_52 tests:panel_74 tests:panel_51 tests:panel_73 tests:panel_32 tests:panel_54 tests:panel_31 tests:panel_53 tests:panel_70 tests:panel_50 tests:panel_72 tests:panel_71	Yes
132	1438310760 - 1438311060	FP__CloudTracking1__08d45f6c-f8c3-47b0-b1cc-e2b0597f2eca #2	tests:panel_34 tests:panel_24 tests:panel_52 tests:panel_41 tests:panel_51 tests:panel_43 tests:panel_42 tests:panel_53	tests:panel_45 tests:panel_44 tests:panel_46 tests:panel_41 tests:panel_40 tests:panel_43 tests:panel_42 tests:panel_12 tests:panel_56 tests:panel_55 tests:panel_14 tests:panel_13 tests:panel_52 tests:panel_51 tests:panel_54 tests:panel_53 tests:panel_16 tests:panel_15 tests:panel_50 tests:panel_23 tests:panel_22 tests:panel_25 tests:panel_24 tests:panel_63 tests:panel_62 tests:panel_21 tests:panel_65 tests:panel_64 tests:panel_20 tests:panel_26 tests:panel_4 tests:panel_5 tests:panel_6 tests:panel_61 tests:panel_60 tests:panel_34 tests:panel_33 tests:panel_36 tests:panel_35 tests:panel_30 tests:panel_74 tests:panel_73 tests:panel_32 tests:panel_2 tests:panel_3 tests:panel_31 tests:panel_75 tests:panel_70 tests:panel_72 tests:panel_71	No Collision with: FP__CloudTracking1__2f931009-0ba7-45bb-893a-3a39add6c860
...	...	...	...	...	...
135	1438311030 - 1438311330	FP__CloudTracking1__2f931009-0ba7-45bb-893a-3a39add6c860 #1	tests:panel_34 tests:panel_14 tests:panel_25 tests:panel_35 tests:panel_24 tests:panel_15	tests:panel_23 tests:panel_45 tests:panel_22 tests:panel_44 tests:panel_25 tests:panel_47 tests:panel_24 tests:panel_46 tests:panel_43 tests:panel_42 tests:panel_27 tests:panel_26 tests:panel_4 tests:panel_5 tests:panel_6 tests:panel_7 tests:panel_12 tests:panel_34 tests:panel_56 tests:panel_33 tests:panel_55 tests:panel_14 tests:panel_36 tests:panel_13 tests:panel_35 tests:panel_57 tests:panel_52 tests:panel_32 tests:panel_54 tests:panel_2 tests:panel_53 tests:panel_3 tests:panel_16 tests:panel_15 tests:panel_37 tests:panel_17	Yes
136	1438311030 - 1438311330	FP__CloudTracking1__2f931009-0ba7-45bb-893a-3a39add6c860 #1	tests:panel_34 tests:panel_25 tests:panel_35 tests:panel_24 tests:panel_52 tests:panel_43 tests:panel_53 tests:panel_42	tests:panel_45 tests:panel_44 tests:panel_47 tests:panel_46 tests:panel_40 tests:panel_12 tests:panel_56 tests:panel_55 tests:panel_14 tests:panel_13 tests:panel_57 tests:panel_54 tests:panel_16 tests:panel_15 tests:panel_17 tests:panel_50 tests:panel_23 tests:panel_22 tests:panel_25 tests:panel_63 tests:panel_62 tests:panel_21 tests:panel_65 tests:panel_64 tests:panel_20 tests:panel_27 tests:panel_26 tests:panel_4 tests:panel_5 tests:panel_6 tests:panel_61 tests:panel_7 tests:panel_60 tests:panel_33 tests:panel_36 tests:panel_35 tests:panel_30 tests:panel_74 tests:panel_73 tests:panel_32 tests:panel_2 tests:panel_3 tests:panel_31 tests:panel_75 tests:panel_37 tests:panel_70 tests:panel_72 tests:panel_71	Yes
137	1438311030 - 1438311330	FP__CloudTracking1__2f931009-0ba7-45bb-893a-3a39add6c860 #1	tests:panel_34 tests:panel_25 tests:panel_24 tests:panel_35 tests:panel_52 tests:panel_41 tests:panel_51 tests:panel_43 tests:panel_42 tests:panel_53	tests:panel_45 tests:panel_44 tests:panel_47 tests:panel_46 tests:panel_41 tests:panel_40 tests:panel_43 tests:panel_42 tests:panel_12 tests:panel_56 tests:panel_55 tests:panel_14 tests:panel_13 tests:panel_57 tests:panel_52 tests:panel_51 tests:panel_54 tests:panel_53 tests:panel_16 tests:panel_15 tests:panel_17 tests:panel_50 tests:panel_23 tests:panel_22 tests:panel_25 tests:panel_24 tests:panel_63 tests:panel_62 tests:panel_21 tests:panel_65 tests:panel_64 tests:panel_20 tests:panel_27 tests:panel_26 tests:panel_4 tests:panel_5 tests:panel_6 tests:panel_61 tests:panel_7 tests:panel_60 tests:panel_34 tests:panel_33 tests:panel_36 tests:panel_35 tests:panel_30 tests:panel_74 tests:panel_73 tests:panel_32 tests:panel_2 tests:panel_3 tests:panel_31 tests:panel_75 tests:panel_37 tests:panel_70 tests:panel_72 tests:panel_71	Yes
..	...	...	...	...	...
260	1438338330 - 1438338630	FP__CloudTracking1__2f931009-0ba7-45bb-893a-3a39add6c860 #1	tests:panel_56 tests:panel_58 tests:panel_47 tests:panel_46 tests:panel_57 tests:panel_48	tests:panel_45 tests:panel_67 tests:panel_44 tests:panel_66 tests:panel_47 tests:panel_69 tests:panel_25 tests:panel_46 tests:panel_68 tests:panel_24 tests:panel_65 tests:panel_64 tests:panel_49 tests:panel_27 tests:panel_48 tests:panel_26 tests:panel_29 tests:panel_28 tests:panel_34 tests:panel_56 tests:panel_78 tests:panel_55 tests:panel_77 tests:panel_36 tests:panel_58 tests:panel_35 tests:panel_57 tests:panel_79 tests:panel_74 tests:panel_54 tests:panel_76 tests:panel_75 tests:panel_38 tests:panel_37 tests:panel_59 tests:panel_39	Yes
...	...	...	...	...	...
279	1438343340 - 1438343640	FP__CloudTracking1__76416d89-9f19-4667-90bc-250e51555d84 #3	tests:panel_94	tests:panel_74 tests:panel_85 tests:panel_96 tests:panel_73 tests:panel_84 tests:panel_95 tests:panel_76 tests:panel_75 tests:panel_86 tests:panel_92 tests:panel_72 tests:panel_83 tests:panel_94 tests:panel_82 tests:panel_93	Yes
281	1438343340 - 1438343640	FP__CloudTracking1__76416d89-9f19-4667-90bc-250e51555d84 #3	tests:panel_85 tests:panel_74 tests:panel_84 tests:panel_95 tests:panel_75 tests:panel_94	tests:panel_67 tests:panel_66 tests:panel_63 tests:panel_85 tests:panel_84 tests:panel_62 tests:panel_65 tests:panel_87 tests:panel_64 tests:panel_86 tests:panel_83 tests:panel_82 tests:panel_56 tests:panel_77 tests:panel_55 tests:panel_57 tests:panel_74 tests:panel_96 tests:panel_52 tests:panel_73 tests:panel_95 tests:panel_76 tests:panel_54 tests:panel_75 tests:panel_97 tests:panel_53 tests:panel_92 tests:panel_94 tests:panel_72 tests:panel_93	Yes
...	...	...	...	...	...

Table C.3.2.: Case 4: Subset of the Area Registration Requests made. First four concerning the beginning of the two Focused Situation Processing Instances #1 & #2. The next 5 concerning the occurring collision between #1 & #2 and the last two concerning the start of a third Focused Situation Processing Instance #3 after the situations separated again.

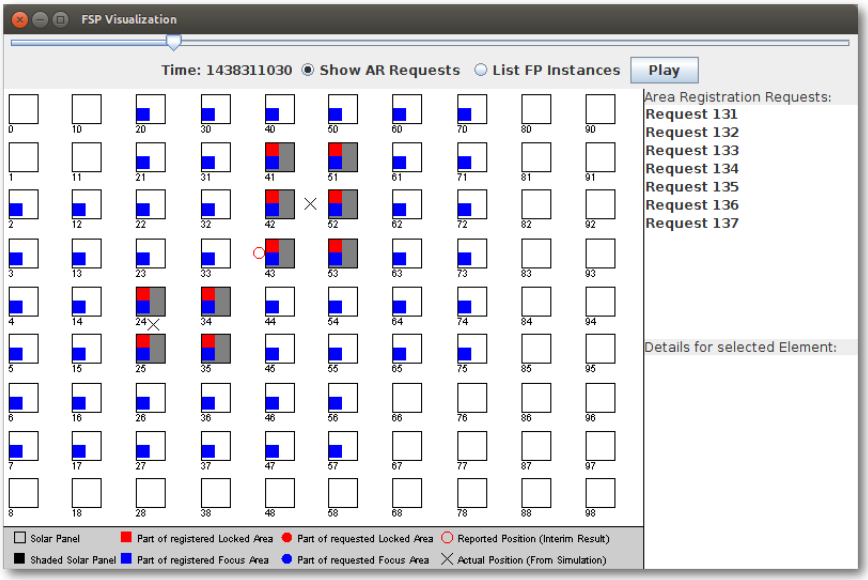


Figure C.3.1.: Case 4: The Collision of two independent situations (clouds) results in the merge of the two FSP Instances #1 and #2 where only instance #1 survives and claims the shaded solar panels of both clouds as its Locked Area.

The iteration processing of both instances is very similar to the iteration processing already discussed for Test Case 1 and is thus only summarized here. The first iteration of each of the two FSP Instances determined that the cloud affected more than the single solar panel set as initial Locked Areas (Table C.3.2 Request 0 & 4) and determined new Locked Areas which covered the whole shaded area for each cloud. The new Locked Areas were successfully acquired by the corresponding FSP Instances (Table C.3.2 Request 2 & 6) and the processing repeated the initial Time Frame during its second iteration.

As for Test Case 1 the third iteration then continued with the next consecutive Time Frame but with the same Locked Area and Focus Area as the previous iteration. This continued until the solar panels affected by the cloud change, while the process is similar to Test Case 1 until the shaded areas of the two clouds get too close to each other, preventing the FSP Instances to keep them separate. The resulting collision and its handling is discussed in the following subsection.

C.3.2. Part 2: Temporary Overlap of Two Situations

The second part of this test case discusses the limitation of the processing model to distinguish situations from each other when their identification criteria (Locked Area for a specific Time Frame) is not unique anymore. In the case discussed here, the ambiguity is the result of a temporary overlap of two situations (clouds) as they cross each other's path in an overlapping time window.

The discussions of this part continues the processing flow discussed in Part 1 starting around the time 1438310880 when the two situations start to collide as shown in Figure C.3.1.

Figure C.3.2.: Case 4: After the shaded areas of the two clouds became separated again, the FSP Instance #1 continued its tracking by following the cloud from the north while the cloud that appeared from the west is *temporarily* not tracked by any FSP Instance.

C.3.2.1. Collision Detection and Handling

A collision was detected as FSP Instance #2 tried to acquire a new Locked Area for the Time Frame 1438310760 to 1438311060 which includes the solar panels 24 and 34 which belong to the shaded area of the cloud tracked by FSP Instance #1. FSP Instance #2 tried to acquire them as they were part of its most recent Focus Area (Table C.3.2 Request 131), and were therefore determined as relevant. As they were close to the shaded area tracked by FSP Instance #2 they were grouped together into one cluster with the tracked cloud during the post iteration processing of FSP Instance #2.

As the Area Registration request collided with the Area Registration of FSP Instance #1 it was rejected (Table C.3.2 Request 132) and FSP Instance #2 was stopped for merging into FSP Instance #1.

Once FSP Instance #1 requested a new Area Registration (Table C.3.2 Request 135) the merge was executed. Afterwards the Area Registration was granted and FSP Instance #1 continued with the next iteration. During this iteration, FSP Instance #1 determined that the nodes 14 and 15 are not shaded anymore but the nodes 52, 53, 42 and 43 were also shaded (by the cloud coming from the north) and belong to the same geographical cluster of nodes. Thus, it updated its Locked Area and Focus Area once again (Table C.3.2 Request 136). The following iteration then determined that the Locked Area was correct and FSP Instance #1 continued with the normal processing flow as already discussed in Part 1.

This continues until the two clouds separate again which is discussed in the next subsection.

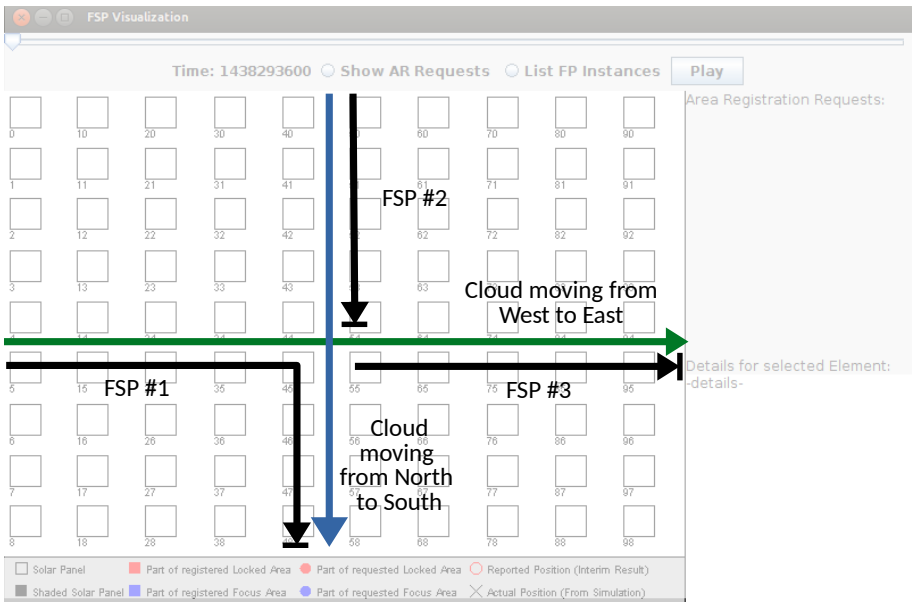


Figure C.3.3.: Case 4: Illustration on the sections of the cloud paths tracked by the three created FSP Instances.

C.3.2.2. Situation Split

As the clouds further moved along their corresponding paths, the shaded area of the two clouds is split into two separate areas. At time 1438338330 these areas were separated enough so that the used clustering mechanism in the post iteration processing of the current FSP Instance #1 did not consider them as a single cluster anymore. (Figure C.3.2) Instead two separate clusters were found during the post processing and FSP Instance #1 selected the biggest of the two clusters to follow.

As shown by the following Area Registration (Table C.3.2 Request 260), the FSP Instance #1 followed the cloud moving from the north to the south while ignoring the shaded panels of the cloud moving from west to east. After this separation the FSP Instance #1 continued with its normal processing flow as discussed during Part 1.

The shaded area split of from FSP Instance #1 continued to move to the east. Due to its change in position it generated additional Possible Situation Indication Events. The first Possible Indication Events raised after the shaded area separated and left the Focus Area of FSP #1 are the events raised for the panels 94 and 95 (Table C.3.1 Events 33 and 34). As the nodes concerned by those events were not part of the Focus Area of FSP Instance #1, they were not assigned anymore to FSP Instance #1 but instead result in the creation of a new FSP Instance #3 as shown in Table C.3.1 for the Possible Situation Indication Event 33.

Further Possible Situation Indication Events (34ff) created by this cloud were then ignored based on the same mechanism that was already discussed for the initial detection of the two clouds.

C.3.3. Case 4 Conclusions

Part 1 of this test case demonstrated the capability of the processing model to be used to track multiple independent situations. Part 2 however demonstrated the limitation of this

process in relying on the uniqueness of the situation identification (Locked Area + Time Frame). This limitation leads to an incorrect tracking of the situations in this case as illustrated in Figure C.3.3. For this specific case, the FSP Instance #1 started by tracking the cloud appearing from the west but due to the temporary situation overlap switched over to tracking the cloud that appeared from the north. FSP Instance #2 started the tracking of the cloud from the north but got terminated due to the merge of the situations, while FSP #1 continued the tracking. Once the clouds separated, a new completely independent FSP Instance #3 was started which continued the tracking of the cloud that appeared from the west and was initially tracked by FSP #1. As such the test case illustrated that the temporary loss of a situation's unique identity results in incorrect results as the processing model relies on these identities.

The results from the three test cases discussed in this chapter are summarized as part of the conclusion of the overall evaluation in Section 7.10.

Bibliography

[AAB+05] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Mitch Cherniack, Jeong hyon Hwang, Wolfgang Lindner, Anurag S. Maskey, Er Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. The Design of the Borealis Stream Processing Engine. In *In CIDR*, pages 277–289, 2005. (Referenced on pages 4, 32, 40, and 45)

[ABB+04] Arvind Arasu, Brian Babcock, Shivnath Babu, John Cieslewicz, Keith Ito, Rajeev Motwani, Utkarsh Srivastava, and Jennifer Widom. STREAM: The stanford data stream management system. Technical Report 2004-10, Stanford InfoLab, 2004. (Referenced on pages 32 and 38)

[ABB+13] Tyler Akidau, Alex Balikov, Kaya Bekirouglu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. MillWheel: Fault-tolerant Stream Processing at Internet Scale. *Proc. VLDB Endow.*, 6(11):1033–1044, aug 2013. (Referenced on page 36)

[ABM10] David Alves, Pedro Bizarro, and Paulo Marques. Flood: Elastic Streaming MapReduce. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS 10, pages 113–114, New York, NY, USA, 2010. ACM. (Referenced on page 33)

[ABQ13] Leonardo Aniello, Roberto Baldoni, and Leonardo Querzoni. Adaptive Online Scheduling in Storm. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, pages 207–218. ACM, 2013. (Referenced on page 37)

[ABW06] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *The VLDB Journal*, 15(2):121–142, jun 2006. (Referenced on pages 38, 41, and 42)

[ACC+03a] Daniel Abadi, Donald Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, C. Erwin, Eduardo Galvez, M. Hatoun, Anurag Maskey, and Alex Rasin. Aurora: a data stream management system. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003. (Referenced on page 39)

[ACC+03b] Daniel J. Abadi, Don Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal - The International Journal on Very Large Data Bases*, 12(2):120–139, 2003. (Referenced on pages 32, 39, and 43)

[AFRS11] Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*, pages 635–644, 2011. (Referenced on page 44)

[AH00] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously Adaptive Query Processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 261–272, New York, NY, USA, 2000. ACM. (Referenced on page 45)

[All83] James F. Allen. Maintaining Knowledge About Temporal Intervals. *Commun. ACM*, 26(11):832–843, nov 1983. (Referenced on page 60)

[Amaa] Amazon Kinesis. Online: http://aws.amazon.com/de/kinesis/. Visited: 28.12.16. (Referenced on page 37)

[Amab] Amazon Simple Notification Service (SNS). Online: https://aws.amazon.com/de/sns/. Visited: 20.01.15. (Referenced on page 35)

[Ant] ANTLR Parser Generator. Online: http://www.antlr.org/. Visited: 08.06.16. (Referenced on page 144)

[Apaa] Apache ActiveMQ. Online: http://activemq.apache.org. Visited: 30.12.16. (Referenced on page 34)

[Apab] Apache ActiveMQ: What is the Prefetch Limit For? Online: http://activemq.apache.org/what-is-the-prefetch-limit-for.html. Visited: 28.12.16. (Referenced on page 31)

[Apac] Apache Camel. Online: http://camel.apache.org. Visited: 30.12.16. (Referenced on page 34)

[Apad] Apache Camel: SEDA Component. Online: http://camel.apache.org/seda.html. Visited: 20.12.16. (Referenced on page 31)

[Apae] Apache Jena. Online: http://jena.apache.org/. Visited: 17.02.2015. (Referenced on page 44)

[Apaf] Apache Samza. Online: http://samza.apache.org/. Visited: 28.12.16. (Referenced on page 36)

[Apag] Apache ServiceMix 1.0-M1 Release. Online: http://servicemix.apache.org/downloads/servicemix-1.0-m1-release.html. Visited: 28.12.16. (Referenced on page 31)

[Apah] Apache Spark Streaming. Online: https://spark.apache.org/streaming/. Visited: 28.12.16. (Referenced on page 36)

[Apai] Apache Zookeeper. Online: http://zookeeper.apache.org/. Visited: 13.07.14. (Referenced on pages 36 and 160)

[Apaj] S4: Distributed Stream Computing Platform. Online: http://incubator.apache.org/s4. Visited: 19.07.13. (Referenced on pages 36 and 37)

[Bab05] Shivnath Babu. *Adaptive Query Processing in Data Stream Management Systems*. PhD thesis, Stanford, CA, USA, 2005. AAI3187262. (Referenced on page 45)

[BB05] Shivnath Babu and Pedro Bizarro. Adaptive query processing in the looking glass. In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR)*, 2005. (Referenced on page 46)

[BBC+09] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *Proceedings of the 18th international conference on World wide web*, pages 1061–1062, 2009. (Referenced on page 44)

[BBL11] David Beckett and Tim Berners-Lee. Turtle - Terse RDF Triple Language. Technical report, March 2011. Online: https://www.w3.org/TeamSubmission/2011/SUBM-turtle-20110328/. Visited: 08.06.16. (Referenced on page 145)

[BD10] Ralf Bruns and Jürgen Dunkel. *Event-Driven Architecture*. Springer Berlin Heidelberg, 2010. (Referenced on pages 30 and 31)

[BGJ08] Andre Bolles, Marco Grawunder, and Jonas Jacobi. Streaming SPARQL - Extending SPARQL to Process Data Streams. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications*, number 5021, pages 448–462. Springer Berlin Heidelberg, jan 2008. (Referenced on page 44)

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996. (Referenced on page 34)

[Bor] Borealis Distributed Stream Processing Engine. Online: http://cs.brown.edu/research/borealis/public/. Visited: 20.07.14. (Referenced on page 40)

[BV07] Thomas Bernhardt and Alexandre Vasseur. Esper: Event Stream Processing and Correlation. Online: http://www.onjava.com/pub/a/onjava/2007/03/07/esper-event-stream-processing-and-correlation.html, 03 2007. Visited: 30.06.14. (Referenced on page 38)

[BW04] Shivnath Babu and Jennifer Widom. StreaMon: An Adaptive Engine for Stream Query Processing. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 931–932, New York, NY, USA, 2004. ACM. (Referenced on page 45)

[CBB+03] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Don Carney, Uğur Çetintemel, Ying Xing, and Stan Zdonik. Scalable distributed stream processing. In *In CIDR*, 2003. (Referenced on pages 40, 43, and 45)

[CCD+03a] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR*, 2003. (Referenced on page 38)

[CCD+03b] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Fred Reiss, and Mehul A. Shah. TelegraphCQ: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668, 2003. (Referenced on pages 32 and 38)

[CCG10] Jean-Paul Calbimonte, Oscar Corcho, and Alasdair J. G. Gray. Enabling ontology-based access to streaming data sources. In *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I*, ISWC'10, pages 96–111, Berlin, Heidelberg, 2010. Springer-Verlag. (Referenced on page 44)

[CDNF01] G. Cugola, Elisabetta Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *Software Engineering, IEEE Transactions on*, 27(9):827–850, Sep 2001. (Referenced on page 35)

[CDTW00] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 379–390, New York, NY, USA, 2000. ACM. (Referenced on page 41)

[CEF+04] Owen Cooper, Anil Edakkunni, Michael J. Franklin, Wei Hong, Shawn R. Jeffery, Sailesh Krishnamurthy, Fredrick Reiss, Shariq Rizvi, and Eugene Wu. HiFi: A Unified Architecture for High Fan-in Systems. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 1357–1360. VLDB Endowment, 2004. (Referenced on page 41)

[CEvA11] Mani K. Chandy, Opher Etzion, and Rainer von Ammon, editors. *The Event Processing Manifesto*, volume 10201 of *Dagstuhl Seminar Proceedings*, 2011. (Referenced on pages 1, 29, and 32)

[CJ09] Sharma Chakravarthy and Qingchun Jiang. *Stream data processing: a quality of service perspective: modeling, scheduling, load shedding, and complex event processing*, volume 36. Springer Science & Business Media, 2009. (Referenced on page 30)

[CM12] Gianpaolo Cugola and Alessandro Margara. Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Comput. Surv.*, 44(3):15–1, jun 2012. (Referenced on page 32)

[Con96] The ACT-NET Consortium. The Active Database Management System Manifesto: A Rulebase of ADBMS Features. *ACM SIGMOD Record*, 25(3):414–471, sep 1996. (Referenced on page 30)

[CRW01] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and Evaluation of a Wide-area Event Notification Service. *ACM Trans. Comput. Syst.*, 19(3):332–383, aug 2001. (Referenced on page 35)

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference and symposium on operating systems design and implementation*. USENIX Association, 2004. (Referenced on page 33)

[DL11] W. Roy Schulte David Luckham. Event Processing Technical Society: Event Processing Glossary 2.0, July 2011. (Referenced on pages 29, 30, and 32)

[dro] Drools Business Rules Management System. Online: http://www.drools.org/. Visited: 20.12.16. (Referenced on pages 39 and 144)

[Dun09] J. Dunkel. On complex event processing for sensor networks. In *Autonomous Decentralized Systems, 2009. ISADS '09. International Symposium on*, march 2009. (Referenced on page 30)

[EFGK03] Patrick Th Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003. (Referenced on page 35)

[EN11] Opher Etzion and Peter Niblett. *Event Processing in Action*. Manning, Greenwich, 74° w. long., 2011. (Referenced on page 32)

[Esp] Esper - Complex Event Processing. Online: http://esper.codehaus.org/. Visited: 10.06.14. (Referenced on page 38)

[GAW+08] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. SPADE: The System s Declarative Stream Processing Engine. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1123–1134, New York, NY, USA, 2008. ACM. (Referenced on page 44)

[GJPPM+12] V. Gulisano, R. Jiménez-Peris, M. Patiño-Martínez, C. Soriente, and P. Valduriez. StreamCloud: An Elastic and Scalable Data Streaming System. *Parallel and Distributed Systems, IEEE Transactions on*, 23(12):2351–2365, Dec 2012. (Referenced on page 41)

[GJPPMV10] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, and P. Valduriez. StreamCloud: A Large Scale Data Streaming System. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pages 126–137, june 2010. (Referenced on pages 37, 41, and 45)

[GKBF98] Stella Gatziu, Arne Koschel, Günter Bültzingsloewen, and Hans Fritschi. Unbundling active functionality. *SIGMOD Rec.*, 27(1):35–40, 1998. (Referenced on page 31)

[GKRS15] Mahbouba Gharbi, Arne Koschel, Andreas Rausch, and Gernot Starke. *Basiswissen für Softwarearchitekten*. dpunkt.verlag, 2 edition, 2015. (Referenced on page 139)

[GKS14] Mario Golling, Robert Koch, and Lars Stiemert. Architektur zur mehrstufigen Angriffserkennung in Hochgeschwindigkeits-Backbone-Netzen. *DFN- Forum Kommunikationstechnologien, ser. LNI, Gesellschaft für Informatik (GI)*, 2014. (Referenced on page 48)

[Gro13] The W3C SPARQL Working Group. SPARQL 1.1 Overview. Technical report, march 2013. Online: https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/. Visited: 20.11.16. (Referenced on pages 111 and 112)

[Gul12] Vincenzo Massimiliano Gulisano. *StreamCloud: An Elastic Parallel-Distributed Stream Processing Engine*. PhD thesis, Universidad Politécnica de Madrid, December 2012. (Referenced on page 41)

[HAG+09] Martin Hirzel, Henrique Andrade, Bugra Gedik, Vibhore Kumar, Giuliano Losa, Mark Mendell, Howard Nasgaard, Robert Soulé, and Kun-Lung Wu. SPL Stream Processing Language Specification. Technical report, IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, November 2009. (Referenced on page 44)

[HFC+00] Joseph M. Hellerstein, Michael J. Franklin, Sirish Chandrasekaran, Amol Deshpande, Kris Hildrum, Samuel Madden, Vijayshankar Raman, and Mehul A. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Eng. Bull.*, 23(2):7–18, 2000. (Referenced on page 46)

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004. (Referenced on page 9)

[HSS+14] Martin Hirzel, Robert Soulé, Scott Schneider, Bugra Gedik, and Robert Grimm. A Catalog of Stream Processing Optimizations. *ACM Comput. Surv.*, 46(4):46–1, mar 2014. (Referenced on pages 4 and 45)

[HW12] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2012. (Referenced on pages 33 and 34)

[KCC+03] Sailesh Krishnamurthy, Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Samuel Madden, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: An architectural status report. *IEEE Data Eng. Bull.*, 26(1):11–18, 2003. (Referenced on page 38)

[KHI11] Holger Knublauch, James A. Hendler, and Kingsley Idehen. SPIN - Overview and Motivation. Online: http://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/, feb 2011. Visited: 13.07.14. (Referenced on page 44)

[KK98] A. Koschel and R. Kramer. Configurable event triggered services for CORBA-based systems. In *Enterprise Distributed Object Computing Workshop, 1998. EDOC '98. Proceedings. Second International*, pages 306–318, nov 1998. (Referenced on page 31)

[KKP11] Wilhelm Kleiminger, Evangelia Kalyvianaki, and Peter Pietzuch. Balancing load in stream processing with the cloud. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering Workshops*, ICDEW '11, pages 16–21, Washington, DC, USA, 2011. IEEE Computer Society. (Referenced on pages 37 and 45)

[KL98] Arne Koschel and Peter C. Lockemann. Distributed events in active database systems: letting the genie out of the bottle . *Data Knowl. Eng.*, 25(1-2):11–28, 1998. (Referenced on page 31)

[KORR12] Boris Koldehofe, Beate Ottenwälder, Kurt Rothermel, and Umakishore Ramachandran. Moving Range Queries in Distributed Complex Event Processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS 12, pages 201–212, New York, NY, USA, 2012. ACM. (Referenced on page 47)

[Kos99] Arne Koschel. *Ereignisgetriebene CORBA-Dienste für heterogene, verteilte Informationssysteme*. PhD thesis, Forschungszentrum Informatik, Karlsruhe (FZI), 1999. (Referenced on page 31)

[Krä07] Jürgen Krämer. *Continuous Queries over Data Streams – Semantics and Implementation*. PhD thesis, Philipps-Universität Marburg , 2007. (Referenced on page 41)

[KS04] Jürgen Krämer and Bernhard Seeger. PIPES – A Public Infrastructure for Processing and Exploring Streams. 2004. (Referenced on page 41)

[LC08] Z. Laliwala and S. Chaudhary. Event-driven Service-Oriented Architecture. In *Service Systems and Service Management, 2008 International Conference on*, pages 1–6, June 2008. (Referenced on page 31)

[LLP+12] Wang Lam, Lu Liu, S. T. S. Prasad, Anand Rajaraman, Zoheb Vacheri, and AnHai Doan. Muppet: MapReduce-Style Processing of Fast Data. *CoRR*, abs/1208.4175, 2012. (Referenced on page 36)

[LRD+11] Mo Liu, E. Rundensteiner, D. Dougherty, C. Gupta, Song Wang, I. Ari, and A. Mehta. High-performance nested CEP query processing over event streams. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 123–134, April 2011. (Referenced on page 44)

[Luc] David Luckham. SOA, EDA, BPM and CEP are all Complementaty. Online: http://complexevents.com/wp-content/uploads/2007/05/SOA_EDA_Part_1.pdf. Visited: 27.03.15. (Referenced on page 31)

[Luc96] David C. Luckham. Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Orderings of Events. Technical report, Stanford, CA, USA, 1996. (Referenced on page 44)

[Luc01] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. (Referenced on pages 30, 31, and 44)

[Luc07a] David Luckham. A Short History of Complex Event Processing Part 2: the rise of CEP. Online: http://complexevents.com/wp-content/uploads/2008/07/2-final-a-short-history-of-cep-part-2.pdf. Visited: 20.11.16, 2007. (Referenced on page 31)

[Luc07b] David C. Luckham. A short history of Complex Event Processing. part 1: Beginnings. Online: http://complexevents.com/wp-content/uploads/2008/02/1-a-short-history-of-cep-part-1.pdf. Visited: 20.11.16, 2007. (Referenced on page 30)

[LY08] Dionysios Logothetis and Kenneth Yocum. Ad-hoc data processing in the cloud. *Proc. VLDB Endow.*, 1(2):1472–1475, aug 2008. (Referenced on page 33)

[Mar] Nathan Marz. A Storm is coming: more details and plans for release. Online: http://engineering.twitter.com/2011/08/storm-is-coming-more-details-and-plans.html. Visited: 19.07.12. (Referenced on page 36)

[Mar06] Jean-Louis Maréchaux. Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus. Technical report, IBM, March 2006. Online: http://www.ibm.com/developerworks/library/ws-soa-eda-esb/. Visited: 25.02.15. (Referenced on page 31)

[MCT14] Alessandro Margara, Gianpaolo Cugola, and Giordano Tamburrelli. Learning from the past: Automated rule generation for complex event processing. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, pages 47–58, New York, NY, USA, 2014. ACM. (Referenced on page 44)

[MFP06] Gero Mühl, Ludger Fiege, and Peter Pietzuch. *Distributed Event-Based Systems*. Springer Berlin / Heidelberg, 2006. (Referenced on pages 30 and 35)

[MOB+08] Y. Magid, D. Oren, D. Botzer, A. Adi, B. Shulman, E. Rabinovich, and M. Barnea. Generating real-time complex event-processing applications. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, volume 47, pages 251–263, July 2008. (Referenced on page 44)

[MSHR02] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously Adaptive Continuous Queries over Streams. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 49–60, New York, NY, USA, 2002. ACM. (Referenced on page 45)

[Mul14] MULE ESB Enterprise Performance. Technical report, MuleSoft, 2014. Online: http://www.mulesoft.com/downloads/mule-performance-metrics.pdf. Visited: 10.12.14. (Referenced on page 31)

[MVE] MVEL Language Guide. Online: https://en.wikisource.org/w/index.php?title=MVEL_Language_Guide&oldid=6036377. Visited: 10.02.16. (Referenced on pages 111 and 113)

[nis12] NIST Framework and Roadmap for Smart Grid Interoperability Standards. Technical Report 1108R2, feb 2012. (Referenced on page 13)

[NRNK10] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed Stream Computing Platform. In *ICDM Workshops*, pages 170–177, 2010. (Referenced on page 37)

[OKR+14a] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, David Lillethun, and Umakishore Ramachandran. MCEP: A Mobility-Aware Complex Event Processing System. *ACM Trans. Internet Technol.*, 14(1):6–1, aug 2014. (Referenced on page 47)

[OKR+14b] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, and Umakishore Ramachandran. RECEP: Selection-based Reuse for Distributed Complex Event Processing. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS 14, pages 59–70, New York, NY, USA, 2014. ACM. (Referenced on page 47)

[PGS+10] Helge Parzyjegla, Daniel Graff, Arnd Schröter, Jan Richling, and Gero Mühl. Design and Implementation of the Rebeca Publish/Subscribe Middleware. chapter From Active Data Management to Event-based Systems and More, pages 124–140. Springer-Verlag, Berlin, Heidelberg, 2010. (Referenced on page 35)

[PSPP14] Om Prasad Patri, Vikrambhai S. Sorathia, Anand V. Panangadan, and Viktor K. Prasanna. The Process-oriented Event Model (PoEM): A Conceptual Model for Industrial Events. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS 14, pages 154–165, New York, NY, USA, 2014. ACM. (Referenced on page 46)

[PVAM12] Adrian Paschke, Paul Vincent, Alex Alves, and Catherine Moxey. Tutorial on Advanced Design Patterns in Event Processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS 12, pages 324–334, New York, NY, USA, 2012. ACM. (Referenced on pages 7 and 8)

[PVM+12] Adrian Paschke, Paul Vincent, Catherine Moxey, Martin Hirzel, and Alex Alves. Event Processing Reference Architecture - Design Patterns. Online: http://de.slideshare.net/isvana/epts-debs2012-event-processing-reference-architecture-design-patterns-v204b, 2012. Visited: 2014.11.20. (Referenced on page 8)

[rdf] Eclipse RDF4J Homepage. Online: http://rdf4j.org/. Visited: 10.06.16. (Referenced on page 145)

[RDS+04] Elke A. Rundensteiner, Luping Ding, Timothy Sutherland, Yali Zhu, Brad Pielech, and Nishant Mehta. CAPE: Continuous Query Engine with Heterogeneous-grained Adaptivity. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 1353–1356. VLDB Endowment, 2004. (Referenced on page 45)

[red] Apache Redis Homepage. Online: http://redis.io/. Visited: 11.07.16. (Referenced on page 160)

[SAG+09] S. Schneider, H. Andrade, B. Gedik, A. Biem, and Kun-Lung Wu. Elastic scaling of data parallel operators in stream processing. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–12, may 2009. (Referenced on pages 37 and 45)

[SAKG14] Marc Schaaf, Irina Astrova, Arne Koschel, and Stella Gatziu Grivas. The om4space activity service: A semantically well-defined cloud-based event notification middleware. *International Journal On Advances in Software*, 7:697 to 709, 2014. (Referenced on page 35)

[SBC+98] Robert Strom, Guruduth Banavar, Tushar Chandra, Marc Kaplan, Kevan Miller, Bodhi Mukherjee, Daniel Sturman, and Michael Ward. Gryphon: An information flow based approach to message brokering. 1998. (Referenced on page 35)

[SGA+13] Robert Soulé, Michael I. Gordon, Saman Amarasinghe, Robert Grimm, and Martin Hirzel. Dynamic Expressivity with Static Optimization for Streaming Languages. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS 13, pages 159–170, New York, NY, USA, 2013. ACM. (Referenced on page 47)

[SGLN+11] Sriskandarajah Suhothayan, Kasun Gajasinghe, Isuru Loku Narangoda, Subash Chaturanga, Srinath Perera, and Vishaka Nanayakkara. Siddhi: A Second Look at Complex Event Processing Architectures. In *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments*, GCE '11, pages 43–50, New York, NY, USA, 2011. ACM. (Referenced on page 44)

[SHCF03] M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, and M. J. Franklin. Flux: an adaptive partitioning operator for continuous query systems. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 25–36, March 2003. (Referenced on page 45)

[SLJR05] Timothy M. Sutherland, Bin Liu, Mariana Jbantova, and Elke A. Rundensteiner. D-CAPE: Distributed and Self-tuned Continuous Query Processing. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pages 217–218, New York, NY, USA, 2005. ACM. (Referenced on page 45)

[Sof] TIBCO Software. StreamSQL Guide. Online: https://docs.tibco.com/pub/stibco_StreamSQLtreambase_cep/7.3.6_aug_2013/pdf/streamsql.pdf. Accessed: 10.12.14. (Referenced on page 44)

[sto] Storm, distributed and fault-tolerant realtime computation. Online: http://storm-project.net/. Visited: 04.06.13. (Referenced on page 36)

[SZS+03] Stan Zdonik Sbz, Stan Zdonik, Michael Stonebraker, Mitch Cherniack, Ugur C. Etintemel, Magdalena Balazinska, and Hari Balakrishnan. The Aurora and Medusa Projects. *IEEE Data Engineering Bulletin*, 26, 2003. (Referenced on page 40)

[Tel] The Telegraph Project at UC Berkeley. Online: http://telegraph.cs.berkeley.edu. Visited: 28.12.16. (Referenced on page 38)

[W3C04] SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Online: http://www.w3.org/Submission/SWRL/, may 2004. Visited: 13.07.14. (Referenced on page 43)

[WDR06] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance Complex Event Processing over Streams. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 407–418, New York, NY, USA, 2006. ACM. (Referenced on page 44)

[Wel02] Matthew David Welsh. *An architecture for highly concurrent, well-conditioned internet services*. PhD thesis, University of California, 2002. (Referenced on page 31)

[WG96] Klaus Peter Wershofen and Volker Graefe. Situationserkennung als Grundlage der Verhaltenssteuerung eines mobilen Roboters. 1996. (Referenced on page 48)

[WS09a] Kerstin Werner and Alexander Schill. An Event-processing Architecture for an RFID based Logistics Monitoring System. In *RFID Systems and Technologies (RFID SysTech), 2009 5th European Workshop on*, pages 1–8, June 2009. (Referenced on page 30)

[WS09b] G. Wishnie and H. Saiedian. A Complex Event Routing Infrastructure for Distributed Systems. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference, COMPSAC '09*, volume 2, pages 92–95, July 2009. (Referenced on page 35)

[WSB+14] Gwendolin Wilke, Marc Schaaf, Erik Bunn, Topi Mikkola, Remo Ryter, Holger Wache, and Stella Gatziu Grivas. Intelligent dynamic load management based on solar panel monitoring. In *Proceedings of the 3rd Conference on Smart Grids and Green IT Systems*, pages 76–81, 2014. (Referenced on page 2)

[WSGL11] Daniela Wolff, Marc Schaaf, Stella Gatziu Grivas, and Uwe Leimstoll. Context-aware website personalization. In *Proceedings of Knowlege-Based and Intelligent Information and Engineering Systems*, pages 51–62. Springer, 2011. (Referenced on page 30)

[WY10] Yongheng Wang and Shenghong Yang. High-performance complex event processing for large-scale RFID applications. In *Proceedings of the 2nd International Conference on Signal Processing Systems (ICSPS)*, July 2010. (Referenced on page 40)

[XZH05] Ying Xing, Stan Zdonik, and Jeong-Hyon Hwang. Dynamic Load Distribution in the Borealis Stream Processor. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 791–802, Washington, DC, USA, 2005. IEEE Computer Society. (Referenced on pages 40 and 45)

[YG02] Yong Yao and Johannes Gehrke. The Cougar Approach to In-network Query Processing in Sensor Networks. *SIGMOD Rec.*, 31(3):9–18, sep 2002. (Referenced on page 41)

[YKPS07] Yin Yang, J. Kramer, D. Papadias, and B. Seeger. HybMig: A Hybrid Approach to Dynamic Plan Migration for Continuous Queries. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):398–411, March 2007. (Referenced on page 4)

[ZRH04] Yali Zhu, Elke A. Rundensteiner, and George T. Heineman. Dynamic Plan Migration for Continuous Queries over Data Streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 431–442, New York, 2004. ACM. (Referenced on page 45)

List of Figures

List of Tables

List of Definitions