5. Proof of concept by case study

5.1 Case study

5.1.1 Background

Nedap - Identification Systems

[TODO]

SENSIT [smart parking] application

The SENSIT Nedap Identification Services smart parking application is devised to monitor parking lots and garages. It employ a huge amount (up to thousands) of affordable LPWA sensor nodes. Each individual parking spot is equipped with one of these sensors to determine its occupation. To determine changes in occupation, each sensor is equipped with an infra-red and magnetic induction sensor. Should a change in occupation be detected, a message containing the measured sensor deltas is sent to the back-end application. This granular approach to smart parking allows the SENSIT application to monitor and visualise the occupation of individual parking spaces in a lot, garage or even across cities.

In order to communicate with the back-end the sensors employ wireless technology. Previously, the sensors were connected to sinks using a proprietary network of relay nodes. However the recent proliforation of large scale cellular IoT networks has caused Nedap to shift towards these technologies. This allows large numbers of sensors to a single cell tower, without the need of deploying and managing a network of relay nodes for new sensor deployments. Additionally the effort in managing and maintianing the network is outsource to professional operators. To connect the sensors to the internet, *Narrow-band Internet of Things* technology was determined to be most suitable. New SENSIT sensors are therefore equipped with u-blox [**?**] NB-IoT radio modules to connect them to operated cell networks.

5.1.2 Context of the Case Study

In this section we will describe and scope the context of the QoS monitoring application to be developed. We will first describe the input for the application in terms of sensor data emitted by the WSN application under investigation. Consequently, the characteristics of the outcomes of the application to be prototyped will be discussed.

Sensor data signature

The sensor devices send a message with key point information (KPI) data with every data message it sends. Alternatively it will send one of these messages periodically if no data messages are sent for [time period]. This entails that a typical sensor sends between 10 and 50 KPI information messages on average per day, with some outliers for more active sensors which can reach up to 250 messages per day on a regular basis.

The data sent by the sensor contains some typical networking data points, such as source IP address, source port, source device ID, message sequence number and a timestamp. Additionally the message contains a hexadecimally encoded string describing the KPI's collected by the u-blox radio module. The data collected by the u-blox module contains mostly data points depicting the signalling functions of the radio module. Such KPI's include the signal-to-noise ratio, signal quality (RSSI), Extended Coverage Level (ECL) and more. Additionally the KPI information includes some physical attributes of the radio module. Attributes such as the module's uptime, number of restarts and temperature.

The ordinary data plus the u-blox KPI data are contains within [512] bytes of data (1/2 KiB). Considering the messaging rate of a typical sensor we yield an imposed per sensor footprint on bandwidth of 5-25 KiB/day for the majority of sensors, with outliers of 125 KiB/day for extremely active sensors.

At this moment only a few nodes equipped with the NB-IoT technology have been deployed. Therefore a large scale test bed for the to be prototyped monitoring application does not exist. Therefore a simulated sensor environment has been devised to test the prototype application for contemporary and near-future smart parking applications. This simulation is based on data signatures and values observed over a half year period emitted by the few nodes that have been deployed.

QoS monitoring needs

In collaboration with Nedap Identification Services a list of requirements for the outcomes of the prototype was compiled. These consequences are to be effected by the prototype application, based on input from (simulated) sensors. However, the actual implementation of the prototype is secondary to this chapter, since the primary goal is to evaluate choices made for the underlying development platform. Therefore a comprehensive, formalized requirements document has not been included in this thesis. We will however shortly describe the features required of the monitoring application to be developed in order to contextualize the implementation efforts of the prototype.

The consequences the application must effect are classified into three categories. The first is sensor feedback. This entails commands sent to sensors to alter its execution strategy, based on observations made in the monitoring application. This can be based on individual sensor data, historic sensor data or higher level data snapshots (e.g. sink level). An example of such feedbacks are to decrease data rates to guarantee a predetermined minimum sensor lifetime or due to poor cell connectivity. This functionality is currently not present in the Nedap sensors, but is intended in the future. Therefore it will be implemented into the simulation environment to test the command & control capabilities of

the platform.

The second type of effect to be caused by the application is instant alerting. The primary use case for this kind of consequence is when physical maintenance is imminently required in the application or its network. Detectable causes of when this might be warranted have been deliberated with Nedap Identification Services and examples include:

- a long term drop in coverage level which might indicate permanent obstruction of signal
- extremely high temperature readings indicating an electrical malfunction
- unusually long periods of inactivity or, conversely, extreme data bursts indicate a rouge node not executing according to a correct strategy.
- calculations estimating node lifetime determining a node needs replacing.

The last type of consequence is reporting. The goal of this is to inform technicians, managers or clients on the general operation of the WSN application. This comprises two types of reporting. The first is *periodical reporting*. Periodical reporting will primarily focus on business goals such as long term performance metrics, compliance to service level agreements of both service providers and clients, and prospected short-term maintenance costs. The other type of reporting is *real-time reporting*. This is useful to technicians monitoring the performance of an application during its runtime. Use cases include monitoring the number of incoming events, latencies of sensor devices and sinks, environmental conditions (such as weather and temperature) and which sensor strategies currently are deployed. Notice that the real-time aspect of this type of reporting does not require events to be reported instantaneously since for such statistics a per second or minute update suffices.

5.2 Validation methodology

With the application, case and its context clear, we will turn our focus to detailing the validation study. Before executing our validation study, in this section we will first depict the taken process. We will begin by clearly stating the claims we aim to confirm and the bounds of our scope. Following that we will describe the intended method of testing those claims specifically, by detailing the quantified criteria the platform implementation must adhere to. We must note that these criteria will only cover the scope of the validation study, not the functional requirements of the implementation for the case. As mentioned before, though important for the outcome of the product for the company, for this validation study they are ancillary.

With the goals clearly stated, parametrized and quantified, we will design and implement a prototype monitoring application built upon our developed software platform, tailored to the QoS monitoring needs of Nedap Identification Services . As mentioned before the actual implementation details are secondary for validation purposes of this chapter. Therefore we will only touch upon it shortly without going into great detail. We will however give a short summary of the developed prototype to provide a context to the validation efforts. During and after the development process we will measure the relevant parameters required to evaluate the determined validation criteria.

We will conclude this chapter by stating, analysing and deliberating the parameters obtained by measuring and observing the development process, and results of the execution of different test scenarios. These results will be compared with the previously determined criteria of the validation study. If these criteria are met, this will validate the claims they are meant to affirm. We will conclude by discussing the process and results in order to deliberate the limitations and lessons learned regarding the proposed development platform.

5.3 Criteria of the Case Study

5.3.1 Claims

In this section we will state the claims regarding the proposed platform we aim to validate. These claims will be closely related to some of the research questions stated in Section 1.3.

the first claim regarding our platform is that the appropriate level of abstraction was chosen. This implies an adequate trade-off between the ease of implementation of the platform and the flexibility of its components. We claim that we have chosen our level of abstraction in such a manner that our collection of components can be adapted to suit a plethora of purposes and target applications. Conversely, the level of abstraction is not that high-level that every implementation requires unnecessarily large development efforts because similar procedures require repeated implementation. This claim mirrors the research question RQ3, which asks "What is the appropriate level of abstraction for a WSN monitoring platform [...]".

The second claim that requires validation is regarding the scalability of the platform. As mentioned numerous times before the extreme scale of WSN applications requires (auxiliary) back-end processes that are at least as scalable as the application they observe, as is captivated in research question RQ5. Our claim is that our platform offers the tools to design a fully scalable WSN monitoring application. In order to validate the scalability of the platform and its implementation, possible congestion points will need to be identified and stress-tested in order to show that proper configuration of the component(s) will alleviate any scalability issue. This will be validated by means of two methods, which will be detailed in section 5.4.

5.3.2 Bounds

Before considering into how we aim to validate the stated claims, the bounds and limitations of this validation study will need to be stated. The first glaring limitation of this study is that it is extremely limited in scope. The platform will only be implemented for a specific WSN application and this study will therefore not state the platform to be appropriate for the entire set of WSN applications that was determined in Section 2.1 of Chapter 2. Instead, this study will at most affirm the platform as a proof-of-concept for WSN application QoS monitoring.

The second limitation worthy of notion is that,aside from only regarding a single WSN application, it will also run on a simulation of that application. As mentioned before, this is because the NB-IoT incorporated sensor devices of the SENSIT application have only recently started deployment. As a consequence

a test bed of significant scale is presently not available. However by simulating a full future deployment of the application we are able to easily adapt the application under investigation, in terms of both scale and functionality, which allows us to not only test for intended regular behaviour but also for extreme and niche conditions. Additionally our simulated environment allows for easy temporal manipulation, which enables us to speed up, halt and repeat simulations.

5.4 Method

5.4.1 General approach

As stated before, the first claim to validate is weather the level of platform and model abstraction was appropriate. In order to validate this claim we will quantify the development effort required to adapt the designed platform to the case. From a business perspective, the most interesting parameter to express the adoption effort is the time required to develop an application based on the proposed technology (for example measured in FTE·weeks). However this parameter is extremely subjective as it heavily depends on the level of skill of the developer and its familiarity with the technology. We will therefore not only measure the time required but also the number of lines of code required to devise a monitoring application built by integration of our platform.

Our method of confirming the scalability of the platform is twofold. First, we will flood the system with events. If our claim of scalability is correct this will not cause a build-up of message anywhere in the topology of the application. Should such a congestion occur this should be able to be alleviated by scaling the deployment configuration of the components, i.e. the number of tasks and workers per component, without requiring a change in the topology or the internals of the components themselves. The second method we will employ to test the scalability of the system is by initially configuring the simulation in for real-world deployment. We will then deliberately trigger an event shower in the sensor simulation. It is expected that the platform will experience a sudden influx of input messages. Should such an event occur, the platform is not expected to hold its ordinary timing constraints. However, the platform is expected to eventually return to its normal execution, i.e. within the bounds typical of ordinary execution. The platform will pass this test if it is able to process the batch of messages caused by the sudden influx and return to ordinary execution within a certain amount of time.

5.4.2 Validation criteria

Before starting the implementation, the criteria the monitoring application and its development process must adhere to must be stated. Fulfilment of these criteria affirms the belief in the claims stated in Section 5.3.1. The criteria will be divided into functional requirements and non-functional requirements. The functional requirements describe features and conditions that must hold for the developed monitoring application, but cannot be quantified or measured. It either holds or it does not. The non-functional requirements however are quantified and measurable.

Again we re-iterate, these criteria and requirements only relate to the validation study, not the requirements of the actual monitoring application prototype that will be designed and developed. Reason for this is that the aim is to evaluate the development platform, not this particular instance of the platform.

Functional requirements

Intuitively, the primary criterium is that an instantiation of the proposed platform should be possible in accordance with the needs and wants of Nedap Identification Services . This seems an obvious and trivial demand, but without stating it, any subsequent criterium is pointless. More specifically, the platform should enable an instantiation which enables iterative and consequent enrichment and accumulation of information. At multiple stages of the consequential iteration the application should be able to generate outputs such as alerts and reports for auxiliary processes and systems.

Non-functional requirements

Though the platform should enable an instantiation according to the needs of Nedap Identification Services , it should do so with minimal development effort. We will express these efforts in the time needed for the implementation and the lines of code required.

Aside from the usability criterium we find the scalability required of the platform. We will formalize this requirement with two criteria. The first regards the general scalability of the platform. It requires the application to be able to cope with fantastic amounts of input data by only reconfiguring the worker tasks of the application, without changing the topological order of those components. Secondly, the implementation should be able to cope with fluctuating data signatures. For this the following criterium was formulated. The platform implementation should return to normal execution parameters within a certain amount of time after experiencing an increased input load.

Criterium parameters

While the functional criteria pose a binary decision on pass or fail, the non-functional criteria require quantification in order to determine weather they hold for the platform implementation. These parameters are based upon contemporary and near-future use cases and have been determined in collaboration with industry experts of Nedap . For the usability requirement it was determined that the implementation should be able to be designed and devised within one FTE·week, i.e. one 40-hour work week. As an absolute upper bound on the amount of code proved to be difficult to determine before-hand, it was defined as "the amount of code required for calculating the QoS parameters in a monolithic application, plus at most 4 lines of code for every component in the platform topology". The parameter of 4 lines of code per component originates in assertion made in Chapter 3.

For the scale of the input signature for the monitoring application, the realistic near-future scale of the sensor application was determined to be 100.000 devices. For the fantastical size of future applications we have taken an increased factor of $\times 10$, i.e. 1 million devices. Finally, for the increased signature

of the temporary event shower to be processed we have chosen a factor of $\times 100$ of the realistic data signature. This burst is supposed to be processed within one minute, after which the application should return to regular execution parameters.

Summary of criteria

The concrete criteria formulated in this section are as follows:

1. An instantiation of the proposed platform should be possible in accordance with the needs and wants of Nedap Identification Services .
2. The platform should enable an instantiation which enables iterative and consequent enrichment and accumulation of information.
3. The platform should be able to output consequences at multiple stages of computation.
4. The instantiation of the platform should take no longer to be developed than one FTE· week (40 hours).
5. The instantiation of the platform should require
   - no more calculation code than it would in a monolithic system, and
   - at most 4 lines of code per component to build the topology.
6. A realistic deployment of the instantiated application should be able to handle an input of 100.000 devices.
7. A reconfiguration of the realistic deployment should be able to handle a hypothetical load of 1 million input devices, without changing the topological order of the components.
8. An event burst of factor 100 should be processed within [number] seconds.

5.5 Design and Implementation

In this section the design for the Nedap SENSIT and its implementation details will be described. We will first take a top-down look at the entire topology. Afterwards we will shortly describe the functionality of the individual components.

5.5.1 Sensor model

To model the state, behaviour and strategies of the sensor we employed the RDM model proposed in Chapter 4. The resulting model is depicted in Figure 5.1. The model takes a few parameters based on the sensor state measurements, such as its current ECL and message rate, and its history, such as its runtime, data already used and budget already used. The model then computes the runtime the sensor has left, current data and budget consumption and the optimal mode of operation. By optimal we mean the operational strategy with the highest message rate that will not exceed the available resources.

Earlier experiments with resource consumption models have shown that a device will act differently when in the beginning than in the end of its life-cycle,

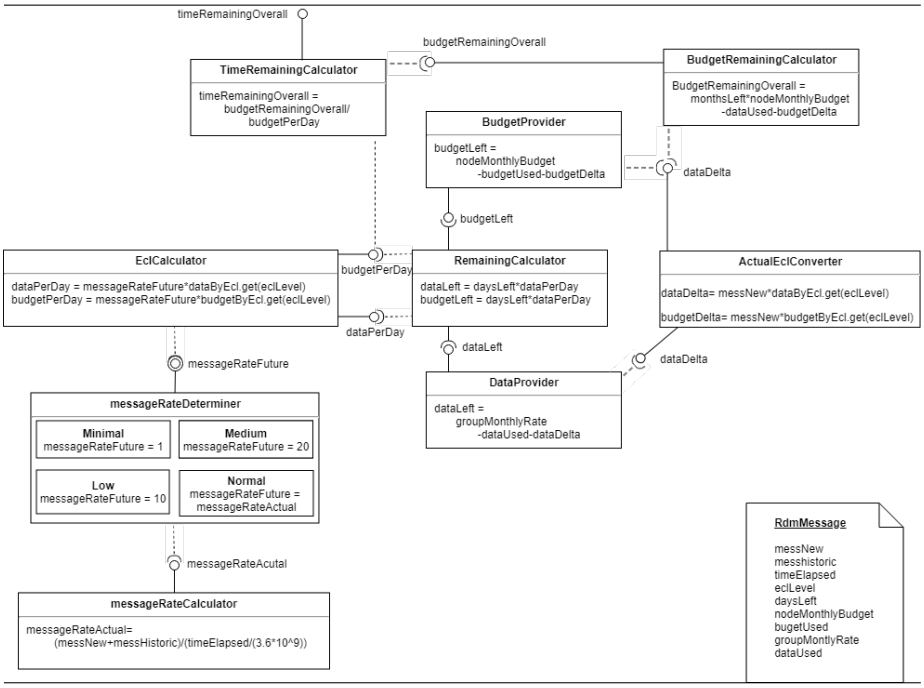

Figure 5.1: Resource Distribution Model for a sensor in the Nedap Identification Services SENSIT WSN application

when there is a scarce resource involved. The reason for this is that in the beginning the models will instruct the device to operate on a strategy that will consume less resources then it is allowed on average. Then, when it has saved up enough of that resource, it is allowed to spend it on a strategy that consumes more than that average. To mitigate this effect is has been chosen to recalculate the available resources on a monthly basis. This way there is still such a cycle, but its period is far shorter and the effect will be much less and much more regular overall.

Final remark on the application design is on the interfaces it provides. The application's inputs and outputs received and provided to Apache Kafka channels. This allows actual services to be easily swapped in and out with test services (even at runtime).

5.5.2 Topology

The designed topology is depicted in Figure 5.2. From this we see that the processing is divided into three stages. In the first stage raw-information snapshots are enriched and averaged. In doing so it improves the information potential and accuracy of the data in the snapshot. The second stage concerns sensor device management. It calculates the state and resource consumption of the devices, and It includes some services that alert if a sensor exhibits abnormal behaviour or long term deviations of its ordinary parameter margins. The final stage concerns snapshot accumulation in order to extract high level information and decisions from it. This stage diverges into three distinct accumulator paths.

The top path performs accumulations of snapshots based on the sensor group ID. It reports on data rate violations (as greed upon in SLA's) and recalculates the share of the data each sensor within a sensor group is allowed to consume. The middle execution path concerns the cells served by nodes. It alerts if a node switches cells more then an allowed amount during a period. The bottom accumulates the all accumulates the all snapshots in order to report on the current state of the application as a whole.

We will conclude the description of the application topology by shortly describing the functions of the individual components.

Senist spout Reads sensor snapshots from a Kafka channel and introduces them into the topology.

Translator Translates the sensor information from hexadecimal to key-value pairs.

Nuancer Averages the data points received from a sensor to eliminate abnormalities.

Attributor Enriches the snapshot with some information known by the backend

Sensor RDM processor Processes the enriched information from the snapshot and calculates the optimal operational strategy.

Switch strategy buffer Buffers the switch strategy messages to prevent superfluous feedback to the sensors. Doesn't switch strategy on first report, only if a switch is requested during an extended period.

Single message analyser Calculates weather the sensor parameter (as calculated by the RDM processor) is within the allowed margins.

Budget recalculator interface If the message rate of a sensor is high enough will execute an immediate budget recalculation. If message rate is low it is allowed to be accumulated over some time to reduce the number of database accesses.

Budget recalculator accumulator Accumulates budget recalculation snapshots and prepares them for batch update.

Budget recalculator Executes (batch) budget recalculation.

Group accumulator Accumulates snapshots by sensor's group ID. Because this is performed on a weekly basis, this is performed two-stage as not to cause a large data build-up overtime.

Group share recalculator Recalculates the share of the sensor group's resources each sensor is allowed to consume, based on the data used by each node over a one week period.

Application accumulator Accumulates the information emitted by the application in order to be presented on an application dashboard.

Cell Switch Analyser Analyses and reports if a node switches between cell towers more then is allowed.

5.5.3 Equipment

[microsoft azure setup]

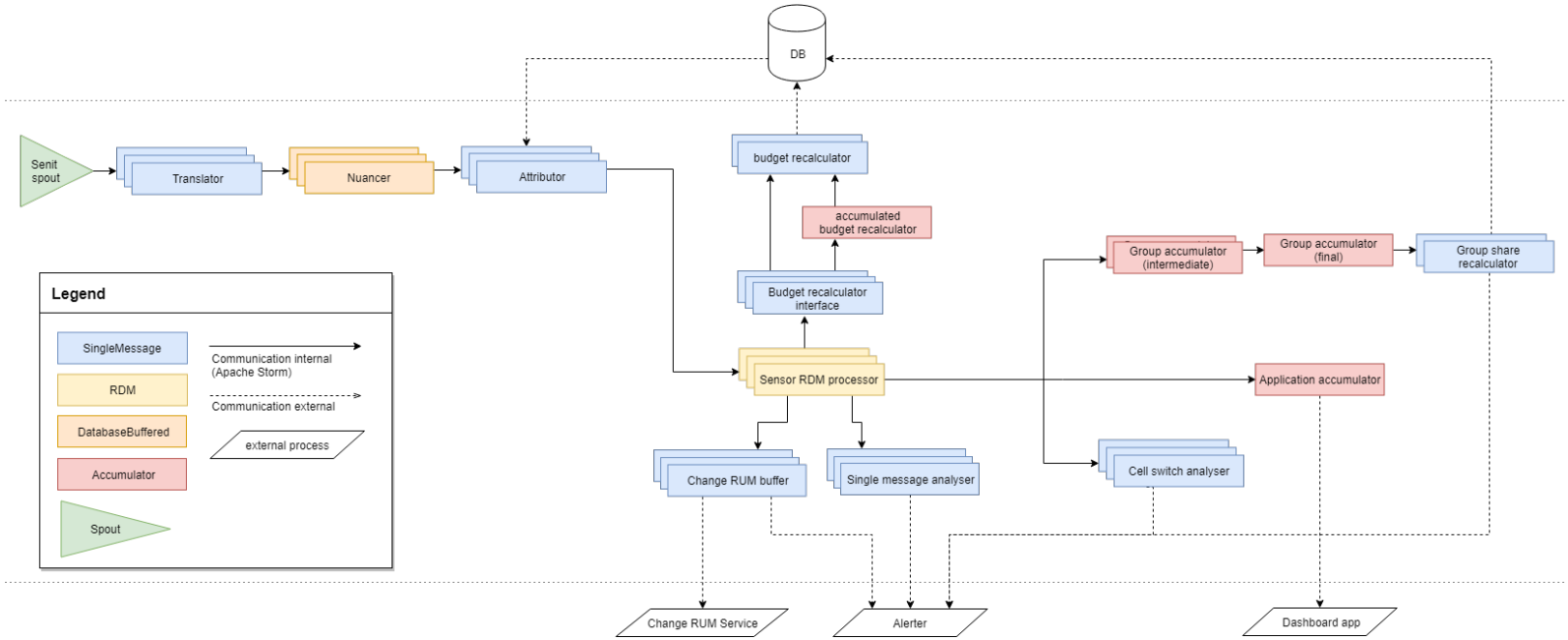

Figure 5.2: Topology of the monitoring application for the Nedap Identification Services SENSIT WSN application

5.6 Results

We will report the results under three headings: applicability, development effort and scalability

Applicability

The sensor model was

The result of the applicability investigation is that the platform suffices as development platform for the purposes of Nedap . The building blocks provided enable the implementation of a functional application and provides functional abstraction of the specifics of the underlying technologies. During implementation of the application it was noted however that the platform does not provide an efficient way of buffering and processing snapshots grouped per node, cell tower, etc.

Development effort

It took about 80 hours to conceive, implement, debug and tune this architecture implementation. This time spent can be divided into roughly 15% design, 35% initial implementation and 50% debugging and tuning [1].

Specifying the sensor model and application topology could be performed within the set parameters. However the internal code of the topology components, which actually perform the calculations and computations, required about twice the amount of code that a monolithic application would. While the actual number of lines of code was only a little higher than then its monolithic counterpart would, the computations and transformations performed on those lines was far more then would be necessary in a monolithic application.

We will deliberate on these discrepancies further in Section 5.7.

Scalability

[TODO timing test]

5.7 Evaluation

In this section we will compare the obtained results and compare them to the criteria set out on in section 5.4.2 and the implications of these results. The criteria will be deliberated in the same order as the results in the previous section were.

5.7.1 Evaluation of Requirement criteria and Claims

Applicability

As stated the building blocks provided by the platform allowed for a sufficient implementation of the intended monitoring application. However it was discovered that, though possibly useful, the platform did not provide an efficient

[1] All hours spent on a component after its first inclusion in the topology and executing it are pooled into the latter category

template to buffer snapshots grouped by a certain snapshot parameter. This component could easily be provided by introducing a mapping function to the *BufferedProcessor* which will determine into which bucket of snapshots a snapshot will be added to. The existing filter, sort and execution methods will then be performed on these buckets individually, providing a means of grouped computations.

However such functionality currently is not present, this absences was easily avoided and was found to be only a minor inconvenience. Since this issue singularly was not sufficient to invalidate the applicability criterium we state that Criteria 1-3 hold.

Development effort

As mentioned it took about 80 hours to construct an develop the prototype application. This is twice as many as was originally stated in the validation criterium (Criterium 4). Secondly, though the code required to compose the resource models and application topology was contained in the parameters specified, the code required was found to be far more than a monolithic application would require. We therefore yield that Criterium 5 was also invalidated.

When we inspect the breakdown of the time spent we yield that it took an enormous amount of time to debug and adapt the components after its original design and implementation. The chief reason for this was found to be the loose coupling between components. The components are completely disjoint and the snapshot variables they share are serialized in between components and accessed with string identifiers. This entails that it is excessively easy to implement a broken component since inappropriate variable access due to misspelled identifiers can occur very easily and is not detected by code checkers and compilers of conventional IDEs. Consequently, when the variable is accessed successfully, the value often requires deserializing into the correct primitive or object. This again introduces a possible point of failure due to parsing and casting, since the compiler cannot detect the actual object type without executing the application.

This required sequential and repeated serializing, deserializing and parsing is also the largest contributing factor to the increased code base within the components. Processing of each (group of) snapshot(s) is prepended with a few lines of code that extract, parse and cast the required variables from the snapshot. After the component's processing is performed a new snapshot is prepared with variable that require its values to be serialized. When the computations of a component only amount to a few lines of code, this (de)serialization can quickly require more code than the actual computations do.

To alleviate this problem we propose the introduction of snapshot struct objects (POJOs). These objects contain the parameters of the messages passed between components. However, in contrast to key-value bindings, these bindings are explicitly defined. This entails that improper variable-access is detected by code checkers and compilers and mismatching or miscasting an object is far more difficult. Additionally these objects can be serialized and deserialized by auxiliary processes in the platform, which alleviates the need for the developer to implement this problematic procedure in each and every component. Combined, this increased traceability and automated (de)serialization should have a noticeable positive effect the time spent debugging and reworking the application and thus the development time as a whole.

Finally, it was noted that after initially specifying the topology and models reworking them proved to be frustrating. The difficulty was mainly in locating the instantiation and declaration of a component in the code that builds the topology. The reason for this is that it constantly requires a developer to transition from a two-dimensional graphic depiction of the model or topology to one-dimensional builder code. This mental transition can be avoided by eventually developing a graphic development tools that allows a developer to conceive a topology by dragging and dropping components and resources. The appropriate computational code can then later be introduced into the components. By doing so a developer would only need to concern themselves with one depiction of the topology instead of two, which might diverge over time.

Scalability

[TODO]

5.7.2 Discussion

We will conclude this chapter by contemplating on the outcomes and limitations of this short validation study.

Conclusions

The main conclusion to draw from this initial validation study is that it indicates the development platform to be a functional tool to develop a functional WSN monitoring application. There are however some small deficiencies and issues to be solved in order to also make the platform more practicable. The first main issue to be resolved are the inclusion functionality to buffer snapshots grouped by some parameter(s) of those snapshots. The second issue regards the inclusion of structs (POJOs) to be used to communicate between components. These structs can be automatically serialized and deserialized and they increase the traceability of datapoints between components. This will reduce the code required. It might be argued that these structs themselves will introduce new code to the application. However this code is easily generated by conventional or custom object generators. This approach will therefore increase the overall development effort required. As the components will no longer be disjunct, but linked by these objects, it will reduce the time spent debugging the application significantly.

Secondly, the inclusion of a graphical model/topology editor will remove the disjoint between graphical design documents and actual implementation. This will further reduce the development effort as a developer is no longer required to transition constantly between two representations of the developed artefacts.

Limitations

Though this validation study demonstrates the platform to be a useful tool, it must be regarded as a proof-of concept. This study only regarded one sensor application and therefore bold statements are not allowed to be made regarding the general applicability of this tool to the field of WSN applications. For such conclusions to be asserted, much more validation on a more varied base of applications is required.

The second large limitation on this study and therefore the conclusions drawn from it, is that this was not designed as a blind study. As the application instantiation of the platform was developed by a developer with full knowledge the validation criteria and intimate knowledge of the internals of the development platform. In order to fully objectively assert the initial conclusions of this study the experiment must be repeated more formally with an impartial subject. This subject (or group of subjects) must be able to repeat the experiments process without knowledge of the parameters of the study, without familiarity of the platforms internals and only the provided documentation of the platform and its exposed APIs.

We however propose that this eventual full-scale study is not performed until the latter stages of platform development and validation. The reason for this is that it is far more resource-efficient to discover initial deficiencies and issues with small case studies, as performed in this chapter. Only when these studies no longer yield suggested improvements to the platform should the scope be focussed towards more expensive, formalized studies.