The handle http://hdl.handle.net/1887/30105 holds various files of this Leiden University dissertation.

**Author:** Etemadi Idgahi (Etemaadi), Ramin  
**Title:** Quality-driven multi-objective optimization of software architecture design: method, tool, and application  
**Issue Date:** 2014-12-11
Chapter 5

Case Studies

In this chapter we validate the AQOSA framework, by applying it to three of software architecture design problems. These cases show the effectiveness and usefulness of automated quality-driven approach for the problem of software architecture design. These case studies will be the basis for the experiments in the following chapters, as well. By executing these experiments based on real-world industrial case studies, this chapter tries to address RQ1 which is mentioned in Section 1.2:

Can meta-heuristic optimization improve the process of designing efficient architectures for a set of given quality attributes in an industrial domain?

To the best of our knowledge this is the largest case study on architecture optimization of real industrial embedded software system.

This chapter is structured as follows. First, Section 5.1 demonstrates the AQOSA framework on a 3-objective optimization problem from the field of business information systems. Through this optimization experiment we also compare the state-of-the-art multi-objective optimization algorithms that are supported by AQOSA. Secondly, Section 5.2 presents a simple embedded software system for cruise control. This case has been used as a demonstration case in the field of multi-objective software architecture design optimization. This case study has 6 software components and between 1 and 6 hardware components. In contrast to the first case, here we study the optimization of 5 design objectives. Thirdly, Section 5.3 presents a study of a real industrial system: the instrument panel. This case was obtained from SAAB automotive. This case has 18 software components and between 1 and 18 hardware components. We study optimization of 5 design objectives. To the best of our knowledge, this is the largest case study on software architecture optimization – esp. of an industrial case. Together these cases demonstrate the effectiveness of the AQOSA framework. Section 5.4 summarizes and reflects on the findings of these experiments in this chapter.
5.1 Case Study 1: Business Report System

The goals of this first case study are (i) to demonstrate the effectiveness of the multi-objective optimization technique for the software architecture domain, and (ii) the comparison between some popular existing evolutionary multi-objective optimization algorithms. The algorithms that we compare are: NSGA-II [DAPM02], SPEA2 [ZLT02], SMS-EMOA [BNE07] and random search (for the domain of software architecture design).

The so-called business reporting system (BRS) is a system which lets users retrieve reports and statistical data about running business processes from a data base. The original case is described in [MKBR10]. It is loosely based on a real system [WW04].

5.1.1 Business Report System Components

From a component-based software development point of view, BRS is a 4-tier system consisting of 8 software components. The \texttt{WebServer} component handles user
requests for generating reports or viewing the plain data logged by the system. It delegates these requests to a Dispatcher component, which in turn distributes the requests to four replicated ReportingServers. The replication helps balancing the load in the system, because the processing load for generating reports from the database contents is considered significant. The ReportingServers access two replicated Databases for the business data. Figure 5.1 depicts an overview of this system.

5.1.2 Execution Scenarios

In addition to a static view of the system, a behavioural view is needed in order to analyse dynamic quality properties of the architecture such as performance. Like the original case in [MKBR10], in this case study we also assume that in the usage scenario, 50 users concurrently access the system. Each user requests a report from the system and then looks at the results for 10 seconds before issuing the next request.

5.1.3 Experiment Setup

In the experiment for the BRS design problem, the following settings are chosen: initial population size: 100, parent population size: 100, number of offspring is 100, for SMS-EMOA (100 + 1), reference point: (1, 1, 1), archive size: 100, the number of generations: 500, crossover rate is set: 0.95, and constant mutation probability: 0.01. AQOSA was run 15 times (≈ 3 hours per run) for each of the evolutionary multi-objective algorithms (NSGA-II, SPEA2, and SMS-EMOA), and also for random search. It was run on a computer with a 2-core processor (each core runs 3.16GHz and 6MB cache) and 2GB memory, using Java platform version 6.

5.1.4 Experiment Results

The resulting set of optimal solutions is visualized in a 3D Pareto front with respect to objectives response time, CPU utilization, and in Figure 5.3a. An interesting finding is that the resulting Pareto front is partitioned into several segments (7 segments in this typical run, in Figure 5.3a). By identifying (using different colors for different topologies) and mapping each individual from the set of solutions to the corresponding design architecture topology, it is observable that solutions from same segmentation share the same architectural topology: Solutions from the same segmentation share the same number of processor nodes. This could be the result of discontinuities in the search space caused by structural transitions.

For comparison between random search and evolutionary algorithms, in Figure 5.3b a 3D plot of random search is also presented. As can be seen, fewer segments (5 in this typical run) have been found by random search. In Figures 5.2a and 5.2b two different
Pareto fronts of two quality attributes (cost vs. CPU utilization and cost vs. response time) are depicted.

The boxplots of the hypervolume indicator with reference point \((1, 1, 1)^T\) for solution sets for NSGA-II, SPEA2, SMS-EMOA, and random search (for a set of 15 runs) are presented in Figure 5.4. From the boxplot (Figure 5.4), it can be seen that SMS-EMOA shows slightly better performance compared to the other algorithms. Random search (even with unbounded archive size) shows the worst performance by far.
5.2 Case Study 2: Cruise Control System

This section discusses a case study about a cruise control (CC) system. This is a safety critical embedded systems from the automotive domain. Our case has been derived from the Vehicle Control system described by Gonzalez-Huerta [GH11]. It is loosely based on Control system described by Hudak et al. [HF07]. The goal of the cruise control system is to maintain a constant speed which is set by the driver. Practically, the system maintains a speed within some (small) interval around the set target speed.

5.2.1 Cruise Control Components

The cruise control system maintains the vehicle speed at the pre-determined value (target value) by storing the speed of the wheel rotation when the speed value is set and attempts to keep the throttle actuator at a position to maintain the vehicle speed at the target value. As the road inclination changes, the vehicle speed changes, and the throttle position should change to maintain the vehicle speed. The control system observes the speed difference between the current speed and the target value and either decreases or increases the throttle actuator position to counter act the speed differential. The algorithm to accomplish this is based on control theory.
Figure 5.5 depicts the flow of the data in CC system in Use Case Maps notation \cite{ALBG99}. The system contains the following six components that aim to generate four output signals:

- **IN\_CONTROL** Component: This component controls the input signals. It receives the brake pedal status and engine status which are its most important signals. It also receives the driver speed panel signals, to enable him to control the car speed. It calculates its output based on the values of input signals.

- **DESIRE\_SPEED** Component: This component computes the desired speed based on the current car speed and signals from controller. It reads the selected speed and selected distance signals and generates the desired speed signal for the next 4 components by applying the control law.

- **AIRBAG** Component: This component generates airbag settings. It is responsible for sending proper airbag signals in case of collision.

- **SECURITY\_BELT** Component: In case of hard braking this component should send signals to the security belts. It receives data from two sensors (i) the obstacle sensor which is responsible for detecting obstacles on the road, and (ii) the distance sensor which is responsible for calculating distances to the next car.

- **THROTTLE** Component: This component is responsible for configuring the settings of the car throttle. It converts the relative speed into a throttle setting for the throttle actuator.

- **BRAKE** Component: This component should activate the car brakes when needed. This component only works in case of decreasing speed. This component converts the relative speed into a brake setting for the brake actuator.

### 5.2.2 Execution Scenarios

For this experimental study, we use the following scenario: a 2-second (2000ms) scenario is defined. At $t = 1900\text{ms}$, the obstacle sensor recognizes an obstacle in front of the car. Then, the cruise control system should react to this event within the predefined deadlines. Prior to this major event, the following periodic events happen continuously:

1. Every 50ms: a signal is sent for security belt settings,

2. Every 100ms: a signal is sent for airbag settings,

3. Every 150ms: a signal is sent for throttle and brake settings.
The system should generate 4 signals on its outputs. Based on the system requirements, these are the system deadlines:

- 50ms for security belt signal output,
- 80ms for airbag signal output,
- 30ms for brake signal output,
- 100ms for throttle signal output.

5.2.3 Experiment Setup

For solving the CC system design problem, a repository which consists of the following hardware components, has been considered:
28 Processors types: ranging over 14 various processing speeds from 66MHz to 500MHz. Each of these has two levels of energy consumption. A processor is more expensive if it has a smaller chance of failure.

15 Buses: ranging over five types of bandwidths (128, 160, 192, 224 and 256 kbps) and each type has three variants with different latencies (1, 3 and 5 ms). Again, a bus is more expensive if it supports higher bandwidth or smaller time delay.

The AQOSA optimization was run with the following parameter settings on a computer with a 2-core processor (each core runs 3.16GHz and 6MB cache) and 2GB memory: initial population size ($\alpha$) = 100, parent population size ($\mu$) = 50, number of offspring ($\lambda$) = 50, archive size (with crowding type) = 50, number of generations = 10000, crossover rate is set to 0.95.

5.2.4 Experiment Results

The 2D Pareto fronts in Figure 5.6 show the trade-off between objectives. Improving one dimension often implies a decrease of the other dimension. Different colors in the plots correspond to different numbers of nodes in the architecture: black for 1-node, orange for 2-node, magenta for 3-node, blue for 4-node, green for 5-node and red for 6-node architectures. As can be seen there is no black node which means AQOSA could not find an optimal solution with a 1-node architecture. As can be seen, the results of this case study is not clearly segmented like the previous case study, but still similar architectures are close to each other.

In Figure 5.7 a 3D Pareto front of a set of optimal solutions is depicted. It shows the trade-off between Cost vs. CPU utilization vs. Response time. The colors in the plot are the same as the color in the 2D plots.
Case Study 3: SAAB Instrument Cluster Sub-System

5.3 Case Study 3: SAAB Instrument Cluster Sub-System

The goal of this case study is to explore in what aspects meta-heuristic optimization improves the process of designing architectures for a set of given quality attributes in an industrial context. This real-world case study shows how an architecture optimization framework helps system architects make better decisions by complementing their domain knowledge and experience.

This study was conducted at Saab Automobile AB in order to evaluate and validate the AQOSA framework in a large industrial design problem. To enable the validation of the results, an existing realization for the Saab 9-5 Instrument Cluster Module ECU (Electronic Control Unit, a node in a network) and the surrounding sub-system have been selected. The aim of the optimization experiment is to find a solution that is cheaper than the current realization while fulfilling the same requirements and constraints. This constitutes a fair comparison between the solution proposed by AQOSA and the current industrial realization. This problem is an ever-existing problem in the automotive industry, because of high cost pressure and variation in constraints during the lifetime of a system design.

Figure 5.7: 3D Pareto front for cruise control system
<table>
<thead>
<tr>
<th><strong>User function</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Speed Indication</td>
<td>Shows the speed of the car on the speedometer gauge. The vehicle speed information is based on the wheel speed sensors, filtered and converted into the chosen metric (either km/h or miles/h).</td>
</tr>
<tr>
<td>Coolant Temperature Indication</td>
<td>Shows the temperature of the engine coolant on the coolant temperature gauge. The coolant temperature information is based on a temperature sensor, filtered and compensated to show the correct temperature.</td>
</tr>
<tr>
<td>Selected Gear Indication</td>
<td>Displays the current automatic gear position of the vehicle. (Possible gear positions are P, R, N, D, and L.) The information is based on position sensors in the Automatic Gearbox.</td>
</tr>
<tr>
<td>Engine Speed Indication</td>
<td>Shows the engine revolutions per minute information on the tachometer gauge. The engine speed information is based on the crankshaft sensor, and filtered before it is shown.</td>
</tr>
<tr>
<td>Odometer Indication</td>
<td>Shows the distance travelled by the car. The odometer is based on the wheel speed sensors, filtered and converted into distance in the chosen metric (either km or miles).</td>
</tr>
<tr>
<td>Ignition Switch Power Moding</td>
<td>The global system power mode in the car. The system power mode information is based on the ignition key switch, with the possible positions OFF, ACCESSORY, RUN, and CRANK. This information is transmitted to all parts of the system.</td>
</tr>
<tr>
<td>Outside Air Temperature Indication</td>
<td>Shows the outside temperature on the display. The outside temperature information is based on a temperature sensor, filtered and compensated to show the correct temperature.</td>
</tr>
<tr>
<td>Low Washer Indication</td>
<td>Notifies the driver when the windshield wiper fluid level is low. The notification is based on a fluid level switch.</td>
</tr>
</tbody>
</table>

**Table 5.1:** Description of user functions for the SAAB Instrument Cluster sub-system

### 5.3.1 Instrument Cluster Components

The purpose of the Instrument Cluster sub-system is to provide the driver with information that is required when driving the car. The information is processed by user functions based on sensor values, and presented on gauges or displays located in front of the driver. The user functions (and their input/output devices) included in the study are shown in Table 5.1.

User functions like the ones in Table 5.1 can be complex due to regulations, safety properties, variation between different car models, or other quality constraints. In
Figure 5.8: Component diagram of SAAB Instrument Cluster sub-system
general, user functions have to satisfy multiple requirements and quality constraints. User functions are specified and implemented using one or more software components. Figure 5.8 shows the component diagram of the Instrument Cluster sub-system. The diagram shows how the software components are connected to satisfy the user functions in Table 5.1. Figure 5.8 also illustrates the size and complexity of this case.

5.3.2 Execution Scenarios

The user functions in Table 5.1 are typically triggered by events captured by their input devices (sporadic tasks), or by an invocation from a scheduler within the sub-system (periodic tasks). These tasks were defined by describing changes in switch/sensor values. The sporadic tasks are defined in Table 5.2 and the periodic tasks are defined in Table 5.3.

As an example of a stimulus and system response for a sporadic task we will look at the first task in Table 5.2. The DriverDoorAjarSwitch detects that the driver door is opened while the vehicle is parked and the engine is OFF. Then the Odometer value shall be displayed within 500ms. The other tasks in Table 5.2 are of similar kind.

As an example of a stimulus and system response for a periodic task we will look at the first task in Table 5.3. The first periodic task is executed when the engine is running and the vehicle speed is changing. Then the VehicleSpeed signal shall be transmitted periodically each 100ms, the VehicleSpeedDisplayValue shall be calculated periodically each 100ms, and the Vehicle Speed pointer shall begin to move

<table>
<thead>
<tr>
<th>Stimulus</th>
<th>System response</th>
</tr>
</thead>
<tbody>
<tr>
<td>IgnitionSwitch = 0 (OFF) ∧ DoorAjarSwitch : False → True</td>
<td>Odometer shall be displayed within 500 ms</td>
</tr>
<tr>
<td>IgnitionSwitch = 0 (OFF) ∧ TripStemButton : False → True</td>
<td>Odometer shall be displayed within 500 ms</td>
</tr>
<tr>
<td>IgnitionSwitch : 0 (OFF) → 2 (RUN)</td>
<td>The Engine Speed pointer shall begin to move to the correct position within 150 ms</td>
</tr>
<tr>
<td>IgnitionSwitch : 0 (OFF) → 2 (RUN)</td>
<td>The Vehicle Speed pointer shall begin to move to the correct position within 150 ms</td>
</tr>
<tr>
<td>IgnitionSwitch = 2 (RUN) ∧ WasherFluidSensor : False → True</td>
<td>Low Washer Fluid Indicator shall be illuminated within 250 ms</td>
</tr>
<tr>
<td>IgnitionSwitch = 2 (RUN) ∧ GearLeverPositionSwitch : 1 (P) → 4 (D)</td>
<td>Driving gear position shall be displayed within 100 ms</td>
</tr>
</tbody>
</table>

Table 5.2: Sporadic tasks included in SAAB Instrument Cluster sub-system
### Table 5.3: Periodic tasks included in SAAB Instrument Cluster sub-system

<table>
<thead>
<tr>
<th>Stimulus</th>
<th>System response</th>
</tr>
</thead>
<tbody>
<tr>
<td>IgnitionSwitch = 2 (RUN) ∧ VehicleSpeedSensor : 0 → 100km/h</td>
<td>VehicleSpeed signal shall be transmitted periodically each 100 ms. VehicleSpeedDisplayValue shall be calculated periodically each 100 ms. The Vehicle Speed pointer shall begin to move to the correct position within 150 ms.</td>
</tr>
<tr>
<td>IgnitionSwitch = 2 (RUN) ∧ CrankshaftSensor : 0 → 5000rpm</td>
<td>EngineSpeed signal shall be transmitted periodically each 100 ms. EngineSpeedDisplayValue shall be calculated periodically each 100 ms. The Engine Speed pointer shall begin to move to the correct position within 150 ms.</td>
</tr>
<tr>
<td>IgnitionSwitch = 2 (RUN) ∧ OutsideTempSensor : 0 → 100%</td>
<td>The outside air temperature shall be calculated once every second. The outside air temperature shall be displayed within 100 ms.</td>
</tr>
<tr>
<td>IgnitionSwitch = 2 (RUN) ∧ CoolantTempSensor : 0 → 100%</td>
<td>EngineCoolantTemp signal shall be transmitted periodically each 100 ms. CoolantDisplayValue shall be calculated periodically each 100 ms. The Coolant Temp pointer shall begin to move to the correct position within 150 ms.</td>
</tr>
</tbody>
</table>

As stated before, user functions like the ones in Table 5.3 are typically implemented by one or more software components. We use sequence diagrams to model the collaboration among these components. Sequence diagrams represent specific scenarios. The use of sequence diagrams was a convenient way of capturing the information needed for AQOSA. It required domain knowledge to create them, especially to understand how the components interact and how to add the timing constraints. Ten sequence diagrams are made (as shown in Figures 5.9 to 5.18), and the sequence diagrams contain between 3 and 6 software components with 4 components as the average.

These sequence diagrams need to be enhanced with information required for AQOSA quality attributes evaluation process. As such, AQOSA requires the number of cycles for executing each operation is needed. The number of execution cycles for each operation was obtained by analyzing the source code of the component with the SCoPE simulation framework [PHS+04]. Components for which the source code was not accessible, were analyzed manually by reading the requirement specification and comparing this to similar components, to estimate the number of execution cycles. This information is added next to the activation on the sequence diagram. AQOSA also
requires the number of bytes being sent in order to compute communication loads. The number of bytes sent in each message was obtained by calculating the size of the data that is sent. This information is added below the messages.

Here, the first sequence diagram is discussed in details. The other sequence diagrams are similar, and follow the same principles. For example, as described in the first sequence diagram in Figure 5.9, the operation `CalculateOAT()` takes 2744 cycles to execute. This number of execution cycles for the `CalculateOAT()` operation was obtained by analyzing the source code of the `ControlOutAirTemp` component with the SCoPE simulation framework. Also in the same diagram (Figure 5.9), we see that the size of message `CalculateOAT()` is 1 byte.

In addition to the information above, AQOSA also needs the timing constraints described in Table 5.2 and Table 5.3. This information was added to the sequence diagram in a standard way to the left of the sequence diagram and with the help of notes. Again, an example from the first diagram is that on the left side in Figure 5.9, we see that the maximum delay for the whole scenario should be less than 100ms. The note added to the message `ObtainOAT()` constrains it to be invoked each 1000ms.

As can be seen in Figure 5.9, there are three actors (User, OAT Sensor, and Display) and three software components (ReadOATSensor, ControlOutAirTemp, and Display_Engine). After these sequence diagrams have been created, it is a simple task to feed this information into AQOSA framework.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wheel Speed Sensor shall always be connected to the Brake Module</td>
</tr>
<tr>
<td>2</td>
<td>Crankshaft Sensor and Engine Coolant Temp Sensor shall always be connected to the Engine Module</td>
</tr>
<tr>
<td>3</td>
<td>The Brake Module and the Engine Module shall always be connected to a HS CAN bus</td>
</tr>
</tbody>
</table>

Table 5.4: Deployment constraints for SAAB Instrument Cluster sub-system

The user functions (see Table 5.1) and the timing constraints (see Table 5.2 and Table 5.3) are needed to obtain the required input to AQOSA, but there is one more important type of constraint for this domain. Deployment constraints state which software components may or may not be mapped onto which hardware components. The deployment constraints used in our case study are stated in Table 5.4.

In the following, the sequence diagram for each user function of the system is depicted. It contains the deadline and the required number of cycles for each operation.
Outside Air Temperature Indication

Figure 5.9 depicts the sequence diagram of OutsideAirTemperatureIndication. As already mentioned, it consists of three actors (User, OAT Sensor, and Display) and three software components.

ReadOATSensor is being called every 100ms. ReadOATSensor itself gets data from the sensor and calls ControlOutAirTemp. After that, ControlOutAirTemp consequently calls Display_Engine to display proper information. The deadline for this task is 100ms.

Figure 5.9: Sequence diagram of OutsideAirTemperatureIndication user function
Vehicle Speed Indication

Figure 5.10 depicts the sequence diagram of VehicleSpeedIndication. It is also a periodic task, in which ReadWheelSpeedSensors component is being called every 100ms. It gets needed data from the wheel sensors and calls the ControlWheelSpeed component. Then, ControlWheelSpeed, calls EngineVehicleInterface, and it calls ControlVehicleSpeedGauge, and it calls Gauge_Engine respectively. At the end, Gauge_Engine sends data to physical or electronic vehicle speed gauge. The deadline for this task is 50ms.

Figure 5.10: Sequence diagram of VehicleSpeedIndication user function
Vehicle Speed on Start

Figure 5.11 depicts the sequence diagram of VehicleSpeed_onStart. It is more or less similar to the previous one, but this task is a sporadic task, instead of a periodic one. Hence, it only happens when the driver turns the ignition switch. As can be seen from the diagram, the components interact with two sensors, **Ignition Switch**, and **Wheel Speed Sensors**. The interactions between the components are quite similar to the previous one. However, the deadline for this task is 150ms.

*Figure 5.11: Sequence diagram of VehicleSpeed_onStart user function*
Engine Speed Indication

The sequence diagram for `EngineSpeedIndication` is depicted in Figure 5.12. It is a periodic task. It consists of three actors (`User`, `Crankshaft Sensor`, and `Engine Speed Gauge`) and three software components. At the start, the `EngineVehicleInterface` component is being called every 100ms. It gets needed data from the crankshaft sensor and calls the `ControlEngineSpeedGauge` component. It also adjusts the data and calls the `Gauge_Engine` component. After that, `Gauge_Engine` sends data to physical or electronic engine speed gauge. The deadline for this task is 50ms.

Figure 5.12: Sequence diagram of `EngineSpeedIndication` user function
Engine Speed on Start

The sequence diagram for EngineSpeed_onStart is depicted in Figure 5.13. Again, it is quite similar to the previous one, but it is a sporadic task. Therefore, we can see Ignition Switch sensor is involved in the sequence. The deadline for this task is 150ms.

Figure 5.13: Sequence diagram of EngineSpeed_onStart user function
Coolant Temperature Indication

CoolantTemperatureIndication is another periodic task. Figure 5.14 shows the sequence diagram for it. As shown in the diagram, the EngineVehicleInterface component is being called every 100ms. It gets the required data for this task from the Engine Coolant Temperature Sensor and calls the ControlCoolantTempGauge component to adjust the data. Then, ControlCoolantTempGauge calls Gauge_Engine, and finally, it sends data to the physical needle. The system has 150ms time to finish this task.

Figure 5.14: Sequence diagram of CoolantTemperatureIndication user function
Selected Gear Indication

The sequence diagram of SelectedGearIndication is shown in Figure 5.15. This task is a sporadic task and it is being triggered by Lever Position Sensor. When this event happens, TransmissionVehicleInterface will be called. It sends the data for the ControlGearSelectedIndication component to interpret the data. After that, it manipulates the data and sends it to Display_Engine to be displayed for the User. The deadline for this task is 100ms.

Figure 5.15: Sequence diagram of SelectedGearIndication user function
Odometer Indication on Trip Stem Button Activation

This is a sporadic task which will be run when the user pushes the Trip Stem Button. Figure 5.16 depicts the sequence diagram of it. It interacts with an external system named Control Odometer Storage which stores odometer data for the future retrieval purposes. As can be observed from the diagram, after the trigger event happens, the ReadTripStemButton component is called. It calls the ControlOdometer component. This component interacts with an external system and then sends proper data to Display_Engine to be displayed on the dashboard panel. The deadline for this task is 500ms.

Figure 5.16: Sequence diagram of OdometerIndication user function on Trip Stem Button Activation event
Odometer Indication on Door Ajar Activation

This is a sporadic task is again quite similar to the previous one, however it will be run when the driver opens the car’s door. Figure 5.17 depicts the sequence diagram of this task. It also interacts with the external system of Control Odometer Storage and its deadline is 500ms. The main sequence of data and components calling is the same as previous one.

Figure 5.17: Sequence diagram of OdometerIndication user function on Door Ajar Activation event
Low Washer Indication

The sequence diagram for \textit{LowWasherIndication} is depicted in Figure 5.18. Its sequence is pretty straightforward. The \textit{ReadLowWasherLevel} component is called when the switch detects that the washer fluid is at a low level. \textit{ReadLowWasherLevel} gets data from the sensor and calls the \textit{ControlWasherLevelIndication} component. It adjusts the data and sends proper information to \textit{Display_Engine} to be displayed on the dashboard panel.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{sequence_diagram.png}
\caption{Sequence diagram of \textit{LowWasherIndication} user function}
\end{figure}
5.3.3 Current Realization

The task for AQOSA is to propose candidate software architectures that are optimized with regards to five quality attributes: bus utilization, cost, CPU utilization, response time, and safety. From the proposed realizations, one solution shall be selected based on the quality attribute values and deployment constraints. The current realization of the Instrument Cluster sub-system is shown in Figure 5.19. This realization is used as baseline when validating the results of the study. In Figure 5.19 each box is an ECU-processor, and the lines between these boxes are communication buses. The displays and gauges are part of the Instrument Cluster Module (To be consistent with Saab terminology, the term ‘module’ has been used for a hardware node in this case study).

The hardware data and cost of the current realization will be used as a baseline for evaluating the proposed solutions. The most important hardware data and the hardware cost are shown in Table 5.5.

5.3.4 Experiment Setup

For generating new software architectures, a repository of hardware components was assembled based on predictions of planned system performance and anticipated market prices;

- 10 Processors: ranging over 5 various processing speeds from 10 MIPS to 100 MIPS. Each of these has two levels of failure rate. A processor is more expensive if it has more processing power or a lower failure rate.
<table>
<thead>
<tr>
<th><strong>ECU</strong></th>
<th><strong>Cost (USD)</strong></th>
<th><strong>MIPS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Brake Module</td>
<td>100</td>
<td>80</td>
</tr>
<tr>
<td>Central Module</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>Door Switch Module</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>Engine Module</td>
<td>120</td>
<td>100</td>
</tr>
<tr>
<td>Instrument Cluster Module</td>
<td><strong>50</strong>(^1)</td>
<td>60</td>
</tr>
<tr>
<td>Transmission Module</td>
<td>50</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Piece of hardware</strong></th>
<th><strong>Cost (USD)</strong></th>
<th><strong>kbps</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>HS CAN (cost/module)</td>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td>LS CAN (cost/module)</td>
<td>0.25</td>
<td>33</td>
</tr>
<tr>
<td>LIN (cost/module)</td>
<td>0.1</td>
<td>10</td>
</tr>
</tbody>
</table>

**Table 5.5: Hardware data and cost**

- 4 Buses: with bandwidths of 10, 33, 125, and 500 kbps, and latencies of 50, 16, 8, and 2 ms. A bus is more expensive if it supports a higher bandwidth.

As an estimate of the size of the design space of this case study, consider the following reasoning: Assume we fix that the architecture has six processors (like the current realization, see Figure 5.19, and thus exclude many alternatives in hardware topology) and three bus lines for their interconnections. For these constraints there are \(10^6 \cdot 4^3\) different possibilities, which is equal to \(64 \cdot 10^6\) architectures. When also considering variations in the architecture topologies, this number would be considerably higher. This design space needs to be searched in an efficient manner.

After defining the above hardware options, AQOSA was run 30 times using the NSGA-II algorithm with the following parameter settings: initial population size (\(\alpha\))=2000, parent population size (\(\mu\))=100, number of offspring (\(\lambda\))=50, archive size=50, number of generations=5000, crossover rate is set to 0.75, constant mutation probability is 0.01 and all quality attributes are aimed to be minimized.

### 5.3.5 Experiment Results

Figure 5.20, Figure 5.21, and Figure 5.22 show the results produced by AQOSA for the Instrument Cluster sub-system. Figure 5.20 contains a set of 2D plots for all pairs of two out of the five quality attributes that are considered in this design problem. Hence these depict 2-dimensional views on the resulting 5-dimensional Pareto front. Each box shows a particular view of the 5D result and the relation between two attributes.

\(^1\)The design cost driven by "look and feel" requirements for display panel have been excluded because this is a cost that is common to all solutions.
Some of the attributes are in conflict like CPU util and cost. Some of the attributes are positively related, like response time and CPU util, and finally, some of them are independent such as bus util and safety or response time and safety. Pareto fronts are used as support for trade-offs between attributes by showing which solutions that are the best, i.e. Pareto-optimal, and also showing which trade-offs are possible. Different

Figure 5.20: Pareto front views for all pairs of two out of five quality attributes
shapes and shades of gray in the plot show solutions with different numbers of nodes in the architecture: from black square for 1-node, dark gray square for 2-node, gray square for 3-node, light gray square for 4-node, black circle for 5-node, dark gray circle for 6-node, and continues up to light gray diamond for 18-node architectures. As can be seen there are architecture solutions containing 1-, 2-, 3-, 4-, 5-, 6-nodes, and more nodes up to 18. In Figure 5.20 the "∗" sign shows the current realization (see Figure 5.19). Figure 5.21 shows a zoom-in on the Pareto front of Response time vs. Cost. The Pareto fronts in Figure 5.20 illustrate that the current realization is not Pareto-optimal. For example, Figure 5.21 shows that AQOSA found other solutions that are better both in response time and cost.

Figure 5.22 shows the parallel coordinate plot of the optimized solutions. In this plot each line (from left to right across the entire diagram) represents one architecture solution: the (normalized) values for the different attributes of the solution are marked by the crossing of this line and the vertical attribute axes. From this representation of the data, it can be observed that no architecture solution is optimal in all attributes.

Next, we discuss some of the proposed solutions in more detail. The cheapest candidate contains 1 node with a 60 MIPS processor, no communication bus, and a total cost of 50 USD. The safest candidate contains 7 nodes connected by 8 buses with a total cost of 459 USD. Table 5.6 shows the attribute values for these candidate solutions. The current realization (see Figure 5.19) is included as for comparison. In Table 5.6, bus utilization is shown as average value (1.0 represents maximum
Case Study 3: SAAB Instrument Cluster Sub-System

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Bus Util</th>
<th>Cost</th>
<th>CPU Util</th>
<th>Response time</th>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheapest</td>
<td>0</td>
<td>0.050</td>
<td>0.164</td>
<td>0.191</td>
<td>0.412</td>
</tr>
<tr>
<td>Safest</td>
<td>1.62E−4</td>
<td>0.459</td>
<td>0.018</td>
<td>0.077</td>
<td>0.409</td>
</tr>
<tr>
<td>Current Solution</td>
<td>9.80E−3</td>
<td>0.331</td>
<td>0.026</td>
<td>0.191</td>
<td>0.414</td>
</tr>
<tr>
<td>Selected Solution</td>
<td>1.62E−4</td>
<td>0.066</td>
<td>0.110</td>
<td>0.158</td>
<td>0.412</td>
</tr>
</tbody>
</table>

Table 5.6: Quality attribute values for a selection of candidate solutions

utilization), cost is shown in USD/1000, CPU utilization is shown as average value (1.0 represents maximum utilization), response time is shown as the average of response time relative to deadline for each task, and safety is shown as failure probability rate. When comparing the attribute values for the cheapest candidate, the safest candidate and the current solution in Table 5.6, it shows that the cheapest candidate is lowest in cost but worst in CPU utilization and response time, and that the safest candidate is best in safety but high in cost and a little worse than the current solution in CPU utilization.

Figure 5.22: Parallel coordinate plot of optimized solutions
The goal of the case study is to find a solution that is cheaper than the current realization while fulfilling the requirements and constraints. The highlighted area in Figure 5.22 delineates the area that the architect considers suitable solutions based on quality attribute values, depicted in a parallel coordinate plot. After analysing the results of applying AQOSA and considering all attributes and constraints, an overall best solution was identified. In Figure 5.22, the bold line represents the selected solution. It contains 2 nodes: one 60 MIPS processor and one 10 MIPS processor, respectively. 12 components are deployed on the 60 MIPS node, and the remaining 6 components are deployed on the 10 MIPS node. The displays and gauges are part of the 60 MIPS node. The nodes are connected to a 500 kbps HS CAN bus. The selected solution is shown in Figure 5.23, and the attribute values are shown in Table 5.6. When comparing the attribute values of the current solution and the selected solution, it shows that the selected solution is significantly lower in cost, better in response time, and slightly better in safety. On the other hand, the selected solution is higher in CPU utilization than the current realization. This is not regarded as a problem, since the CPU utilization is still on a low level and the response time is low. The bus utilization is low in both solutions. So, it can be concluded that the selected solution is a cheaper and better realization, given the user functions and the attributes in the case study. This is also confirmed by the Pareto fronts in Figure 5.20 and Figure 5.21, which show that the selected solution (denoted by a black square symbol) is Pareto-optimal.
5.3.6 Validation and Discussion of Results

As mentioned earlier, the goal of this case study was to find a solution that is cheaper than the current realization while fulfilling the requirements and constraints. This is an ever-existing problem in the automotive industry, because of high cost pressure and variation in constraints during the lifetime of a system design. The product cost is important for every industrial sector producing embedded systems in large numbers, such as the automotive sector. The expected lifetime of an automotive system design is around 5 – 8 years. This lifetime is challenged by changes in constraints, such as optional functions becoming standard functions, or hardware parts becoming obsolete. Therefore, the problem studied in the case study is a real problem in the automotive industry.

We used the same requirement specifications for AQOSA as was used by the architects at Saab when designing the current realization. However, for practical reasons the focus of our case study research was on the most important user functions, and the most important constraints. Thus, the case study is based on requirement specifications for an actual industrial design problem.

The candidate architectures produced by the AQOSA optimization framework were presented to one of the main architects behind the current solution\(^2\). He agreed that the suggested solution by AQOSA is a suitable starting architecture for the next generation of cars at Saab.

A limiting factor for the accuracy of our results is the method to obtain the number of execution cycles for an operation. Components for which the source code was available, were analyzed with the SCoPE simulation framework [PHS+04], to compile the number of execution cycles. Components for which the source code was not available, were analyzed manually by reading the requirement specification and comparing this to similar components, to estimate the number of execution cycles. This was an efficient way of using the information that was available for each component. A quantitative accuracy assessment of the results from the AQOSA optimization framework could be obtained by measuring the quality attribute values of the real system and comparing to the simulated quality attribute values. This is part of the future work.

Regarding threats to the validity of our results, the main threat is external validity which concerns generalization of the results outside the context of the study [RHRR12]. The case study was conducted at one automotive company using specifications, software, and data from that particular company. The results of the case study suggest that the architecture optimization framework can be applied to other embedded systems, but this needs to be assessed by conducting additional case studies in other contexts.

---

\(^2\)The architect now works for a different company in the automotive sector.
5.3.7 Interpretation of Results

The purpose of the AQOSA framework is to support the architect in the early phases of architecture design, and especially with considering various quality constraints. So, the underlying approach is to combine the domain knowledge and experience of the architect with the optimization, simulation, and analysis skills of the AQOSA framework. The case study illustrates how this combination can solve a practical problem. The domain knowledge and experience of the architect is needed when defining the problem to be solved, when creating the models used as input to AQOSA and when evaluating the output from AQOSA. The optimization, simulation, and analysis skills of AQOSA are needed when searching very large design spaces, when analyzing a large number of potential solutions, and when considering various quality attributes in the analysis.

The case study shows the importance of considering all attributes and constraints when designing the architecture. If only a subset of the attributes are considered during design, there is a risk to select a solution that is infeasible with respect to other equally important attributes. Five quality attributes and three deployment constraints were considered during design. The theoretical design space is $64 \cdot 10^6$ given the repository of hardware components. Solving this design problem using the AQOSA framework required two sources of effort: a manual effort of 113 man hours in for creating the models of the components (this was largely done based on profiling of source code) in the repository as well as the behaviour models. Also, some man-hours were spent on analyzing the output of AQOSA. Additionally, 90 hours of continuous computer execution time was used for running the AQOSA optimization software. For a next design problem in the same domain the models of the components and behaviour can be reused with could reduce the time needed for making an analysis.

We found that the AQOSA framework supports the architect by proposing revolutionary [Axe09] [LH12] architectural solutions. A human architect normally uses previous architectures as a starting point for new architectures, which might prevent revolutionary ideas. However, the results from the case study show that AQOSA proposes very different solutions ranging from 1 hardware node up to 18 hardware nodes. Architects at Saab also confirmed that they considered AQOSA as a good tool for generating completely new solution ideas. Nevertheless, it should be noted that a human architect needs to assess and potentially modify the solutions proposed by AQOSA. Moreover, the framework uses plug-ins for analyzing several quality attributes simultaneously, which is important support because considering several quality attributes simultaneously is a difficult task for a human architect. Hence, by providing automatic and simultaneous analysis of quality attributes, AQOSA saves manual effort and development cost.
5.4 Summary

The studies in this chapter demonstrated the usefulness of software architecture optimization framework through 3 different case studies. These studies range from business information systems to embedded systems in the automotive industry. These case studies will be used as running case studies for next chapters as well.

The last case in this chapter reports on a real-world large-scale industrial case study applying a meta-heuristic optimization approach for automated software architecture design which supports multiple quality attributes. The case shows in an industrial context how meta-heuristic optimization approaches can improve software architecture design with respect to multiple quality attributes, and can suggest a wide range of optimized architectural solutions. Comparing the solutions proposed by AQOSA to the existing realization shows that AQOSA is able to synthesize efficient solutions in all quality attributes while fulfilling given constraints. Also, in contrast to human architects who tend to propose solutions based on previous architectures, AQOSA proposes revolutionary solutions.

Although the case study shows the saving of manual effort which is beneficial for time-to-market and development cost, it also shows that the proposed architecture solutions needs to be assessed by human architects. So, this chapter demonstrates how an architecture design framework like AQOSA complements the domain knowledge and experience of the architect.

To summarize, the case studies in this chapter show that meta-heuristic optimization can improve the process of designing efficient architectures for a set of given quality attributes in the following aspects:

- efficient search of large solution space
- considers multiple system quality attributes in design and analysis
- proposes revolutionary solutions