Case Studies on E/E Architectures for Power Window and Central Door Locks Systems

Jordan Ross, Michał Antkiewicz, and Krzysztof Czarnecki
Technical Report: GSDLAB-TR-2016-06-23
Generative Software Development Lab, University of Waterloo, Canada
{mantkiew,j25ross,kczarnec}@gsd.uwaterloo.ca

Abstract—Architectural optimization for software-intensive systems is an emerging area. The automotive industry needs optimized architectures in order to develop cheaper, lighter, and more reliable cars which are growing in software complexity. However, there is a lack of benchmarks that are used to evaluate the performance of the optimzation algorithms which target the area of architecture optimization, as proposed by Aleti et. al. in [2]. In this technical report, we contribute two automotive electric/electronic (E/E) architecture case studies for a power window and a door locks systems. Additionally, we define a reference model focused on early design, which is similar to the EAST-ADL domain model. We use the modeling language *Clafer* to model both case studies; however, we provide sufficient details to allow researchers replicating them in other modeling languages such as SysML or AADL.

Index Terms—automotive E/E architecture, case study, Clafer, early design, synthesis, optimization

1 Introduction

Over the past decade the need for modeling of automotive electric/electronic (E/E) architectures has grown. The EAST-ADL [5] is one formalized way of approaching the issue. In addition, numerous research tools which focused on automotive E/E architectures have aimed to provide some additional benefits for modeling. One such way is by optimizing some aspect of these architectures, whether it is the deployment of functions to ECU's, the number of nodes needed in an architecture, or the routing of wires through a harness.

Aleti et. al. identified the lack of benchmark support for evaluating architecture optimization problems [2]. In this work, we present two case studies detailing three concrete variable system architectures: a single door power window (Section 4.1), a two door power window system (Section 4.2), and a central door locks system (Section 5). By publishing the cases in full detail, we hope to make the models presented into benchmarks that can be used to evaluate future architecture optimization research.

In their survey, Aleti also identified a lack of support for practitioners when modeling architectures in various languages used in research tools [2]. They proposed using a well-known modeling language, such as UML, for expressing the models and then having some translation to the reasoning language as one way of supporting the practitioners in analyzing the models. In this work we use *Clafer*, a lightweight, general-purpose, structural modeling language to model the case studies. While Clafer is not a well-known language, it has full formal semantics and existing translations to reasoners allowing us to perform automatic analyses and optimization of the models. Also, Clafer models can be translated to UML or SysML (with OCL constraints) and we invite the community to do so. In the meantime, we present an informal background on Clafer in Section 2 so that readers can understand how to build architectural models of their own. In addition, we present a reference model in Section 3 that defines all domain concepts used in modeling the case studies. Using a reference model gives architects a means for modeling different systems with a standardized meta-model, similar to that of the EAST-ADL.

Lastly, in Section 6 we give a set of Clafer patterns for best practice and to improve backend reasoning. While the patterns are for modeling in Clafer, some patterns could be adapted to languages such as UML and SysML.

2 Background

Clafer [3] is a lightweight, general-purpose, structural modeling language that was originally developed for feature modeling [4]. In this section we provide readers with an informal background and basic understanding of the Clafer language and its constructs. The formal semantics of the language can be found in [4].

Clafer currently supports two backend solvers that generate instances from Clafer models. The first solver (not used in this work) is Alloy [9] and the second is Choco3 CSP-based solver called *chocosolver*[1]. Clafer compiler translates an input Clafer model into the input language of each backend solver. We use *chocosolver* to generate both non-optimal and optimal instances for the modeled E/E architectures. Figure 1 shows the workflow of how a Clafer file is compiled into one of the backend model formats then processed to generate instances. *Chocosolver* uses exact algorithms in order to find optimal instances based on single or multiple objectives.

In the following sections, the constructs of a Clafer model are explained along with examples. Additionally, we use the solver

1. https://github.com/gsdlab/chocosolver

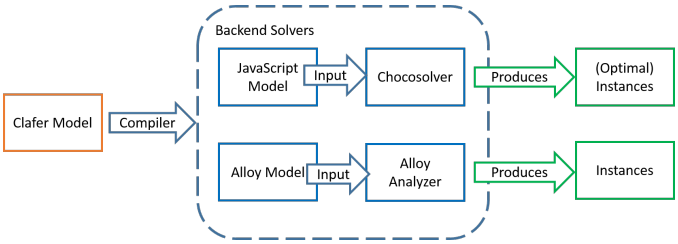

Fig. 1. Overall workflow of Clafer and supported backend solvers.

to show what instances are generated from the corresponding example models.

2.1 Types of clafers and inheritance

In Clafer, a model consists of *clafers*[2]. The name "clafer" comes from the words class, feature, and reference because a clafer provides modeling capabilities of all these language constructs.

Clafers are organized in a containment hierarchy: clafers can contain nested clafers, similarly to how classes can contain other classes and attributes. Each clafer defines a set of instances, similarly to how a class defines a set of its instances (objects). A clafer can be one of two types: abstract or concrete, similarly to classes. A concrete clafer results in an instance being generated, while an abstract one does not. Clafers are also organized in an inheritance hierarchy, like classes, with a restriction that only abstract clafers can be superclafers. Listing 1 shows an example of two abstract clafers `Car` and `ElectricCar` and two concrete clafers `JanesCar` and `JohnsCar`. Comments begin with `//`.

Listing 1. Example for concrete and abstract clafers

```
abstract Car                    // top-level abstract clafer
    engine                         // nested concrete clafer

   // an abstract clafer inheriting from an abstract clafer
abstract ElectricCar : Car
    battery

    // a concrete clafer inheriting from an abstract clafer
JanesCar : ElectricCar

           // error: cannot inherit from a concrete clafer
JohnsCar : JanesCar
```

The containment is specified using indentation, whereas inheritance if specified using colon (`:`). With respect to the containment hierarchy, we say that a clafer is a child of another clafer (e.g., `engine` is a child of `Car`) and a clafer is a parent of another clafer (e.g., `Car` is the parent of `engine`). With respect to the inheritance hierarchy, we say that a clafer is a superclafer of another one (e.g., `Car` is the superclafer of `ElectricCar`, which itself is a superclafer of `JanesCar`). Conversely, a clafer inherits from/is a subclafer of/extends another clafer (e.g., `ElectricCar` inherits from/is a subclafer of/extends `Car`).

2.1.1 *Instance generation*

Given a Clafer model, a backend reasoner can generate all instances of the model in a given finite scope. For example, for the model in Listing 1 (without the incorrect `JohnsCar`) we get the following instance:

```
=== Instance 1 Begin ===

JanesCar
    engine
    battery

--- Instance 1 End ---
```

As we can see, only instances of the concrete clafer `JanesCar` and the inherited concrete clafers are generated. For simplicity, the instances have the same names as the clafers. To be fully explicit one would have to write the following, which we usually ommit (we overload : to mean *instanceOf* relationship).

```
=== Instance 1 Begin ===

JanesCar : JanesCar
    engine : engine
    battery : battery

--- Instance 1 End ---
```

Note, that because inheritance is transitive, the instance `JanesCar` is also an instance of `ElectricCar` and `Car`.

2.2 Clafer multiplicity and group cardinality

Up to this point none of the concepts introduced have allowed for expressing variability in the model. Clafer provides two constructs expressing variability: clafer multiplicity and group cardinality. Clafer multiplicity is a range `n..m` indicating the allowed number of instances of the given clafer with respect to its parent. The most common multiplicities are `1..1` (mandatory) which requires one instance of the given clafer per instance of its parent; and `0..1` (optional) which allows for at most one instance of the given clafer per instance of its parent.

Group cardinalities are used to express variability over a group of children of the given clafer, hence the name. By default, clafers have the group cardinality of `0..*`, that is, they do not impose any constraints on their children.

In Listing 2, we explicitly show the multiplicities of all clafers. By default, all abstract clafers have the multiplicity of `0..*`, which we usually ommit. Also, by default, clafers have the multiplicity of `1..1` unless they are children of a group with cardinality other than `0..*`, in which case the default multiplicity is `0..1`. For example, the clafer `engine` has multiplicity of `1` (short for `1..1`), `wheel` has multiplicity of `4`, and `seat` has the multiplicity of `2..4`. These multiplicities specify that every instance of `Car` contains exactly one instance of `engine`, four instances of `wheel`, and between two and four instances of `seat`, respectively.

In Listing 2, the clafer `transmission` has group cardinality `xor` (keyword for `1..1`) meaning that every instance of `transmission` must contain exactly one instance of its children (`automatic` or `manual`).

Listing 2. Example with explicit clafer multiplicities and a group cardinality

```
abstract Car 0..*
  engine 1
  xor transmission 1
    automatic 0..1
```

2. Throughout this paper if the word Clafer begins with an uppercase letter it refers to the language while a lowercase one refers to the construct

```
    manual ?
  wheel 4
  seat 2..4

abstract Chevy : Car 0..*

JohnsCar : Chevy 1
```

Using the instance generator, two of the 6 instances are:

```
=== Instance 1 Begin ===

JohnsCar
  engine
  transmission
    automatic
  wheel
  wheel$1
  wheel$2
  wheel$3
  seat
  seat$1

--- Instance 1 End ---

=== Instance 4 Begin ===

JohnsCar
  engine
  transmission
    manual
  wheel
  wheel$1
  wheel$2
  wheel$3
  seat
  seat$1
  seat$2

--- Instance 4 End ---
```

Note that in order to distinguish the multiple instances of the same clafer from each other, a suffix `$n` where n is the instance number is added.

Both model instances satisfy all multiplicity and group cardinality constraints; for example, the instance generator will never generate an instance which has both kinds of transmission or neither of them.

2.3 References

The last Clafer construct which allows for variability in models are reference clafers, that is, clafers whose instances can point to instances of other clafers or primitive values.

Listing 3 shows an example of a clafer `Car` that contains two references `driver` and `passenger`, which denote the driver and passengers of the car, respectively. The type of the reference clafer `driver` is `Person`, which means that every instance of the clafer `driver` points to one instance of the clafer `Person`. The multiplicity of `driver` is `1..1`, by default, so every instance of a `Car` will be always connected with one instance of `Person` via an instance of the reference clafer `driver`. The reference clafer `passenger` can have between zero and four instances, each pointing to an instance of `Person`. There are two kinds of reference clafers: set (specified using `->`) and bag (multiset, specified using `->>`). In our example, we do not want to allow the same person to be a passenger more than once, that is, the collection of passengers should be a set and we used `->` to express that constraint.

Listing 3. Example using references

```
abstract Person

abstract Car
  driver -> Person
  passenger -> Person 0..4

MyCar : Car
John : Person
Jane : Person
```

A correct instance of this model then would have `John` or `Jane` as the driver because they are the only clafers of type `Person` in the model. The reference clafer `passenger` points to a set of size 0 to 4 meaning that up to four passengers can be in the car. A correct instance of this model can have no passengers, both John and Jane as passengers, or only John or only Jane as a passenger.

Two of eight possible instances for Listing 3 are:

```
=== Instance 1 Begin ===

MyCar
  driver -> John
John
Jane

--- Instance 1 End ---

=== Instance 8 Begin ===

MyCar
  driver -> Jane
  passenger -> John
  passenger$1 -> Jane
John
Jane

--- Instance 8 End ---
```

We can see two instances of the reference clafer `passenger` (`passenger` and `passenger$1`) pointing to the instances of `John` and `Jane`, respectively. The following instance is incorrect.

```
MyCar
    passenger -> John
    passenger$1 -> John

John
Jane
```

First, a required instance of `driver` is missing, which violates the multiplicity `driver 1..1`; second, the same instance of `John` is a passenger twice, which violates the set constraint.

2.4 Writing Basic Constraints

So far all we have shown with Clafer is the ability to create models with large amounts of variability using references and cardinalities. When modeling real systems and problems a modeler also wants to use constrains in order to restrict the targets of references and the allowed configurations of the model. Additionally, constraints are used when wanting to query a model for one or more specific instances which satisfy the given constraints.

In this section, we give readers some of the basics for writing constraints but for conciseness it is not exhaustive. Readers should refer to other documentation found at http://www.clafer.org (we recommend "Clafer Cheat Sheet"[3] for an informal

3. http://t3-necsis.cs.uwaterloo.ca:8091/ClaferCheatSheet

language reference). One general rule when writing constraints in Clafer, is that, there are no scalars in the language, only sets, and therefore writing a number `1` really means a singleton set containing the number one. Similarly, we cannot directly access clafer instances (since they only exist at instance generation time) and instead we often use singleton sets. For example, the concrete clafer `John : Person 1` is a singleton set which will contain exactly one instance, thus by writing constraints about the clafer `John`, we write constraints about its only instance.

Listing 4 builds on the previous two examples and adds some constraints. The constraints that we want to model are:

- C1: *John is the driver of MyCar*
- C2: *Only Dan and Jane can be passengers in MyCar*
- C3: *The driver should not be in the set of passengers in a car*
- C4: *The total number of passengers with the driver can't exceed the number of seats in a car*

A constraint is a Boolean expression surrounded by square brackets "[ ]". Just like clafers, constraints can be either top-level or nested. Nested constraints must hold for every instance of their context clafer (the clafer they are nested under); consequently an instance of the context clafer cannot exist unless all of its nested constraints hold.

In Listing 4, the constraint C1 is captured in line 10, whereby `.dref` gets the target value of the reference clafer (much like dereferencing a pointer in a C/C++). C2 is then captured in line 11 in the model. Note the subtle differences between C1 and C2 in the constraints and the wording. For C1, set equality (the operator `=`) is used to restrict that the target of the reference *has to be* `John`. In C2, the wording is "can be" so subsetting is used (the keyword `in`). The difference between `in` and `=` is that the former can let the set be any of values to the right of the operation where the latter requires that the set is equal to the right of the operation (note the correct instances that can be generated). In C2, the expression `Jane, Dan` computes a union of instances of Jane and Dan (alternatively, it can be written as `Jane ++ Dan`).

Listing 4. Example using constraints

```
1  abstract Person
2  abstract Car
3    seat 2..4
4    driver -> Person
5    passenger -> Person 0..4
6    [driver.dref not in passenger.dref] // C3
7    [#(passenger, driver) <= #seat] // C4
8
9  MyCar : Car
10   [driver.dref = John] // C1
11   [passenger.dref in (Jane, Dan)] // C2
12 John : Person
13 Jane : Person
14 Dan : Person
```

Constraints C3 and C4 pertain to every instance of `Car` so the constraints are nested under `Car`. For C3 the `not in` enforces that the target of `driver` is not present in the set `passenger`.

C4 uses # to get the size of a set such that one can say the size of the union of sets `passenger` and `driver` is less than or equal to the size of the set `seat`. When using the instance generator 12 instances are found that satisfy the constraints and two of them are as follows:

```
=== Instance 3 Begin ===

MyCar
  seats
  seats$1
  driver -> John
  passenger -> Dan
John
Jane
Dan

--- Instance 3 End ---

=== Instance 11 Begin ===

MyCar
  seats
  seats$1
  seats$2
  seats$3
  driver -> John
  passenger -> Jane
  passenger$1 -> Dan
John
Jane
Dan

--- Instance 11 End ---
```

Constraints are one of the most important aspects of Clafer for modeling systems since restrictions on system configurations, deployments, etc. can be constrained such that only valid systems are synthesized.

2.5 Working with Integers

Clafer and its backends also support working with integers allowing for adding quantitative information to models. Currently, only integer arithmetic is supported, so working with numbers is cumbersome when modeling real systems.

Constraints can be used to model numerical relationships between components and get the total for a set of component. For example, Listing 5 takes the car example and models three features which have an associated cost.

Listing 5. Example using constraints

```
abstract Feature
  cost -> integer
abstract Car
  bluetooth : Feature
  heatedSeats : Feature
  passiveKeyEntry : Feature

MyCar : Car
  [bluetooth.cost = 5]
  [heatedSeats.cost = 10]
  [passiveKeyEntry.cost = 25]
```

The constraints nested under `MyCar` configure an instance of `Car` by giving values to the different feature costs. The dot "." operator navigates from a parent to a child clafer or to the target of a reference. Using the instance generator gives just one correct instance as there is no variability in the model (all references have been tightly constrained and all multiplicities are fixed).

```
=== Instance 1 Begin ===

MyCar
```

```
  passiveKeyEntry
    cost -> 25
  heatedSeats
    cost$1 -> 10
  bluetooth
    cost$2 -> 5

--- Instance 1 End ---
```

2.6 Optimization Objectives

In order to do optimization over qualities of a model, objectives must be defined. In Clafer, optimization objectives are captured by surrounding a numerical expression with « and » as in lines 26 and 27 of Listing 6. In this example the car with features is used but a second quality is added to represent the *comfort level* for a user. Also note that all of the features are optional (denoted by the `?`, meaning cardinality 0..1) such that there is variability among the available features. In order to populate the used feature set of the `Car` the constraint C1 on lines 9, 13, and 17 constrains that `this` (the Feature) is in the set of `parent`'s (the Car's) child `feature`. The constraint C2 on lines 21 and 22 sum the cost and comfort (respectively) of all the present features in the car to be used for the optimization goals.

Listing 6. Example using optimization

```
1  abstract Feature
2    cost -> integer
3    comfort -> integer
4  abstract Car
5    feature -> Feature 2..*
6    totalCost -> integer
7    totalComfort -> integer
8    bluetooth : Feature ?
9      [this in parent.feature] // C1
10     [cost = 5]
11     [comfort = 30]
12   heatedSeats : Feature ?
13     [this in parent.feature] // C1
14     [cost = 30]
15     [comfort = 10]
16   passiveKeyEntry : Feature ?
17     [this in parent.feature] // C1
18     [cost = 40]
19     [comfort = 10]
20
21   [totalCost = sum feature.cost] // C2
22   [totalComfort = sum feature.comfort]
23
24 MyCar : Car
25
26 << minimize MyCar.totalCost >>
27 << maximize MyCar.totalComfort >>
```

The instance generator configured for optimization finds 2 Pareto-optimal instances which are shown below:

```
=== Instance 1 Begin ===

MyCar
  feature -> bluetooth
  feature$1 -> heatedSeats
  totalCost -> 35
  totalComfort -> 40
  bluetooth
    cost -> 5
    comfort -> 30
  heatedSeats
    cost$1 -> 30
    comfort$1 -> 10

--- Instance 1 End ---

=== Instance 2 Begin ===

MyCar
  feature -> bluetooth
  feature$1 -> heatedSeats
  feature$2 -> passiveKeyEntry
  totalCost -> 75
  totalComfort -> 50
  bluetooth
    cost -> 5
    comfort -> 30
  heatedSeats
    cost$1 -> 30
    comfort$1 -> 10
  passiveKeyEntry
    cost$2 -> 40
    comfort$2 -> 10

--- Instance 2 End ---
```

What are the tradeoffs between these two instances? Instance 1 has the lowest cost, while sacrificing the comfort. Instance 2 has the highest comfort at the higher cost. Clafer tools include Clafer Multi-Objective Optimization (MOO) Visualizer, a tool for visualizing and exploring the set of optimal instances of a model, which allows the users to perform tradeoff analysis and find the instances most suitable for their needs [8]. A screenshot from the tool is shown in Figure 2 when visualizing the instances generated from Listing 6.

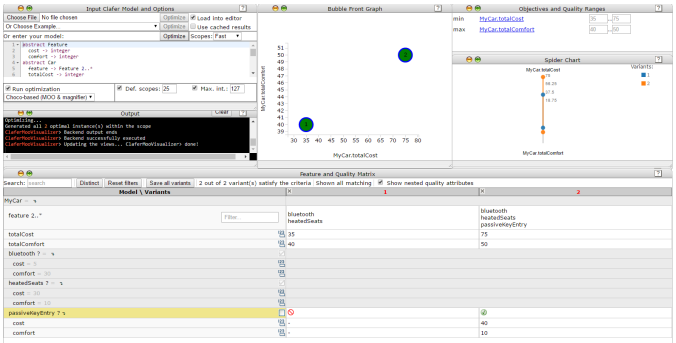

Fig. 2. Screenshot of MOO Visualizer for Listing 6.

3 Reference Model for E/E System Architecture

Before modeling the case studies we define a reference model (or domain model) for early design of automotive E/E architectures. In this section, we define the reference model along with its Clafer encoding. Having a reference model creates a uniform understanding of the components that make up a system and the relationships and rules among the components for modelers.

The reference model described here is similar to the EAST-ADL as it contains multiple layers, each describing the system at a different level of abstraction. Figure 3 shows the reference model we use for capturing the E/E architecture for the early design phases when considering multiple candidate architectures.

The perspectives on the right of Figure 3 augment the system with analysis-task or stakeholder-specific information such as points of variability, latency, and mass. In the following subsections, we consider each layer and perspective of the reference model.

3.1 Feature Model

The feature model captures features of the modeled system. A feature is a broad concept with many interpreted definitions;

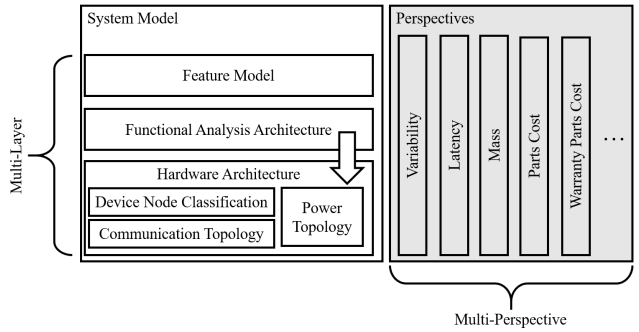

Fig. 3. The reference model used for early design of automotive E/E systems architecture.

however, in this context a feature is a high-level system characteristic relevant to some stakeholder, such as the customer or the user. Features may represent functionality or performance. Some concrete examples could be a passive key entry feature in a door lock system or an automatic closing of a window in a power window.

When encoded in Clafer, a feature and feature model are just two abstract clafers that are used to *type* concrete clafers for readability as no rules or attributes are associated with them. Listing 7

Listing 7. Meta model for feature model

```
abstract FeatureModel
abstract Feature
```

In Sections 4 and 5 we show how to apply these concepts and others, described in the subsequent subsections, to two systems.

3.2 Functional Analysis Architecture

The functional analysis architecture (FAA) captures functions in the system that implement the features described in the feature model. There are two types of functions defined in the FAA: *analysis functions* which model control functions with their inputs and outputs and *functional devices* which capture functions that represent sensors and actuators. Many of the analysis functions will be realized as software components; however, some functions or parts of them may be realized by specialized hardware (such as digital signal processing). Functional devices will typically be realized by hardware sensors and actuators plus additional software, such as device drivers or signal conditioning software.

The FAA also models the communication between functions using function connectors. These connectors represent the transfer of messages at a logical level between two functions. The analysis functions and functional devices are then mapped onto the device nodes which are apart of the hardware architecture (Section 3.3). The function connectors are then deployed to the physical media that are captured in the communication topology (Section 3.4). These deployments from the FAA to the hardware architecture are depicted by the block arrow in Figure 3.

The Clafer encoding of the FAA is more interesting compared to the feature model as it captures the deployments, implementation choice, and some constraints to impose restrictions on the deployments. Listing 8 contains the encoded FAA concepts in Clafer.

Listing 8. Meta model for functional analysis architecture.

```
abstract FunctionalAnalysisComponent                                    1
  deployedTo -> DeviceNode                                              2
  xor implementation                                                    3
    hardware                                                            4
      [deployedTo.type in (EEDeviceNode, SmartDeviceNode)]              5
    software                                                            6
      [deployedTo.type = SmartDeviceNode]                               7
abstract AnalysisFunction : FunctionalAnalysisComponent                 8
abstract FunctionalDevice : FunctionalAnalysisComponent                 9
abstract FunctionConnector                                              10
  sender -> FunctionalAnalysisComponent                                 11
  receiver -> FunctionalAnalysisComponent                               12
  deployedTo -> HardwareDataConnector ?                                 13
    [parent in this.deployedFrom]                                       14
    [(sender.deployedTo.dref, receiver.deployedTo.dref) in              15
        (deployedTo.endpoint.dref)]
  [(sender.deployedTo.dref = receiver.deployedTo.dref) <=>              16
      no this.deployedTo]
```

The deployment of analysis functions and functional devices is captured through the reference from `FunctionalAnalysisComponent` to `DeviceNode` on line 2. Additionally, implementation choice is captured in `FunctionalAnalysisComponent` on lines 3-6. Based on the implementation, deployment is restricted by a set of rules; the constraints on lines 4 and 6 capture these rules which are further explained in the next subsection. Section 3.3 explains each of the device node types.

The deployment of function connectors to the communication topology media is encoded by line 13 where the constraints nested under impose constraints on the deployment. Function connectors have an optional deployment since if the two functions are deployed to the same device node there is no need for a physical communication medium (captured by line 16). The constraint on line 15 enforces that the medium being used by the function connector does in fact connect the two device nodes in which the communicating functions are deployed.

3.3 Device Node Classification

The device node classification captures the device nodes in the architecture and their properties. A device node is a piece of physical hardware such as a sensor, actuator, ECU (electronic control unit), or battery. One of the properties is the type of device node, which our reference model we define three concrete types as follows:

- **Smart**: A device node that can be programmed with one or more analysis functions or functional devices which are implemented in software. Examples would be an ECU or a hardware device with embedded microcontroller.
- **Electric/Electronic (E/E)**: A device node that can not be programmed with executable software but rather implements some hardware functionality described by a functional device or analysis function that is implemented in hardware. Examples would be a sensor or actuator.
- **Power**: A device node that generates, stores, or relays power to other device nodes. In this paper, we do not allow a power device node to have any logic associated with it

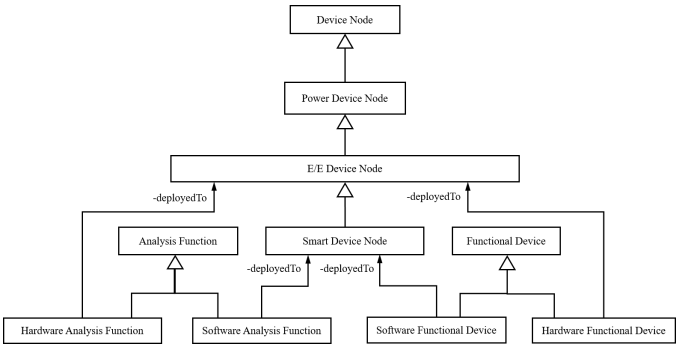

Fig. 4. Class diagram showing classification of device nodes and functions.

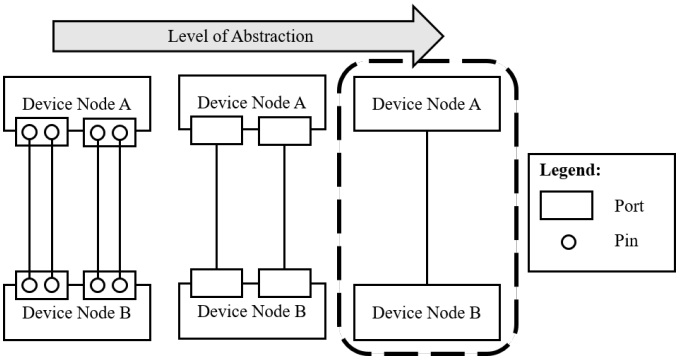

Fig. 5. Modeling of data connectors at different levels of abstraction. The lowest level is shown on the left where connectors are modeled between pins, the middle shows connectors between ports, the right is the highest level where two devices nodes are connected via a connector. The dashed selection shows the level of abstraction used for data and power connectors.

Listing 9. Meta model for device node classification

```
abstract DeviceNodeClassification
enum DeviceNodeType = SmartDeviceNode | EEDeviceNode |
     PowerDeviceNode
abstract DeviceNode
  type -> DeviceNodeType
```

(i.e., no functional devices or analysis functions deployed to it). An example would be a battery or fuse box.

Figure 4 shows a class diagram of the classification for device nodes, analysis functions, and functional devices. It also shows the allowed deployments of the different functions to the various device node types. Notice from the figure that the classification is a lattice, that is, a smart device also has E/E and power capabilities, whereas an E/E device also has power capability but not smart capability. Thus, a functional device or analysis function implemented in hardware can be deployed to either an E/E or a smart device node.

Listing 9 shows the encoding of the device node and its types in Clafer. The reference clafer `type` nested under `DeviceNode` has a cardinality of 1..1 meaning that only one of the device node types is selected in an instance. Additionally, the constraints on lines 4 and 6 of the FAA listing (Listing 8) capture the allowed deployments of functions to device nodes.

3.4 Communication Topology

The next part of the hardware architecture is the communication topology, which defines the physical media that function connectors use. The following connectors make up the topology:

- **Discrete data connectors**: A connector used to indicate the status of a binary input, such as a switch. Figure 5 shows the different levels of abstraction for modeling hardware connectors. We chose to model them at the highest level (shown in the dashed box) to reduce model complexity, thus we don't model ports or pins on a device node.

- **Analog data connectors**: A connector which sends an analog signal (as opposed to digital); typically encoded as voltage amplitude. This is also an abstract connector similar to the discrete data connector.

- **Bus connectors**: An abstract connector between two or more device nodes in which all connected nodes may pass

Listing 10. Meta model for communication topology

```
abstract HardwareConnector                                          1
                                                                    2
abstract HardwareDataConnector : HardwareConnector                  3
  endpoint -> DeviceNode 2..*                                       4
  deployedFrom -> FunctionConnector 1..*                            5
    [this.deployedTo = parent]                                      6
                                                                    7
abstract DiscreteDataConnector : HardwareDataConnector              8
  [#(endpoint) = 2]                                                 9
abstract AnalogDataConnector : HardwareDataConnector                10
  [#(endpoint) = 2]                                                 11
                                                                    12
abstract BusConnector : HardwareDataConnector                       13
  [endpoint.type = SmartDeviceNode]                                 14
  xor type                                                          15
    LowSpeedCAN                                                     16
    HighSpeedCAN                                                    17
    LIN                                                             18
    FlexRay                                                         19
                                                                    20
abstract LogicalBusBridge : HardwareDataConnector                   21
  [endpoint.type = SmartDeviceNode]                                 22
  bus -> BusConnector 2                                             23
```

messages; these are typically serial buses such as CAN and LIN.

In order to capture the transfer of messages between two buses, a bridge is defined between two bus connectors. Listing 10 shows the encoding of the different communication topology elements. The `deployedFrom` reference on line 5 models the inverse relationship back to the function connector in which the cardinality is 1..* to enforce that at least one function connector is using the connector to be present.

3.5 Power Topology

The last part of the hardware architecture we consider is the power topology, which models how device nodes are connected with power and in which we consider two types of power connectors:

- **Load Power Connector**: An abstraction for lower gauge wires that distributes higher power (around 12 V) to device nodes. These connectors are often used to power motors, heaters, and lights.

Listing 11. Meta model for power topology

```
abstract HardwareConnector
abstract PowerConnector : HardwareConnector
  source -> DeviceNode
  sink -> DeviceNode

abstract LoadPowerConnector : PowerConnector
abstract DevicePowerConnector : PowerConnector
```

- **Device Power Connector**: An abstraction for higher gauge wires that distributes lower power (around 5 V) to device nodes. These connectors are often used to power smart devices.

Listing 11 shows the encoding of the power topology elements in Clafer. The connectors are modeled as unidirectional by having a source and sink which is typical when capturing power distribution.

3.6 Variability Perspective

The variability perspective is captured in the reference model not by adding any additional concepts but rather using the built-in mechanisms of Clafer (multiplicity, group cardinality, references). A concrete example of this is the deployment of function to device nodes. In Listing 8, a reference `deployedTo` is defined which allows the target of the reference to be any device node in the model. Additionally, variability can be expressed for any instantiated reference model type by using cardinalities. Sections 4 and 5 give concrete examples of modeling variability for the case studies.

3.7 Mass, Parts Cost, & Warranty Parts Cost Perspectives

The mass, parts cost, and warranty parts cost perspectives differ from the variability perspective since they require additional constructs being defined in the reference model instead of being built into the language. The three perspectives are similar since all are only captured at the hardware architecture level. In this reference model, only the unit cost of hardware components is considered.

Listing 29 in Appendix A contains the complete reference model with all of the annotations for the mass, cost, warranty cost, and latency perspectives.

The mass of an architecture is measured by the mass of the device nodes and the physical connectors. In the reference model, a mass property is given to a device node, captured in line 55 of Listing 29. The mass of the connectors is measured by having a coefficient representing the mass per unit length (captured by the integer clafers on lines 124-148) of a wire and having a property for the length of a connector. Additionally for the discrete and analog connectors the mass is multiplied by the number of function connectors deployed to it since each one needs a single wire to communicate. This relationship is captured by the constraints on lines 84 and 89 in Listing 29.

The cost of an architecture is modeled in the same way as the mass. For warranty cost, only the device nodes are considered since only reliability is captured for them. The metric used for reliability is part per million (ppm) and the warranty cost is calculated by taking the replacement cost of a device node multiplied by the ppm value.

3.8 Latency Perspective

Unlike mass, cost, and warranty cost, the latency perspective is not only captured at the hardware architecture but also in the FAA. We do this through *timing chains* — a sequence of executing functions with their communication as defined in the FAA. With the latency of a system being modeled, timing requirements such as *end-to-end latency* — the time taken to reach a target function from some source function — or *input synchronization latency* — the maximum difference in time taken to reach a target function from more than one source function. The case studies in the following sections give concrete examples of creating the chains, but before one must model the timing properties of the reference model element that have a role. The following list contains each of the elements that contribute to the latency of a system:

- Functional device & analysis function: a base latency is specified by the user. If the function is implemented in software the base latency represents the latency a function is expected to take at some base ECU speed factor. If the function is implemented in hardware the base latency is simply the hardware specification latency.
- Function connector: a size is specified by the user to represent the size of the message being sent. Then based on whether the function connector is deployed and to what type of hardware connector, the latency is calculated using the transfer rate of the hardware connector.
- Device node: a speed factor can be given to the device node which affects the resulting latency of deployed analysis functions.
- Discrete/analog data connector: the latency is assumed to be zero in our model.
- Bus connector: the transfer rate is dependent of the type of bus which then affects the deployed to function connectors.

The assumptions that are made when using this timing model are as follows:

- All deployments to nodes are considered to be schedulable.
- The deployment of functions to nodes is such that when any two functions communicate via a bus there is no blocking time.

These assumptions greatly simplify the timing analysis done during optimization. Additionally, during early design, engineers are interested in estimating timing budgets that will be refined and further analyzed in more detailed design revisions. Therefore, such a model is sufficient for very early exploration. Listing 29 in Appendix A shows the complete reference model with the latency perspective added.

4 Power Window Case Study

The first case study we present in this report is the E/E architecture for a power window system in a car. First a single driver side door system is presented and then a second system for the front passenger side door power window is added with communication between the two. This case study has been developed and presented in two previous works, one by Murashkin [7] and one by Akhtar [1]. The case study presented in this work is an extension of the former while the latter uses the case study for a comprehensive design analysis, not just exploration.

The reason for choosing the power window as a case study was due to the fact it was self-contained and not overly complex. The material for the architecture designs and sources of variability was obtained through publicly available service manuals from companies such as Nissan, Infinity, BMW, and GM. The quality attribute information for cost and mass was obtained from OEM part supply websites and Amazon.com. Latency quality attributes were mainly created artificially using typical values as a base line (based on domain expertise). Lastly, the reliability attributes (ppm) were obtained from standard values and handbook calculations. In the last part of this section, we give details on how the values were formatted as inputs to the model.

4.1 Single Door: Driver

4.1.1 Feature Model

The features that are considered for the single door power window are as follows:

- `Basic Up/Down` The basic operation of the power window. When the switch is held in the up position the window retracts until closed or release of the switch. There is an identical reverse operation for holding the switch down.
- `Express` The express *down* feature in which when a user pushes the button down into an express position, the window opens until the window is completely open without the user having to continue pressing down the switch.
- `Express Up` The express *up* feature is similar to that of the express down except for the reverse operation. Additionally, if an object is detected in the window travel path then the window should stop and retract to be fully open again.

Figure 6 is a feature model which organizes features into a hierarchy and indicates their variability as Kang et. al. proposed in [6]. Note that for `expressUp` to exist the feature `express` must be present. This relationship can be captured using nesting in Clafer and is highlighted in Listing 12. Additionally, using clafer multiplicities the features `express` and `expressUp` are made optional.

Listing 12. Clafer feature model for single door power window

```
DWinSysFM : FeatureModel
  basicUpDown : Feature
  express : Feature ?
    expressUp : Feature ?
```

4.1.2 Functional Analysis Architecture

The functional analysis architecture for the power window needs to support two sets of functions, one for the basic operation and one for detecting obstacles for the express up feature. Figure 7 shows the FAA for the system using a graphical domain-specific language (the legend is shown in Figure 8) that can be translated to Clafer.

The functions present in the FAA and their allowed implementations are as follows:

- `WinSwitch`: A sensor functional device that reads the switch position requested by the user. *Allowed implementation(s): hardware*
- `WinArbiter`: An analysis function that arbitrates which incoming signal should be sent to the controller. For the driver window this component is not needed since only one input is present, however we still model it. *Allowed implementation(s): hardware, software*
- `WinControl`: The main control logic of the power window. It takes the switch position request, the current value from the motor and if an object is present (if pinch detection is required) and translates it to a command to send to the motor. *Allowed implementation(s): software*
- `WinMotor`: The actuator that takes the desired command from the controller and translates it to moving the motor to close and open the window. *Allowed implementation(s): hardware*
- `CurrentSensor`: A sensor placed on the motor to measure the current being pulled by the motor. This is used to ensure that the motor does not "stall" by continuing to move in a direction in which it can no longer move (i.e. the window is fully open or closed). *Allowed implementation(s): hardware*
- `PositionSensor`: A sensor to detect the current position of the window in the path of travel. *Allowed implementation(s): hardware*
- `PinchDetection`: Another piece of control logic that takes the position sensor reading and determines if an object is present or not. *Allowed implementation(s): software*

The `PinchDetectionFAA` is a functional analysis architecture nested inside the driver power window FAA and it is optional (since the express up feature is). By putting the `PositionSensor` and `PinchDetection` and the two function connectors in the FAA it allows for those components to be removed when `Pinch`

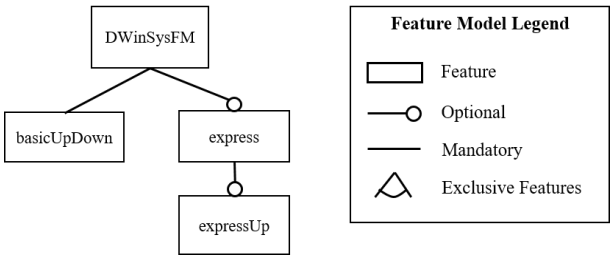

Fig. 6. The feature model for a single door power window

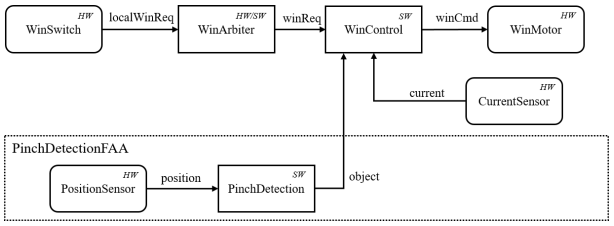

Fig. 7. The functional analysis architecture for a single door power window. The "HW" or "SW" in upper right corner of a function indicates the implementation choice in hardware or software respectively. "HW/SW" indicates the function can be implemented in either software or hardware.

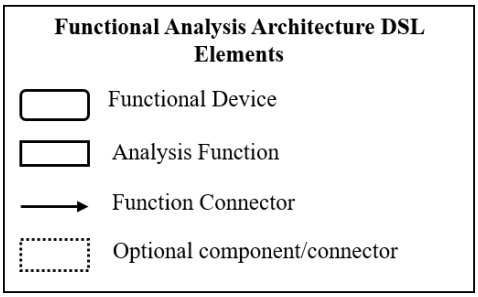

Fig. 8. Legend for the graphical domain-specific language (DSL) for describing the FAA for E/E architectures

`DetectionFAA` is not present.

Listing 13 shows a snippet of the FAA encoded using Clafer for the single door power window. Note the use of nesting for the components inside of `PinchDetectionFAA`.

Listing 13. Snippet of Clafer model for single door functional analysis architecture

```
1  DWinSysFAA : FunctionalAnalysisArchitecture
2    WinSwitch : FunctionalDevice
3      [implementation.hardware]
4      [baseLatency = 20]
5    WinController : AnalysisFunction
6      [implementation.software]
7      [baseLatency = 2]
8    WinMotor : FunctionalDevice
9      [implementation.hardware]
10     [baseLatency = 10]
11   ...
12   winReq : FunctionConnector
13     [sender = WinArbiter && receiver = WinController]
14     [messageSize = 1]
15   winCmd : FunctionConnector
16     [sender = WinController && receiver = WinMotor]
17     [messageSize = 2]
18   PinchDetectionFAA : FunctionalAnalysisArchitecture ?
19     PinchDetection : AnalysisFunction
20       [implementation.software]
21       [baseLatency = 2]
22     PositionSensor : FunctionalDevice
23       [implementation.hardware]
24       [baseLatency = 10]
25     object : FunctionConnector
26       [sender = PinchDetection && receiver = WinController]
27       [messageSize = 2]
28     ...
29     [DWinSysFAA.PinchDetectionFAA <=> DWinSysFM.express.
           expressUp]
```

The constraint on line 29 of Listing 13 expresses equivalence between the `PinchDetectionFAA` and the feature `expressUp` in the feature model, which ensures that pinch detection functionality is present if and only if the feature express up is present.

4.1.3 Device Node Classification

The device node classification consists of local nodes, which belong to the system, and remote nodes which are shared among many systems. The local nodes that belong to the driver door power window system are:

- `Switch`: The physical switch sensor that is present on the door for the driver to control their window. *Allowed type(s): smart, electric/electronic*
- `Motor`: The motor that moves the window armature to open and close the window. *Allowed type(s): smart, electric/electronic*
- `Door Module`: An optional ECU that is housed inside the door. *Allowed type(s): smart*
- `Door Inline`: An interconnect that connects the wiring from the main body harness to the door harness. *Allowed type(s): power*

The remote nodes in the system are:

- `BCM` (Body Control Module): The main ECU that houses much of the body control software. *Allowed type(s): smart*
- `EC` (Electric Center): The main fuse box that is the primary source of power *Allowed type(s): power*

A snippet of the device node classification encoding in Clafer is shown in Listing 14. In order to model the shared components, a clafer `Car` is defined which houses any remote components to the model. Then, the concrete `BCM` and `EC` are declared in `Car`. Inside the definition of the device node classification for the driver system, two references are defined to point to the central components. Using a reference instead of local declaration represents the fact that the driver device node classification has to be able to access the BCM and EC; however, since the BCM reference is optional, the driver system might not need it.

Listing 14. Snippet of Clafer model for single door device node classification

```
abstract SwitchNode : DeviceNode                                    1
  numSwitches -> integer                                            2
                                                                    3
DWinSysDN : DeviceNodeClassification                                4
  BCM -> DeviceNode ?                                               5
  EC -> DeviceNode                                                  6
  Switch : DeviceNode                                               7
    [type in (SmartDeviceNode, EEDeviceNode)]                       8
    [mass = 173]                                                    9
    [cost = 110]                                                    10
    [replaceCost = 110]                                             11
    [if (type in SmartDeviceNode) then (ppm = 50) else (ppm         12
          = 10)]
    [(type in SmartDeviceNode) => (speedFactor = 10)]               13
    [numSwitches = 2]                                               14
  Motor : DeviceNode                                                15
    [type in (SmartDeviceNode, EEDeviceNode)]                       16
    ...                                                             17
  DoorInline : DeviceNode                                           18
    [type = PowerDeviceNode]                                        19
    ...                                                             20
  DoorModule : DeviceNode ?                                         21
    [type = SmartDeviceNode]                                        22
    ...                                                             23
  [BCM = Car.BCM]                                                   24
  [EC = Car.EC]                                                     25
                                                                    26
Car                                                                 27
  BCM : DeviceNode ?                                                28
     [type = SmartDeviceNode]                                       29
```

```
  ...
EC : DeviceNode
  [type = PowerDeviceNode]
  ...
```

Listing 14 also contains a sub-type of `DeviceNode`, namely `SwitchNode`. This subtype allows the modeling of a general type of switch panel on a car and denotes the number of switches on the panel. Thee number of switches could have been captured by multiplicities but the solver exhibits a performance slow down. Patterns like this are discussed in more detail in Section 6.

4.1.4 Power Topology

The power topology for the power window models two types of power, load and device. The device power topology is quite straight forward; if a device node is smart then it must have a connection from the EC to the device node using a device power connector. The topology is modeled by defining optional power connectors from the EC to the door inline and then from the door inline to the respective nodes. Figure 9 shows the power topology for the single door window and the two sets of power connectors. Figure 10 is a legends for understanding the graphical symbols used to model the hardware architecture elements.

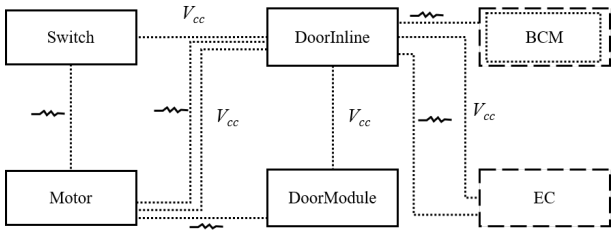

Fig. 9. The power topology for a single door power window. The inside dotted box for the BCM denote that it is an optional remote device node.

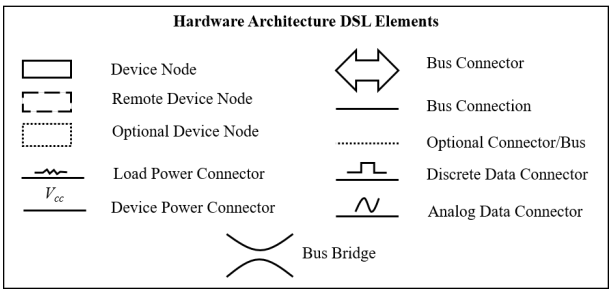

Fig. 10. Legend for the graphical domain specific language (DSL) for describing the hardware architecture for E/E architectures.

In order to have the correct device power connectors present in the architecture the following rules are applied:

- **Rule 1:** A device power connector between a local device node and the door inline must exist if the device node is smart.

- **Rule 2:** A device power connector between the EC and the door inline must exist if there is at least one device power connector leaving the door inline.

The second power connector that is of interest is the load power connector. Load power is required by the motor in order to move the window glass. The configuration of the load power is driven by the deployment of the `WinControl` analysis function. This is because the `WinControl` function acts as a driver for the motor. Therefore, a load power connector must be defined from each of the device nodes that can be smart to the motor (as seen in Figure 9). Secondly, the load power must get from the EC to the device node that has the `WinControl` function deployed. This results in 4 concrete topologies for the load power which can be described by the following rules:

- **Rule 3:** If the control is deployed to the motor then a load power connector must be present from the door inline to the motor and the EC to the door inline.

- **Rule 4:** If the control is deployed to the switch or door module, then a load power connector must exist from the door inline to the local deivce node, from the local device node to the motor, and the EC to the door inline.

- **Rule 5:** If control is deployed to the BCM then a load power connector must exist from the door inline to the motor, BCM to the door inline, and EC to the BCM.

Listing 15 shows a snippet of the power topology encoded in Clafer with the quality attribute values as well. Comments are used to show which constraint capture what rules for defining the allowed configurations. A group cardinality `xor` is used in order to model the exclusive configurations outlined in rules 3, 4, and 5.

Listing 15. Snippet of Clafer model for single door power topology

```
DWinSysPT : PowerTopology
  dn -> DWinSysDN
  MotorLoadPowerWire : LoadPowerConnector
    [sink = dn.Motor]
  SwitchLoadPowerWire : LoadPowerConnector ?
    [source = dn.DoorInline && sink = dn.Switch]
    [length = 45]
  DoorModuleLoadPowerWire : LoadPowerConnector ?
    [source = dn.DoorInline && sink = dn.DoorModule]
    [length = 35]
  DoorInlineLoadPowerWire : LoadPowerConnector
    [sink = dn.DoorInline]

  xor MotorLoadPowerConfig
    SwitchIsMotorDriver //R4
      [MotorLoadPowerWire.source = dn.Switch]
      [MotorLoadPowerWire.length = 40]
      [DoorInlineLoadPowerWire.source = dn.EC.dref]
      [DoorInlineLoadPowerWire.length = 40]
      [SwitchLoadPowerWire && DoorInlineLoadPowerWire && no
           DoorModuleLoadPowerWire]
    DoorModuleIsMotorDriver //R4
      ...
    BCMIsMotorDriver //R5
      ...
    MotorIsMotorDriver //R3
      ...

  switchInlineDP : DevicePowerConnector ?
    [source = dn.DoorInline && sink = dn.Switch]
    [length = 45]
  motorInlineDP : DevicePowerConnector ?
  ...
  [switchInlineDP <=> (dn.Switch.type in SmartDeviceNode)]
       //R1
  [motorInlineDP <=> (dn.Motor.type in SmartDeviceNode)] //
       R1
  [ha.pt.inlineECDP <=> some(motorInlineDP, switchInlineDP,
        doorModuleInlineDP)] //R2
```

4.1.5 Communication Topology

The communication topology contains two mediums for communication, a bus and discrete connectors. For the driver door power window there is a single bus that allows for communication between the BCM and the local device nodes. The discrete connectors, as stated in the reference model, represent bundles of wires between two nodes. Unlike the power topology, a single connector is modeled between two communicating nodes so the door inline does not play any part in routing the connectors and it is abstracted away. Making this assumption allows for simpler models which accommodates early design.

Figure 11 shows the communication topology for the power window. Note that there is no possible discrete connector between the door module and BCM due to the assumption made that if a device is smart (when both are) they must use the bus.

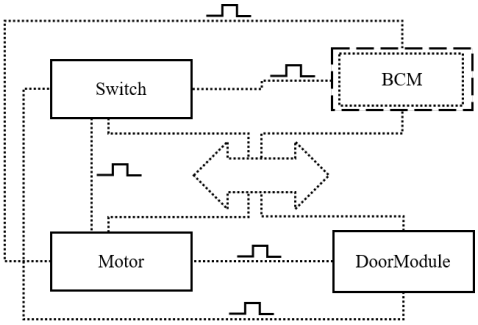

Fig. 11. The communication topology for a single door power window

Listing 16 shows a snippet of the corresponding encoding in Clafer along with the quality attributes. The endpoints constraint on line 7 restricts the possible device nodes that *can be* connected to the bus. The endpoint constraints for the discrete wires explicitly gives the two endpoints for the connector such that the solver along with the constraint on line 16 in Listing 8 can synthesize the correct connectors needed. The deployment rules for the function connectors to the communication topology is then shown in the following section.

Listing 16. Snippet of Clafer model for single door communication topology

```
1  DWinSysCT : CommTopology
2    dn -> DWinSysDN
3
4    logicalLowSpeedBus : BusConnector ?
5      [type.LIN || type.LowSpeedCAN]
6      [length = 70]
7      [endpoint in (dn.Motor, dn.Switch, dn.DoorModule, dn.
           BCM.dref)]
8
9    logicalSwitchMotorDisc : DiscreteDataConnector ?
10     [length = 40]
11     [endpoint = (dn.Switch, dn.Motor)]
12
13   logicalSwitchBCMDisc: DiscreteDataConnector ?
14   ...
```

4.1.6 Deployment

To complete the single door power window case study the last part needing to be modeled is the deployment of the functional analysis architecture onto the hardware architecture. The deployment consists of only rules (or constraints) that detail what the allow mappings of functions to nodes and function connectors to mediums. The rules that need to be modeled are as follows:

- **Rule 1:** The analysis functions can be deployed to any of the local or central nodes.
- **Rule 2:** The `WinMotor`, `CurrentSensor`, and `Position Sensor` must be deployed to the `Motor`.
- **Rule 3:** The function connectors can be deployed to any of the possible discrete connectors or the bus.

The rules can then be encoded using constraints in Clafer; a snippet of model is shown in Listing 17. Additionally on lines 12-15 the power topology configuration is constrained based on the deployment of the controller as detailed earlier.

Note the use of implication "=>" in the constraints pertaining to the pinch detection elements. This is such that the deployment constraints don't restrict that the pinch detection FAA must be present (these constraints are conditional on the presence of the pinch detection FAA).

Listing 17. Snippet of Clafer model for single door deployment

```
DWinSysDpl : Deployment                                          1
  fa -> DWinSysFAA                                               2
  ha -> DWinSysHA                                                3
                                                                 4
  [fa.WinArbiter.deployedTo.dref in (ha.dn.BCM.dref, ha.dn.      5
       Switch, ha.dn.Motor, ha.dn.DoorModule)]
  [fa.PinchDetectionFAA => (fa.PinchDetectionFAA.                6
       PinchDetection.deployedTo.dref in (ha.dn.BCM.dref, ha
       .dn.Switch, ha.dn.Motor, ha.dn.DoorModule))]
  [fa.WinSwitch.deployedTo.dref = ha.dn.Switch]                  7
  [fa.PinchDetectionFAA => (fa.PinchDetectionFAA.                8
       PositionSensor.deployedTo.dref = ha.dn.Motor)]
                                                                 9
  [(fa.WinController.deployedTo.dref = ha.dn.Switch) <=> ha      10
       .pt.MotorLoadPowerConfig.SwitchIsMotorDriver]
  [(fa.WinController.deployedTo.dref = ha.dn.Motor) <=> ha.      11
       pt.MotorLoadPowerConfig.MotorIsMotorDriver]
  [(fa.WinController.deployedTo.dref = ha.dn.BCM.dref) <=>       12
       ha.pt.MotorLoadPowerConfig.BCMIsMotorDriver]
  [(fa.WinController.deployedTo.dref = ha.dn.DoorModule)         13
       <=> ha.pt.MotorLoadPowerConfig.
       DoorModuleIsMotorDriver]
                                                                 14
  [(fa.localWinReq.deployedTo.dref in (ha.ct.                    15
       logicalLowSpeedBus, ha.ct.logicalSwitchMotorDisc, ha.
       ct.logicalSwitchBCMDisc, ha.ct.logicalMotorBCMDisc,
       ha.ct.logicalSwitchDoorModuleDisc, ha.ct.
       logicalMotorDoorModuleDisc))]
...                                                              16
```

4.2 Two Door: Driver & Front Passenger

In this section, the previous system is scaled up to a two door system. When building the second system, we observed that the passenger door was very close in structure to the driver. So in the first subsection this commonality is addressed and abstract clafers are used to generalize the core elements. The following section then gives the specifics for how the case study extends the common core to model the passenger door.

4.2.1 Generalizing the Core Elements

The passenger door is almost identical to the driver door with the exception of an added function and some additional communication. Thus, the generalized elements turn out to be the

same as the driver system. Starting with the feature model, the Clafer model in Listing 12 can be modified to be abstract and have two separate concretizations of it, as shown in Listing 18.

Listing 18. Generalized feature model and two concretizations

```
abstract WinSysFM : FeatureModel
  basicUpDown : Feature
  express : Feature ?
    expressUp : Feature ?

DWinSysFM : WinSysFM
PWinSysFM : WinSysFM
```

The same can be done for the remaining layers presented in the single door power window. Listing 30 in Appendix B shows the full generalized architecture.

4.2.2 Extending for the Passenger System

The passenger door system requires some extensions to the base model which are not present in the driver door system. Starting with the feature model, it only makes sense for the passenger to have less features than the driver. The following rules can be derived then:

- **Rule 1:** The passenger should not have the feature `express` if the driver does not have it.
- **Rule 2:** The passenger should not have the feature `expressUp` if the driver does not have it.

These rules turn into constraints captured in the specialization of the passenger feature model in Listing 19.

Listing 19. Clafer feature model for two door system

```
DWinSysFM : WinSysFM
PWinSysFM : WinSysFM
  [express => DWinSysFM.express]
  [express.expressUp => DWinSysFM.express.expressUp]
```

The functional analysis architecture for the passenger system needs to include the additional switch and connector which models the driver side switch controlling the passenger window. Since the generalized FAA contains all other functionality, the only thing the passenger concretization needs is a functional device to represent the driver side switch and a function connector from the switch to the arbiter. Figure 12 shows such architecture and the corresponding Clafer model is shown in Listing 20. Observe that the arbiter can now always prefer the input from the driver-side switch, which will override the passenger's input.

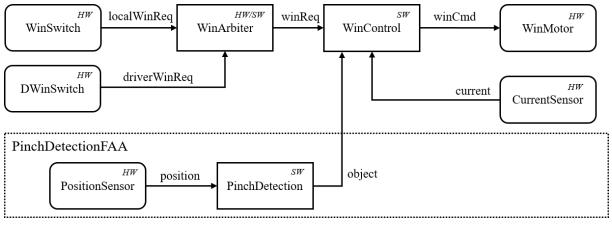

Fig. 12. The functional analysis architecture for a passenger door power window system

Listing 20. Clafer functional analysis architecture for a two door system

```
DWinSysFAA : WinSysFAA
  [DriverWinSys.DWinSysFM.express.expressUp <=> DWinSysFAA.
       PinchDetectionFAA]

PWinSysFAA : WinSysFAa
  [PassengerWinSys.PWinSysFM.express.expressUp <=>
       PWinSysFAA.PinchDetectionFAA]
  DWinSwitch : FunctionalDevice
    [deployedTo.hardware]
    [baseLatency = 10]
  driverWinReq : FunctionConnector
    [sender = DWinSwitch && receiver = WinArbiter]
    [messageSize = 1]
```

The only difference for the passenger device node classification is an addition of a reference to the driver side device node `Switch`. This is shown on lines 113 and 114 of Listing 31 in Appendix C.

For the power topology, there are no extensions for either system from the generalized topology. However, the communication topology requires some additional connectors which is shown in Figure 13.

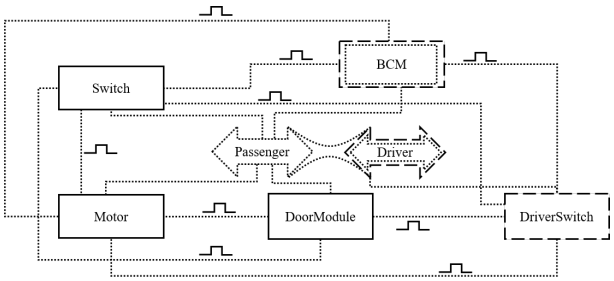

Fig. 13. The communication topology for the passenger door power window system. The "Driver" bus is shown as a reference for the bridge between the two buses. It is assumed that any node connected to the "Passenger" bus can use the bridge to send a message over the bridge to the "Driver" bus.

Listing 21 shows the concretization of the two common communication topologies and the extension for the passenger. Note the use of a logical bus bridge in order to model the communication between the passenger local bus and the driver local bus.

Listing 21. Clafer communication topology for two door system

```
DWinSysCT : WinSysCT
  ...
PWinSysCT : WinSysCT
  logicalDoorBusBridge : LogicalBusBridge ?
    [bus = (PWinSysCT.logicalLowSpeedBus, DWinSysCT.
         logicalLowSpeedBus)]
    [gatewayTransferTimePerSize = 10]
    [endpoint in (PWinSysDN.Motor, PWinSysDN.Switch,
         PWinSysDN.DoorModule, PWinSysDN.BCM.dref, DWinSysDN
         .Motor, DWinSysDN.Switch, DWinSysDN.DoorModule)]
  logicalDriveSwitchPassSwitch : DiscreteDataConnector ?
    [length = 260]
    [endpoint = (PWinSysHA.PWinSysDN.DSwitch.dref,
         PWinSysHA.PWinSysDN.Switch)]
  logicalDriveSwitchPassMotor : DiscreteDataConnector ?
    [length = 260]
    [endpoint = (PWinSysHA.PWinSysDN.DSwitch.dref,
         PWinSysHA.PWinSysDN.Motor)]
  logicalDriveSwitchPassDoorModule : DiscreteDataConnector
       ?
    [length = 250]
    [endpoint = (PWinSysHA.PWinSysDN.DSwitch.dref,
         PWinSysHA.PWinSysDN.DoorModule)]
  logicalDriveSwitchBCM : DiscreteDataConnector ?
    [length = 85]
    [endpoint = (PWinSysHA.PWinSysDN.DSwitch.dref,
         PWinSysHA.PWinSysDN.BCM.dref)]
  ...
```

The added communication topology and additional functional device requires the deployment for the passenger system to be extended as well. Listing 22 gives a snippet of the Clafer model for the two deployments.

Listing 22. Clafer functional analysis architecture for two door system

```
DWinSysDpl : WinSysDpl
  ...
PWinSysDpl : WinSysDpl
  [PWinSysFA.DWinSwitch.deployedTo.dref = PWinSysHA.
       PWinSysDN.DSwitch.dref]
  [PWinSysFA.dWinReq.deployedTo.dref in (
    PWinSysHA.PWinSysCT.logicalDoorBusBridge,
    PWinSysHA.PWinSysCT.logicalDriveSwitchPassSwitch,
    PWinSysHA.PWinSysCT.logicalDriveSwitchPassMotor,
    PWinSysHA.PWinSysCT.logicalDriveSwitchPassDoorModule,
    PWinSysHA.PWinSysCT.logicalDriveSwitchBCM)]
  ...
```

4.3 Quality Attributes & Timing Analysis

4.3.1 Normalizing the Quality Attributes

As stated earlier, Clafer and *chocosolver* can currently reason over integers only. Therefore the gathered values for mass, cost, etc. must be scaled and rounded to provide meaningful numerical results while working with integers.

The first step is to choose some unit base for the different qualities such as grams for mass. Next, all the values are observed in the converted unit base and a check is made for any values that are less than 1. If any such values do exist then they must be scaled upwards in order to increase precision. For example, if a transfer rate was $0.0001ms/byte$ then it could be converted to $1\mu s/byte$. Then, any other qualities that are summed with the new unit base would need to be converted as well (i.e. all of the data connection latencies due to buses). If a unit is not scalable such as dollars, one can change the unit to be *dollars per thousand*.

An issue that arises with working with *chocosolver* is that the total of any summation or multiplication can't overflow. Thus, care needs to be expressed when changing the base of numbers such that the numbers don't become too large to work with.

4.3.2 End-to-End Latency Constraints

For both power window systems there exist two end-to-end latency constraints that are of interest to model and constrain; they are as follows:

- **TR1:** The time it takes for the switch to read the pressed value to the time the motor begins actuation must be less than $750ms$.
- **TR2:** The time it takes for the position sensor to read a value to the time the motor begins actuation must be less than $500ms$.
- **TR3:** The time it takes the switch and position sensor to read the data and be sent to the control must be synchronized to less than $50ms$.

The requirements are captured by creating two *timing chains*, one from `WinSwitch` to `WinMotor` and another from `PositionSensor` to `WinMotor`. These chains are captured in lines 12-26 for the driver power window in Listing 31 in Appendix C. The requirements are then modeled in lines 29-32 of the same Listing.

For TR3 a input synchronization constraint must be modeled using Clafer which the snippet of interest is shown in Listing 23. The use of the set `min` and `max` are use to get the smallest and largest latencies from the three chains.

Listing 23. Snippet of Clafer to show the encoding of TR4.

```
ControlInputDifference -> integer
[ControlInputDifference = (max(SwitchToControlLatency.dref,
      PositionSensorToControlLatency.dref)
- min(SwitchToControlLatency.dref,
     PositionSensorToControlLatency.dref))]
```

Also of interest to engineers is the margin that exists between the end-to-end latency requirement and the actual end-to-end latency. This is a good optimization parameter when optimizing latency as it increases the robustness of the system when the margin is maximized by allowing more room for error. Lines 35-41 capture the margins for the driver power window in Listing 31.

5 Door Locks Case Study

The second case study we present in this report is a E/E architecture for a central door locks system. Only the locking control for the driver and three passenger doors was considered; not the trunk or fuel lid. In this case study, we considered features such as remote key access (where a remote control is used to unlock and lock the car) and passive key entry (where a key fob is used to lock and unlock without touching the remote).

We chose this system in order to build on the power window by modeling a second system in the body domain. With two such systems, future work can be done in incorporating the two systems together and exploring trade-offs when sharing components.

Similar to the power window case study, the material was gathered from OEM service manuals such as GM, BMW, and Nissan. The domain knowledge for the passive key entry was obtained from various articles and suppliers.

5.1 Feature Model

The door locks system contains many more features than the power window which is expected since it is more complex. Figure 14 shows the feature model for the door locks system and the variability that exists. The feature model layer alone encodes 16 variants of the system.

The description for the features is as follows:

- `Basic`: The basic operation of the door locks system using the inside lock switch and cylinder key switch. It also includes unlocking all doors when the car is in park.
- `Speed Smart Lock`: The feature that will lock the car when a certain threshold speed has been reached.

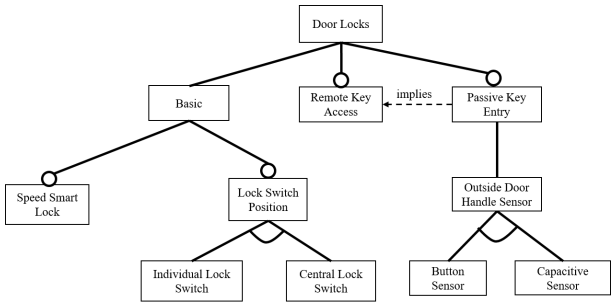

Fig. 14. The feature model for a central door locks system

- `Lock Switch Position`: The feature that dictates where the inside lock switch will be located. The possibilities are a lock switch on the driver and front passenger door or a shared switch in the center (i.e., the console).
- `Remote Key Access (RKA)`: The feature that allows a car to unlock and lock when a remote control button is pressed.
- `Passive Key Entry (PKE)`: The feature that allows a car to unlock/lock when the door handle is touched/button pressed without touching the key fob.
- `Outside Door Handle Sensor`: The feature for what type of sensor is used to detect that the user wishes to lock/unlock the car using PKE. The capacitive sensor is a touch device that detects when a user has grabbed/touched the door handle. The button sensor is a physical push button placed on the outside door handle of the car.

Listing 24 shows the feature model encoded in Clafer. Note that instead of using an `xor` grouping for the `LockPositionSwitch`, a cardinality 0..1 was chosen to reduce the feature model size.

Listing 24. Clafer feature model for door locks

```
DLockFM : FeatureModel
  Basic : Feature
    IndividualLockSwitch : Feature ?
    SpeedSmartLock : Feature ?
  RKA : Feature ? // Remote Key Access
  PKE : Feature ? // Passive Key Entry
    xor OutsideDoorHandleSensor
      ButtonSensor : Feature
      CapacitiveSensor : Feature
  [PKE => RKA]
```

5.2 Functional Analysis Architecture

The functional analysis architecture for the door locks is split into three fragments based on the features (`Basic`, `RKA`, and `PKE`). Figure 15 shows the basic functionality. For readability purposes, the common functional components that are identical across the four (or two) doors are grouped together with a single connector. Additionally, the connectors are not named for readability.

The following is a detailed description of the basic functional components (only one from a grouping is explained):

- `[Central]DoorLockButton`[4] A door lock button placed in the respective location to lock and unlock all doors. *Allowed implementation(s): hardware*
- `SpeedSensor` A functional device that reads the current speed of the car which is needed for the feature `SpeedSmartLock`. *Allowed implementation(s): hardware*
- `GearPositionSensor` A functional device that detects the current gear position of the car (Park, Reverse, Drive, etc.). *Allowed implementation(s): hardware*
- `[Driver]DoorLockMotor` A functional device that locks and unlocks the door. *Allowed implementation(s): hardware*
- `[Driver]DoorCylinderSwitch` A functional device the detects the position of the cylinder switch when a key is inserted and turned. *Allowed implementation(s): hardware*
- `[Driver]DoorContact` A functional device that detects if the door is ajar or closed. *Allowed implementation(s): hardware*
- `[Driver]DoorLockSensor` A functional device that detects the current position of the lock for the door (i.e. the door is locked or unlocked). *Allowed implementation(s): hardware*
- `DoorLockControl` The control function that reads in the sensor inputs and gives the appropriate signal to the motors for locking/unlocking the doors. *Allowed implementation(s): software*

The second functional analysis fragment is for the RKA feature which is shown in Figure 16. The `DoorLockControl` analysis function is replicated as it is the only shared component with the basic FAA fragment.

The functions for the RKA FAA are as follows:

- `CentralRFAntenna` A radio frequency (RF) antenna that receives signals from the key remote and are sent to a receiver/transciever for decoding. *Allowed implementation(s): hardware*
- `CentralRFReceiver` A receiver that decodes the antenna signal. *Allowed implementation(s): hardware*
- `IDAuthentication` The analysis function that takes the decoded signal from the receiver and determines if the key that sent the signal has permission to unlock/lock the car. *Allowed implementation(s): software*

The last functional analysis fragment, shown in Figure 17, contains the functionality for the PKE feature. The functions are described as follows:

- `[Driver]SideOutsideLFAntenna` A low frequency (LF) antenna that broadcasts a signal generated by the transmitter to the outside perimeter of the [driver] side door. *Allowed implementation(s): hardware*
- `[Driver]SideLFTransmitter` Transmitter that encodes a signal from the control to send to the antenna. *Allowed implementation(s): hardware*
- `Inside[Front]LFAntenna` A low frequency antenna that broadcasts a signal generated by the transmitter inside

4. We use the square brackets to denote the variable portion of the name

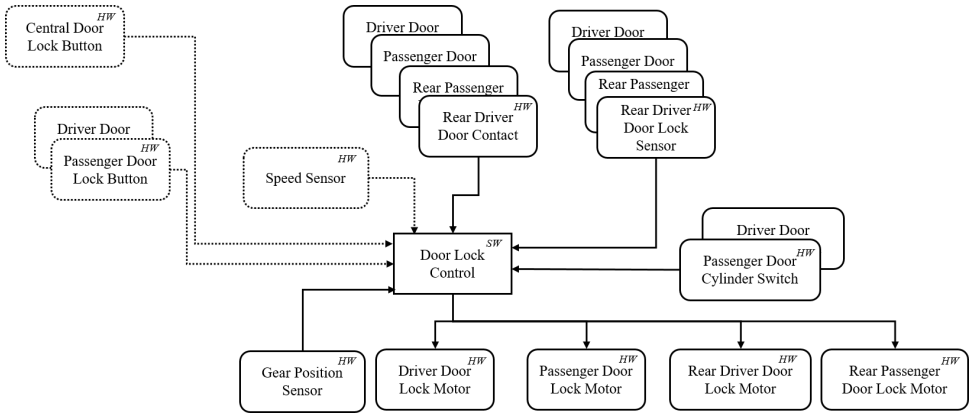

Fig. 15. The functional analysis architecture for the basic features in the door locks system

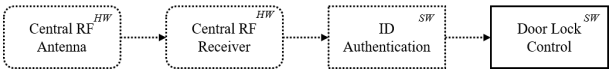

Fig. 16. The functional analysis architecture for the RKA fragment in the door locks system

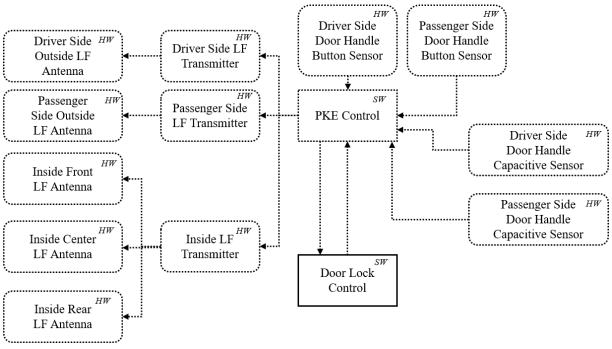

Fig. 17. The functional analysis architecture for the PKE fragment in the door locks system

[front] region of the car. *Allowed implementation(s): hardware*

- `InsideLFTransmitter` Transmitter that encodes a signal from the control to send to the inside antennas. *Allowed implementation(s): hardware*

- `[Driver]SideDoorHandleButtonSensor` A functional device that detects when the button on the outside door handle has been pressed. *Allowed implementation(s): hardware*

- `[Driver]SideDoorHandleCapacitiveSensor` A functional device that detects when the handle is grabbed/touched. *Allowed implementation(s): hardware*

- `PKEControl` The control function that takes the inputs and determines what messages to broadcast to the antennas and dictates to the door lock control what the lock/unlock request is. *Allowed implementation(s): software*

The three fragments then can be encoded using Clafer with their links to the feature model. A snippet of the functional analysis architecture for the door locks is shown in Listing 25. The complete FAA can be found in Listing 32 in Appendix D.

Listing 25. Clafer functional analysis architecture snippet for door locks

```
abstract DoorLockFAA : FunctionalAnalysisArchitecture
  // --  Core Components --//
  // Cylinder Switches
  DriverDoorCylinderSwitch : FunctionalDevice
    [implementation.hardware]
    [baseLatency = 10]
  ...
  // Door Lock Control
  DoorLockControl : AnalysisFunction
    [implementation.software]
    [baseLatency = 4]
  ...
  // -- Optional Fragments/Components --//
  // Speed Smart Lock FA Components
  SpeedSmartLockFA : FunctionalAnalysisArchitecture ?
    SpeedSensor : FunctionalDevice
      [implementation.hardware]
      [baseLatency = 10]
    speed : FunctionConnector
      [messageSize = 1]
      [sender = SpeedSensor && receiver = DoorLockControl]
  // Central or Distributed Lock Switch
  xor DoorLockButtonFA : FunctionalAnalysisArchitecture
    IndividualLockSwitchFA : FunctionalAnalysis
      DriverDoorLockButton : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
      PassDoorLockButton : FunctionalDevice
      ...
    CentralLockSwitchFA : FunctionalAnalysis
      CentralLockButton : FunctionalDevice
      ...
  RemoteKeyAccessFA : FunctionalAnalysisArchitecture ?
    CentralRFAntenna : FunctionalDevice
    ...
  PassiveKeyEntryFA : FunctionalAnalysisArchitecture ?
    DriverOutsideLFAntenna : FunctionalDevice
    ...
    xor OutsideDoorHandleSensor
      ButtonSensor
        DriverDoorButtonSensor : FunctionalDevice
        ...
      CapacitiveSensor
        DriverDoorCapacitiveSensor : FunctionalDevice
        ...

DLockFAA : DoorLockFAA
  [DoorLockButtonFA.IndividualLockSwitchFA <=> DLockFM.
       Basic.IndividualLockSwitch]
  [SpeedSmartLockFA <=> DLockFM.Basic.SpeedSmartLock]
  [RemoteKeyAccessFA <=> DLockFM.RKA]
  [PassiveKeyEntryFA <=> DLockFM.PKE]
  [PassiveKeyEntryFA.OutsideDoorHandleSensor.ButtonSensor
       <=> DLockFM.PKE.OutsideDoorHandleSensor.ButtonSensor]
  [PassiveKeyEntryFA.OutsideDoorHandleSensor.
       CapacitiveSensor <=> DLockFM.PKE.
       OutsideDoorHandleSensor.CapacitiveSensor]
```

Similarly to the power window, a functional analysis architec-

ture fragment is defined for an optional feature. This allows for a group of functions and connectors to be made optional by just having the fragment being optional. In addition to just defining fragments, the door locks FAA is also using `xor` groupings to model exclusive functions that stem from the feature model which was not done for the power window.

5.3 Device Node Classification

Similar to the functional analysis architecture, the device node classification splits the local device nodes into three fragments. In addition to the local nodes, there exists four remote nodes which are shared with other systems. Two of the four are the electric center and BCM which were described in the power window case study. A detailed description of each of the local device nodes and their allowed types is as follows:

- `[Driver]SideDoorLockMotorAssembly`: A hardware device that contains the motor and various sensors for the locking mechanism. *Allowed type(s): electric/electronic.*
- `[Driver]LockPowerSwitch`: A physical switch that can lock or unlock the car. *Allowed type(s): electric/electronic.*
- `CentralRFAntennaModule`: A hardware module that contains an RF antenna, a receiver, and some processing power. *Allowed type(s): smart.*
- `Transmitter`: A transmitter hardware module. *Allowed type(s): electric/electronic.*
- `PassiveKeyModule`: An ECU dedicated for PKE functions. *Allowed type(s): smart.*
- `[Driver]DoorButtonHandleModule`: A hardware module embedded in the door handle that contains an LF antenna and a button sensor. *Allowed type(s): electric/electronic.*
- `[Driver]DoorCapacitiveHandleModule`: A hardware module embedded in the door hanlde that contains a LF antenna and a capacitive sensor. *Allowed type(s): electric/electronic.*
- `Inside[Front]LFAntenna`: An LF antenna hardware device. *Allowed type(s): electric/electronic.*

In contrast to the power window case study, the device nodes are of fixed types. The only variability that exists at the device node level is the presence. The remote device nodes that are unique to the door locks case study are as follows:

- `Transmission Control Module (TCM)` A device node responsible for handling any control associated with the transmission. For this case study it is used to house the gear position sensor functional device. *Allowed type(s): smart.*
- `Combination Meter` A hardware device responsible for measuring the current speed of the car. *Allowed type(s): smart.*

The Clafer encoding of the device node classification is different than in the power window since for the door locks it is split into multiple fragments to group the nodes associated with the features. A snippet of the Clafer is shown in Listing 26. Like the FAA, `xor` groupings are used to model the exclusive sets of device nodes that follow from the feature model.

Listing 26. Clafer device node classification snippet for door locks

```
abstract DoorLockDN : DeviceNodeClassification
  //-- Core Device Nodes --//
  DriverDoorLockMotorAssembly : DeviceNode
  ...
  TCM -> DeviceNode
  BCM -> DeviceNode
  EC -> DeviceNode
  // -- Optional Device Nodes --//
  // Speed Smart Lock Nodes
  CombinationMeter -> DeviceNode ?
  // Central or Individual Lock Nodes
  xor DoorLockButtonDN
    IndividualLockSwitchDN : DeviceNodeClassification
      DriverLockPowerSwitch : DeviceNode
      ...
    CentralLockSwitchDN
      CenterLockPowerSwitch : DeviceNode
      ...
  RemoteKeyAccessDN : DeviceNodeClassification ?
    CentralRFAntennaModule : DeviceNode
    ...
  PassiveKeyEntryDN : DeviceNodeClassification ?
    Transmitter : DeviceNode ?
    ...
    xor OutsideDoorHandleSensor
      ButtonSensor
        DriverDoorButtonHandleModule : DeviceNode
        ...
      CapacitiveSensor
        DriverDoorCapacitiveHandleModule : DeviceNode
        ...
    InsideFrontLFAntenna : DeviceNode
    ...
DLockDN : DoorLockDN
  [BCM = Car.BCM]
  [TCM = Car.TCM]
  [EC = Car.EC]
  [CombinationMeter => CombinationMeter = Car.
       CombinationMeter]
  [DoorLockButtonDN.IndividualLockSwitchDN <=> DLockFM.
       Basic.IndividualLockSwitch]
  [CombinationMeter <=> DLockFM.Basic.SpeedSmartLock]
  [RemoteKeyAccessDN <=> DLockFM.RKA]
  [PassiveKeyEntryDN <=> DLockFM.PKE]
  [PassiveKeyEntryDN.OutsideDoorHandleSensor.ButtonSensor
       <=> DLockFM.PKE.OutsideDoorHandleSensor.ButtonSensor]
  [PassiveKeyEntryDN.OutsideDoorHandleSensor.
       CapacitiveSensor <=> DLockFM.PKE.
       OutsideDoorHandleSensor.CapacitiveSensor]
```

5.4 Power Topology

In the device node classification, the inline that was modeled in the power window was not for the door locks. An inline was not modeled in this case study as there was no interesting power configurations that were affected by the door inline as there were in the power window. The power topology for the door locks did not contain the configurations in which the power window did due to the `DoorLockControl` analysis function being always deployed to the `BCM`.

The power topology, shown in Figure 18 contains no variability itself as the only optional components are driven by the selection of the device nodes used (i.e. there is no variability in the configuration of the topology given a set of device nodes). The only unique piece to the door locks is the need for a device power connector when using a capacitive sensor.

The full Clafer encoding of the power topology is shown in Listing 32 in Appendix D.

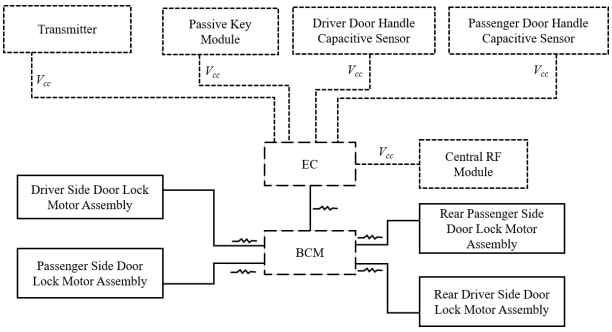

Fig. 18. The power topology for the door locks system

5.5 Communication Topology

As opposed to the power topology, the communication topology is interesting and also larger than the power window. In order to manage the size and complexity, it is split into 3 figures. First, Figure 19 shows the communication topology fragment for the basic and RKA features. Two buses are needed: the first is a high speed bus (which would be either high speed CAN or FlexRay) which handles safety critical nodes; the second is a low speed bus (either low speed CAN or LIN) which handles the non-critical body domain nodes. This low speed bus is assumed to be routed through the main body harness and not to the doors as was so in the power window.

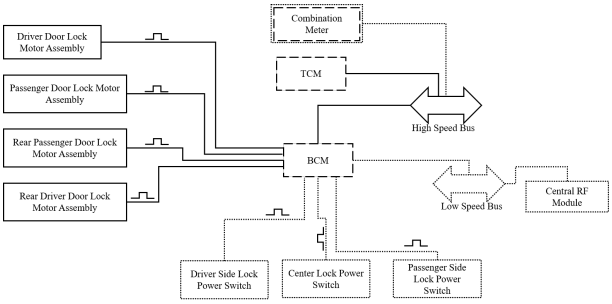

Fig. 19. The communication topology for the door locks system basic and RKA fragments.

The other two figures (Figures 20 and 21) show two views which can be overlaid to describe to complete communication architecture of the PKE feature. The first view shows the possible communication connectors needed when the BCM acts as a LF transmitter (i.e. the `DriverSideLFTransmitter`, `PassengerSideLFTransmitter`, and `InsideLFTransmitter` are all deployed to the `BCM`). The second view, shows when the transmitter device node is used instead.

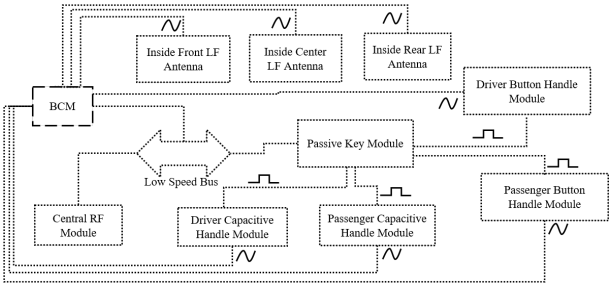

Fig. 20. The communication topology for the door locks system PKE fragment which uses the BCM as a transmitter.

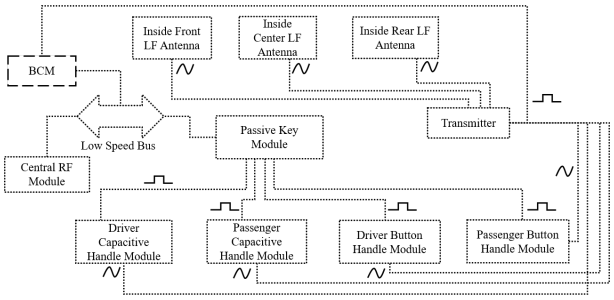

Fig. 21. The communication topology for the door locks system PKE fragment which uses the transmitter device node.

The full Clafer encoding of the communication topology is shown in Listing 32 in Appendix D.

5.6 Deployment

To complete the door locks case study, the last part to be modeled in the deployment. As in the power window, the deployment for the door locks is just a set of constraints that restricts the set of possible targets for the functional analysis components. The unique part of the door locks encoding is using fragments to split up the deployment by features. It allows us to drop the repeated use of implication (denoted by `=>`) in the constraints. For example, the Clafer snippets shown below are identical for modeling the deployment constraints. Listing 27 shows how the constraint is written using an implies (as in the power window model), whereas Listing 28 shows how the constraints can be nested inside a fragment.

Listing 27. Using implies in constraint expression to handle conditional deployment.

```
[DLockFM.PKE => (fa.PassiveKeyEntryFA.DriverLFTransmitter.
     deployedTo in (ha.dn.PassiveKeyEntryDN.Transmitter, ha.
     dn.BCM.dref))]
```

Listing 28. Nesting the constraint under a fragment to handle conditional deployment.

```
PassiveKeyEntryDpl ?
  [fa.PassiveKeyEntryFA.DriverLFTransmitter.deployedTo in (
       ha.dn.PassiveKeyEntryDN.Transmitter, ha.dn.BCM.dref)]
[PassiveKeyEntryDpl <=> DLockFM.PKE]
```

We present other modeling techniques in Section 6. The full Clafer encoding for the deployment is shown in Listing 32 in Appendix D.

5.7 Quality Attributes & Timing Analysis

Similar to that of the power window case study, the values for the different quality attributes were obtained from various sources then normalized so that they could be used in integer operations. Also, for the door locks case study end-to-end timing constraints were constructed for three requirements as well as an input synchronization constraint as follows:

- **TR1:** The time it takes for the individual switch to read the pressed value to the time the motor begins to move to lock/unlock the door must be less than $500ms$.

- **TR2:** The time it takes for the central switch to read the pressed value to the time the motor begins to move to lock/unlock the door must be less than $500ms$.

- **TR3:** The time it takes for the handles sensor the read the user request to the time the motor begins to lock/unlock the door must be less than $750ms$.

- **TR4:** The time it takes the door contacts, door sensors, and the lock switch to read the data and send to the control must by synchronized to less than $50ms$.

TR1 through TR3 are end-to-end timing constraints that are quite similar to that of the power window. Lines 700-731 in Listing 32 in Appendix D shows the required Clafer to encode the requirements.

6 Techniques for Modeling Complex Systems in Clafer

In this section, we provide and discuss some techniques that arose while developing the case studies. For each of the techniques, we associate one or more tags with it in order to classify what the technique is for. The tags are as follows:

- *Clafer*: A technique that is only useful in the Clafer modeling language and can not be applied to other languages such as UML or SysML. We leave the details for how patterns can be applied to SysML or UML out of this report.

- *Performance*: A technique that has performance implications when using the Clafer backend *chocosolver*.

- *Readability*: A technique that is used to improve the readability of the model to either the modeler or an outside reader.

- *Best-Practice*: A technique that is consider "best-practice" when creating models as it improves the general understanding of the model.

Technique 1: Partition the Clafer model When starting a new Clafer model for modeling a complex system such as an automotive E/E architecture it is good to divide up the model as follows:

1) Meta-model elements: The abstract Clafers that make up the meta model (or types) for the models being created.

2) Generic systems & components: An abstract model of systems and/or components that are re-used among several concrete model instances.

3) Concrete system model: The model that instantiates from the previous two sections.

Tag(s): Best-Practice, Readability

Technique 2: Every concrete clafer should have a "type" In Clafer, every abstract or concrete clafer is a *thing*. What makes Clafer into a domain-specific modeling language is when the modeler defines a "type" (an abstract clafer) for every clafer. This is accomplishing by having every concrete clafer inheriting from an abstract one. *Tag(s): Clafer, Best-Practice, Readability*

Technique 3: Use nesting to show sub-systems and/or fragments in an architecture or topology Nesting can capture a group of elements for a subsystem or fragment under a common parent (i.e., a system or fragment). Making the parent optional makes all nested elements optional. *Tag(s): Clafer, Best-Practice, Performance*

Technique 4: Use references to model components that physically reside outside the system and instantiation to model ownership References should be used when components are used inside a particular system but do not belong to that subsystem. For example:

```
Car : System
  BCM : DeviceNode
    ready ?
PowerWindow : System
  BCM -> DeviceNode

[PowerWindow.BCM = Car.BCM]
```

Modeling this way shows that the `Car` owns the `BCM` and resides in that system but the `BCM` is also used inside the `PowerWindow` system.

Using a constraint to assign `PowerWindow.BCM`; however, does not allow referring to the clafer `ready` of the `BCM` because the type of the reference is `DeviceNode`. If possible, it is recommended to specify `Car.BCM` as the type of the reference as follows:

```
PowerWindow : System
  BCM -> Car.BCM
  [ BCM.ready ]
```

As we can see, now the clafer `ready` is accessible in constraints.

Tag(s): Best-Practice

Technique 5: Don't model any relationships between two references An example of this would be when modeling connectors in a system; a connector should not be defined between reference components such as the BCM and electric center. The reason for this is so that if multiple people are working on the model concurrently, there are not multiple or conflicting relationships. *Tag(s): Best-Practice*

Technique 6: Define an general environment to own components outside the system For any component that is not owned to any of the systems being modeled, they should be declared inside some general environment. For example the environment could be `Car` which declares environment component `BCM`.

Tag(s): Clafer, Best-Practice

Technique 7: Nest constraints under concretized clafers, not references This technique stops writing a constraint in the power window system, instead of the car environment, which determines that the BCM has a cost of $100 or has to be `smart`. The reasoning for this technique is the same as for not modeling relationships between 2 references. *Tag(s): Clafer, Best-Practice*

Technique 8: Only constrain deployment targets in one direction When creating a deployment reference, it is sometimes beneficial to creating an inverse of the reference (i.e. a *deployed to* and *deployed from* relationship). When an inverse is present, care should be taken to only constrain deployment

targets in one direction and use general constraints to populate the inverse such as in the following model:

```
abstract Function
  deployedTo -> Device
    [parent in this.deployedFrom]
abstract Device
  deployedFrom -> Function *
    [this.deployedTo = parent]
```

This is so that over time no inconsistencies arise in the relationships. *Tag(s): Clafer, Best-Practice*

Technique 9: Use upper bounds on cardinalities when possible If possible it is always better to give an upper bound for a clafer cardinality (i.e. don't use $*$) in order for the scope analyzer to calculate a more accurate scope (which could lead to finding more correct instances or reducing the size of the problem). *Tag(s): Clafer, Performance*

Technique 10: Avoid instantiating multiple copies of a component in the model When deciding how to represent a set of components that are the same, the modeler should see if there is a way they can achieve the same result using multiple references to a single core component instead of multiple references to multiple identical components. A concrete example of this is abstracting a set of wires into a connector like when creating 6 identical `logicalSwitchMotor` discrete wires and deploying a single command to each instead of having all commands be deployed to a single discrete connector. *Tag(s): Best-Practice, Performance*

Technique 11: Use an implication when writing constraints involving optional clafers When writing constraints there will be some clafers that may not need to satisfy the constraint because the component was not used. In order to maintain the variability one should make sure to have an constraint that says "if we have component A then constraint C should hold". For example:

```
[fa.PinchDetectionFA => (fa.PinchDetectionFA.PositionSensor
     .deployedTo.ref = ht.dn.Motor)]
```

Tag(s): Clafer

Technique 12: Consider making a subtype of a reference model component to express performance friendly properties In some cases there may be a reference model component that needs to be abstracted such that instantiating unnecessary clafers is avoided. A concrete example would be the switches for the driver power window. The switch panel includes two switches (one fore the driver and one for the passenger). One way to model this would be to use a cardinality on the definition `Switch` as follows:

```
Switch : DeviceNode 2
  [mass = 10]
// Total mass = 20g
```

However by adding another clafer, it will increase the complexity of the model and the search space. An alternative would be to subtype device node and have a `numSwitches` child integer clafer that represents the number of switches the device has. This then can be use in adjusting the mass and cost of the device node.

```
abstract SwitchNode : DeviceNode
  numSwitches -> integer
  baseMass -> integer
  [mass = baseMass*numSwitches]
Switch : SwitchNode
  [baseMass = 10]
// Total mass = 20g
```

Tag(s): Performance

Technique 13: Do not sum/multiply up quality attributes for referenced components Only components that are instantiated should be included when summing up the quality attributes for components to ensure that a component is not added or multiplied twice. *Tag(s): Clafer, Best-Practice*

Technique 14: Care should be expressed when assigning quality attribute values to abstractions Care should be taken because it restricts the user from assigning a different value when extending the abstraction. *Tag(s): Best-Practice*

Technique 15: References to optional clafers need to be optional When a clafer is instantiated as being optional then any reference to that clafer must also be made optional. This is to maintain the optionality of the instantiated component. However, if one always wants the reference to exist then it should not be made optional and should know the instantiated component will always exist. *Tag(s): Clafer*

Technique 16: Modeling Many to Many Relationships The following snippet shows how to model many-to-many relationships in Clafer.

```
abstract LogicalBus
  realizedBy -> PhysicalBus 1..*
    [parent in this.realizes.ref]

abstract PhysicalBus
  realizes -> LogicalBus 1..*
    [parent in this.realizedBy.ref]
```

Tag(s): Clafer

Technique 17: Modeling Performance Friendly Inverse Relationships Many times an inverse relationship is not queried and thus the set is not needed but the cardinality is of importance. This can be captured through a constraint instead of declaring a new clafer and reference in order to be much more performance friendly. Consider the refactoring of the example in the previous technique:

```
abstract LogicalBus
  realizedBy -> PhysicalBus 1..*

abstract PhysicalBus
  [some lb : LogicalBus | this in lb.realizedBy]
```

Tag(s): Clafer, Performance

Technique 18: Alternatives for modeling exclusive property sets There are two alternatives for modeling an exclusive property. For example in the power window our device nodes have a property stating if they are smart, electric/electronic, or power.

Alt 1: Using abstract clafers and references The benefit of modeling this way is the reduced number of clafers needed. However if the types have constraints associated with them it may not be best.

```
enum DeviceNodeType = SmartDeviceNode | EEDeviceNode |
     PowerDeviceNode
```

```
abstract DeviceNode
  type -> DeviceNodeType

Motor : DeviceNode
  [type in (SmartDeviceNode, EEDeviceNode)]
```

Alt 2: Using abstract clafers and references The benefit of modeling this way is a constraint can be nested under each of the children of `type`. The downside of this alternative is the increased number of clafers.

```
abstract DeviceNode
  xor type
    smart
    ee
    power

Motor : DeviceNode
  [type.smart || type.ee]
```

Tag(s): Clafer, Performance

7 Conclusion

In this technical report, we presented a reference model and used it to model two automotive E/E architectures: a power window and a door locks system. Additionally, we demonstrated how Clafer could be used to model the two case studies and capture the variability at each of the different layers. This work contributes two case studies that can be used in future evaluations of optimization algorithms as a benchmark for automotive architecture.

In future work, the case studies could be extended by modeling the two body domain systems together as well as extending the power window to four doors. Additionally, the case studies could be extended with quality attributes by adding resource constraints for memory and computing power as well as more optimization objectives such as reliability.

References

[1] Zubair Akhtar. Model based automotive system design: A power window controller case study. Master's thesis, University of Waterloo, 03 2015. https://uwspace.uwaterloo.ca/handle/10012/9215, last accessed Jun 23, 2016.

[2] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *Software Engineering, IEEE Transactions on*, 39(5):658–683, May 2013.

[3] Kacper Bąk and Michał Antkiewicz. Clafer: A lightweight modeling language. Web site. http://www.clafer.org/, last accessed Jun 23, 2016.

[4] Kacper Bąk, Zinovy Diskin, Michał Antkiewicz, Krzysztof Czarnecki, and Andrzej Wąsowski. Clafer: Unifying class and feature modeling. *Software and Systems Modeling*, 2014. The final publication is available at Springer via DOI.

[5] EAST-ADL Association. *EAST-ADL domain model specification, version V2.1.12.* http://east-adl.info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf, last accessed Jun 23, 2016.

[6] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231, last accessed Jun 23, 2016.

[7] Alexandr Murashkin. Automotive electronic/electric architecture modeling, design exploration and optimization using clafer. Master's thesis, University of Waterloo, 09 2014. https://uwspace.uwaterloo.ca/handle/10012/8780, last accessed Jun 23, 2016.

[8] Alexandr Murashkin, Michał Antkiewicz, Derek Rayside, and Krzysztof Czarnecki. Visualization and exploration of optimal variants in product line engineering. In *SPLC*, 2013.

[9] Software Design Group, MIT. Alloy: A language and tools for relational models. Web site. http://alloy.mit.edu/alloy/, last accessed Jun 23, 2016.

Appendix A
Reference Model

Listing 29. Complete reference model

```
//--------------------------- Meta-Model Elements -------------------------//
// Meta-Model Elements - This section contains all meta-model elements that
// are used to model a general automotive E/E architecture. Most of the
// elements are adapted from the EAST-ADL v2 specification.

// System is our version of the EAST-ADL "System Model". The two are similiar
// but have a couple differences:
//  - The implementation level is ignored.
//  - The analysis level and design level have been combined into the
//      architecture
abstract System
abstract FeatureModel
abstract Architecture
abstract FunctionalAnalysis
abstract HardwareArchitecture
abstract DeviceNodeClassification
abstract CommTopology
abstract PowerTopology
abstract Deployment

// Some generic "types" of Clafer's. Some types don't have properties but
// are rather used for readability for a user
abstract Feature

abstract FunctionalAnalysisComponent
    deployedTo -> DeviceNode
    xor implementation
        hardware
            [latency = baseLatency]
            [deployedTo.type in (EEDeviceNode, SmartDeviceNode)]
        software
            [latency = baseLatency*deployedTo.speedFactor]
            [deployedTo.type in SmartDeviceNode]
    baseLatency -> integer // [ms]
    latency -> integer // [ms]
abstract AnalysisFunction : FunctionalAnalysisComponent
abstract FunctionalDevice : FunctionalAnalysisComponent
abstract FunctionConnector
    sender -> FunctionalAnalysisComponent
    receiver -> FunctionalAnalysisComponent
    deployedTo -> HardwareDataConnector ?
        [parent in this.deployedFrom]
        [(sender.deployedTo.dref, receiver.deployedTo.dref) in (deployedTo.endpoint.dref)]
    [(sender.deployedTo.dref = receiver.deployedTo.dref) <=> no this.deployedTo]
    latency -> integer // [us]
    messageSize -> integer // [byte]
    [if (deployedTo) then (latency = messageSize*deployedTo.transferTimePerSize) else (latency = 0)]


enum DeviceNodeType = SmartDeviceNode | EEDeviceNode | PowerDeviceNode

abstract DeviceNode
    type -> DeviceNodeType
    speedFactor -> integer // unitless
    mass -> integer // [g]
    cost -> integer // [dollar]
    ppm -> integer // unitless
    replaceCost -> integer // [dollar]
    warrantyCost -> integer = ppm*replaceCost // [dollar per million]
    [(type in (PowerDeviceNode, EEDeviceNode)) => (speedFactor = 0)]

// Hardware Connection Mediums
abstract HardwareConnector
    length -> integer // [cm]
    mass -> integer // [mg]
    cost -> integer // [dollar per thousand]
abstract PowerConnector : HardwareConnector
    source -> DeviceNode
    sink -> DeviceNode
abstract LoadPowerConnector : PowerConnector
    [mass = Data.MassPerLength.LoadPowerConnector*length]
    [cost = Data.CostPerLength.LoadPowerConnector*length]
abstract DevicePowerConnector : PowerConnector
    [mass = Data.MassPerLength.DevicePowerConnector*length]
    [cost = Data.CostPerLength.DevicePowerConnector*length]

abstract HardwareDataConnector : HardwareConnector
    endpoint -> DeviceNode 2..*
    deployedFrom -> FunctionConnector 1..*
        [this.deployedTo = parent]
    transferTimePerSize -> integer // [us/byte]
```

```
82
83  abstract DiscreteDataConnector : HardwareDataConnector
84      [mass = length*(#deployedFrom)*Data.MassPerLength.DiscreteDataConnector]
85      [transferTimePerSize = 0]
86      [cost = Data.CostPerLength.DiscreteDataConnector*length*(#deployedFrom)]
87
88  abstract AnalogDataConnector : HardwareDataConnector
89      [mass = length*(#deployedFrom)*Data.MassPerLength.AnalogDataConnector]
90      [transferTimePerSize = 0]
91      [cost = Data.CostPerLength.AnalogDataConnector*length*(#deployedFrom)]
92
93  abstract BusConnector : HardwareDataConnector
94      [endpoint.type = SmartDeviceNode]
95      xor type
96          LowSpeedCAN
97              [transferTimePerSize = Data.TimePerSize.LowSpeedCANBus]
98              [mass = Data.MassPerLength.LowSpeedCANBus*length]
99              [cost = Data.CostPerLength.LowSpeedCANBus*length]
100         HighSpeedCAN
101             [transferTimePerSize = Data.TimePerSize.HighSpeedCANBus]
102             [mass = Data.MassPerLength.HighSpeedCANBus*length]
103             [cost = Data.CostPerLength.HighSpeedCANBus*length]
104         LIN
105             [transferTimePerSize = Data.TimePerSize.LINBus]
106             [mass = Data.MassPerLength.LINBus*length]
107             [cost = Data.CostPerLength.LINBus*length]
108         FlexRay
109             [transferTimePerSize = Data.TimePerSize.FlexRayBus]
110             [mass = Data.MassPerLength.FlexRayBus*length]
111             [cost = Data.CostPerLength.FlexRayBus*length]
112
113 abstract LogicalBusBridge : HardwareDataConnector
114     [endpoint.type = SmartDeviceNode]
115     bus -> BusConnector 2
116     gatewayTransferTimePerSize -> integer // [us/byte]
117     [transferTimePerSize = gatewayTransferTimePerSize + sum(bus.transferTimePerSize)]
118     [length = 0]
119     [mass = 0]
120     [cost = 0]
121
122
123 // ---------------------- Quality Attribute Data -----------------------------//
124 Data
125     MassPerLength // [mg/cm]
126         LoadPowerConnector -> integer = 185
127         DevicePowerConnector -> integer = 104
128         DiscreteDataConnector -> integer = 110
129         AnalogDataConnector -> integer = 110
130         LowSpeedCANBus -> integer = 20
131         HighSpeedCANBus -> integer = 20
132         LINBus -> integer = 20
133         FlexRayBus -> integer = 40
134     CostPerLength // [dollar per thousand / cm]
135         LoadPowerConnector -> integer = 9
136         DevicePowerConnector -> integer = 9
137         DiscreteDataConnector -> integer = 13
138         AnalogDataConnector -> integer = 13
139         LowSpeedCANBus -> integer = 52
140         HighSpeedCANBus -> integer = 104
141         LINBus -> integer = 26
142         FlexRayBus -> integer = 208
143     TimePerSize // [us/byte]
144         LowSpeedCANBus -> integer = 64
145         HighSpeedCANBus -> integer = 32
146         LINBus -> integer = 400
147         FlexRayBus -> integer = 1
148     ReferenceSpeedFactor -> integer = 10
```

Appendix B
Generalized Power Window

Listing 30. Complete generalized E/E architecture for power window

```
1  //------------------- Power Window Abstract Clafer -------------------------//
2  // Power Window Abstract Clafer - This section contains all abstract clafers
3  // that detail a generic system/component that can be used in the concerete
4  // system model.
5
6  abstract WinSysFM : FeatureModel
7      basicUpDown : Feature
8      express : Feature ?
9          expressUp : Feature ?
10
11 abstract WinSysFA : FunctionalAnalysis
12     WinSwitch : FunctionalDevice
```

```
        [implementation.hardware]
        [baseLatency = 20]
    WinArbiter : AnalysisFunction
        [baseLatency = (if implementation.software then 1 else 5)]
    WinController : AnalysisFunction
        [implementation.software]
        [baseLatency = 2]
    WinMotor : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    CurrentSensor : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 5]

    localWinReq : FunctionConnector
        [sender = WinSwitch && receiver = WinArbiter]
        [messageSize = 1]
    winReq : FunctionConnector
        [sender = WinArbiter && receiver = WinController]
        [messageSize = 1]
    winCmd : FunctionConnector
        [sender = WinController && receiver = WinMotor]
        [messageSize = 2]
    current : FunctionConnector
        [sender = CurrentSensor && receiver = WinController]
        [messageSize = 1]

    PinchDetectionFA : FunctionalAnalysis ?
        PinchDetection : AnalysisFunction
            [implementation.software]
            [baseLatency = 2]
        PositionSensor : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]
        object : FunctionConnector
            [sender = PinchDetection && receiver = WinController]
            [messageSize = 2]
        position : FunctionConnector
            [sender = PositionSensor && receiver = PinchDetection]
            [messageSize = 1]

abstract WinSysDN : DeviceNodeClassification
    BCM -> DeviceNode ?
    EC -> DeviceNode
    Switch : SwitchNode
        [type in (SmartDeviceNode, EEDeviceNode)]
        [baseMass = 173]
        [cost = 110]
        [replaceCost = 110]
        [if (type in SmartDeviceNode) then (ppm = 50) else (ppm = 10)]
        [(type in SmartDeviceNode) => (speedFactor = 10)]
    Motor : DeviceNode
        [type in (SmartDeviceNode, EEDeviceNode)]
        [mass = 453]
        [if (type in SmartDeviceNode) then (cost = 107) else (cost = 122)]
        [if (type in SmartDeviceNode) then (ppm = 50) else (ppm = 20)]
        [if (type in SmartDeviceNode) then (replaceCost = 107) else (replaceCost = 122)]
        [(type in SmartDeviceNode) => (speedFactor = 10)]
    DoorInline : DeviceNode
        [type = PowerDeviceNode]
        [mass = 10] //TODO: Not a realistic number
        [cost = 4] //TODO: Not a realistic number
        [ppm = 1]
        [replaceCost = 2] //TODO: Not a realistic number
    DoorModule : DeviceNode ?
        [type = SmartDeviceNode]
        [mass = 368]
        [cost = 300]
        [ppm = 50]
        [replaceCost = 300]
        [speedFactor = 10]

abstract WinSysPT : PowerTopology
    dn -> WinSysDN

    inlineECDist -> integer
    inlineBCMDist -> integer


    MotorLoadPowerWire : LoadPowerConnector
        [sink = dn.Motor]
    SwitchLoadPowerWire : LoadPowerConnector ?
        [source = dn.DoorInline && sink = dn.Switch]
        [length = 45]
    DoorModuleLoadPowerWire : LoadPowerConnector ?
        [source = dn.DoorInline && sink = dn.DoorModule]
        [length = 35]
```

```
    DoorInlineLoadPowerWire : LoadPowerConnector
        [sink = dn.DoorInline]

    xor MotorLoadPowerConfig
        SwitchIsMotorDriver
            [MotorLoadPowerWire.source = dn.Switch]
            [MotorLoadPowerWire.length = 40]
            [DoorInlineLoadPowerWire.source = dn.EC.dref]
            [DoorInlineLoadPowerWire.length = inlineECDist]
            [SwitchLoadPowerWire && DoorInlineLoadPowerWire && no DoorModuleLoadPowerWire]
        DoorModuleIsMotorDriver
            [MotorLoadPowerWire.source = dn.DoorModule]
            [MotorLoadPowerWire.length = 30]
            [DoorInlineLoadPowerWire.source = dn.EC.dref]
            [DoorInlineLoadPowerWire.length = inlineECDist]
            [no SwitchLoadPowerWire && DoorInlineLoadPowerWire && DoorModuleLoadPowerWire]
        BCMIsMotorDriver
            [MotorLoadPowerWire.source = dn.DoorInline]
            [MotorLoadPowerWire.length = 45]
            [DoorInlineLoadPowerWire.source = dn.BCM.dref]
            [DoorInlineLoadPowerWire.length = inlineBCMDist]
            [no SwitchLoadPowerWire && DoorInlineLoadPowerWire && no DoorModuleLoadPowerWire]
        MotorIsMotorDriver
            [MotorLoadPowerWire.source = dn.DoorInline]
            [MotorLoadPowerWire.length = 45]
            [DoorInlineLoadPowerWire.source = dn.EC.dref]
            [DoorInlineLoadPowerWire.length = inlineECDist]
            [no SwitchLoadPowerWire && DoorInlineLoadPowerWire && no DoorModuleLoadPowerWire]

    switchInlineDP : DevicePowerConnector ?
        [source = dn.DoorInline && sink = dn.Switch]
        [length = 45]

    motorInlineDP : DevicePowerConnector ?
        [source = dn.DoorInline && sink = dn.Motor]
        [length = 45]

    doorModuleInlineDP : DevicePowerConnector ?
        [source = dn.DoorInline && sink = dn.DoorModule]
        [length = 35]

    [doorModuleInlineDP <=> dn.DoorModule]

    inlineECDP : DevicePowerConnector ?
        [source = dn.EC.dref && sink = dn.DoorInline]
        [length = WinSysPT.inlineECDist]

abstract WinSysCT : CommTopology
    dn -> WinSysDN
    inlineBCMDist -> integer


    logicalLowSpeedBus : BusConnector ?
        [type.LIN || type.LowSpeedCAN]
        [length = 70+inlineBCMDist]
        [endpoint in (dn.Motor, dn.Switch, dn.DoorModule, dn.BCM.dref)]

    logicalSwitchMotorDisc : DiscreteDataConnector ?
        [endpoint = (dn.Switch, dn.Motor)]
        [length = 40]
    logicalSwitchBCMDisc: DiscreteDataConnector ?
        [endpoint = (dn.Switch, dn.BCM.dref)]
        [length = 45+inlineBCMDist]
    logicalMotorBCMDisc : DiscreteDataConnector ?
        [endpoint = (dn.Motor, dn.BCM.dref)]
        [length = 45+inlineBCMDist]
    logicalSwitchDoorModuleDisc : DiscreteDataConnector ?
        [endpoint = (dn.Switch, dn.DoorModule)]
        [length = 25]
    logicalMotorDoorModuleDisc : DiscreteDataConnector ?
        [endpoint = (dn.Motor, dn.DoorModule)]
        [length = 30]

abstract WinSysHA : HardwareArchitecture
    dn -> WinSysDN
    pt -> WinSysPT
    ct -> WinSysCT


abstract WinSysDpl : Deployment
    fa -> WinSysFA
    ha -> WinSysHA

    // The most general deployment constraint that we have is that the
    // FunctionalAnalysisComponents must be deployed to its own HardwareTopology
    [fa.WinArbiter.deployedTo.dref in (ha.dn.BCM.dref, ha.dn.Switch, ha.dn.Motor, ha.dn.DoorModule)]
    [fa.WinController.deployedTo.dref in (ha.dn.BCM.dref, ha.dn.Switch, ha.dn.Motor, ha.dn.DoorModule)]
```

```
187     [fa.PinchDetectionFA => (fa.PinchDetectionFA.PinchDetection.deployedTo.dref in (ha.dn.BCM.dref, ha.dn.Switch, ha.dn.
            Motor, ha.dn.DoorModule))]
188
189     // More specific constraints on functional analysis component...
190     [fa.WinSwitch.deployedTo.dref = ha.dn.Switch]
191     [fa.WinMotor.deployedTo.dref = ha.dn.Motor]
192     [fa.CurrentSensor.deployedTo.dref = ha.dn.Motor]
193     [fa.PinchDetectionFA => (fa.PinchDetectionFA.PositionSensor.deployedTo.dref = ha.dn.Motor)]
194
195     // Constraints pertaining to the power topology selection based on analysis function deployment
196     [(fa.WinController.deployedTo.dref = ha.dn.Switch) <=> ha.pt.MotorLoadPowerConfig.SwitchIsMotorDriver]
197     [(fa.WinController.deployedTo.dref = ha.dn.Motor) <=> ha.pt.MotorLoadPowerConfig.MotorIsMotorDriver]
198     [(fa.WinController.deployedTo.dref = ha.dn.BCM.dref) <=> ha.pt.MotorLoadPowerConfig.BCMIsMotorDriver]
199     [(fa.WinController.deployedTo.dref = ha.dn.DoorModule) <=> ha.pt.MotorLoadPowerConfig.DoorModuleIsMotorDriver]
200
201     [ha.pt.switchInlineDP <=> (ha.dn.Switch.type in SmartDeviceNode)]
202     [ha.pt.motorInlineDP <=> (ha.dn.Motor.type in SmartDeviceNode)]
203     [ha.pt.inlineECDP <=> some(ha.pt.motorInlineDP, ha.pt.switchInlineDP, ha.pt.doorModuleInlineDP)]
204
205     // Constraints pertaining to the communication topology selected based on analysis function deployement
206     [(fa.localWinReq.deployedTo.dref in (ha.ct.logicalLowSpeedBus, ha.ct.logicalSwitchMotorDisc, ha.ct.
            logicalSwitchBCMDisc, ha.ct.logicalMotorBCMDisc, ha.ct.logicalSwitchDoorModuleDisc, ha.ct.
            logicalMotorDoorModuleDisc))]
207     [(fa.winReq.deployedTo.dref in (ha.ct.logicalLowSpeedBus, ha.ct.logicalSwitchMotorDisc, ha.ct.logicalSwitchBCMDisc, ha
            .ct.logicalMotorBCMDisc, ha.ct.logicalSwitchDoorModuleDisc, ha.ct.logicalMotorDoorModuleDisc))]
208     [(fa.winCmd.deployedTo.dref in (ha.ct.logicalLowSpeedBus, ha.ct.logicalSwitchMotorDisc, ha.ct.logicalSwitchBCMDisc, ha
            .ct.logicalMotorBCMDisc, ha.ct.logicalSwitchDoorModuleDisc, ha.ct.logicalMotorDoorModuleDisc))]
209     [(fa.current.deployedTo.dref in (ha.ct.logicalLowSpeedBus, ha.ct.logicalSwitchMotorDisc, ha.ct.logicalSwitchBCMDisc,
            ha.ct.logicalMotorBCMDisc, ha.ct.logicalSwitchDoorModuleDisc, ha.ct.logicalMotorDoorModuleDisc))]
210     [(fa.PinchDetectionFA.object.deployedTo.dref in (ha.ct.logicalLowSpeedBus, ha.ct.logicalSwitchMotorDisc, ha.ct.
            logicalSwitchBCMDisc, ha.ct.logicalMotorBCMDisc, ha.ct.logicalSwitchDoorModuleDisc, ha.ct.
            logicalMotorDoorModuleDisc))]
211     [(fa.PinchDetectionFA.position.deployedTo.dref in (ha.ct.logicalLowSpeedBus, ha.ct.logicalSwitchMotorDisc, ha.ct.
            logicalSwitchBCMDisc, ha.ct.logicalMotorBCMDisc, ha.ct.logicalSwitchDoorModuleDisc, ha.ct.
            logicalMotorDoorModuleDisc))]
212
213 abstract SwitchNode : DeviceNode
214     numSwitches -> integer
215     baseMass -> integer
216     [mass = baseMass*numSwitches]
```

Appendix C
Two Door Power Window

Listing 31. Complete E/E architecture for two door power window case study

```
1  //-------------------- Power Window System Model ----------------------------//
2  // Power Window System Model - This section is the concrete model of the power
3  // window system. This is the model that instances will be generated for. It
4  // will heavily use the previous two sections.
5
6  // Driver Window System
7  DriverWinSys : System
8      DWinSysFM : WinSysFM
9      DWinSysFA : WinSysFA
10         [DriverWinSys.DWinSysFM.express.expressUp <=> DWinSysFA.PinchDetectionFA]
11         // Timing Chains
12         SwitchToControlDeviceLatency -> integer = WinSwitch.latency + WinArbiter.latency
13         ControlToMotorDeviceLatency -> integer = WinController.latency + WinMotor.latency
14         SwitchToControlCommLatency -> integer = localWinReq.latency + winReq.latency
15         ControlToMotorCommLatency -> integer = winCmd.latency
16         SwitchToMotorEndToEndLatency -> integer = SwitchToControlDeviceLatency + ControlToMotorDeviceLatency + (
               SwitchToControlCommLatency+ControlToMotorCommLatency)/1000
17
18         PositionSensorToControlDeviceLatency -> integer = PositionSensor.latency + PinchDetection.latency
19         PositionSensorToControlCommLatency -> integer = position.latency + object.latency
20         PositionSensorToMotorEndToEndLatency -> integer = PositionSensorToControlDeviceLatency +
               ControlToMotorDeviceLatency + (PositionSensorToControlCommLatency+ControlToMotorCommLatency)/1000
21
22         SwitchToControlLatency -> integer = SwitchToControlDeviceLatency + SwitchToControlCommLatency/1000
23         PositionSensorToControlLatency -> integer = PositionSensorToControlDeviceLatency +
               PositionSensorToControlCommLatency/1000
24         ControlInputDifference -> integer
25         [ControlInputDifference = (max(SwitchToControlLatency.dref, PositionSensorToControlLatency.dref)
26             - min(SwitchToControlLatency.dref, PositionSensorToControlLatency.dref))]
27
28         // End-to-End Timing Constraint(s)
29         [(TimingRequirements.BasicEndToEndLatency) => (SwitchToMotorEndToEndLatency <= TimingRequirements.
               BasicEndToEndLatency)]
30         [(PinchDetectionFA && TimingRequirements.PinchDetectionEndToEndLatency) => (PositionSensorToMotorEndToEndLatency
               <= TimingRequirements.PinchDetectionEndToEndLatency)]
31         // Input Synchronization Contraint(s)
32         [(PinchDetectionFA && TimingRequirements.ControlInputSynchLatency) => ControlInputDifference <= TimingRequirements
               .ControlInputSynchLatency]
33
```

```
34          // Timing Margins
35          BasicEndToEndLatencyMargin -> integer ?
36          [if TimingRequirements.BasicEndToEndLatency then (BasicEndToEndLatencyMargin = (TimingRequirements.
                BasicEndToEndLatency - SwitchToMotorEndToEndLatency))
37              else (no BasicEndToEndLatencyMargin)]
38          PinchDetectionEndToEndLatencyMargin -> integer ?
39          [if TimingRequirements.PinchDetectionEndToEndLatency then (
40              PinchDetectionEndToEndLatencyMargin = (TimingRequirements.PinchDetectionEndToEndLatency -
                    PositionSensorToMotorEndToEndLatency))
41              else (no PinchDetectionEndToEndLatencyMargin)]
42      DWinSysHA : WinSysHA
43          DWinSysDN : WinSysDN
44              [this.BCM = Car.BCM]
45              [this.EC = Car.EC]
46              [this.Switch.numSwitches = 2]
47          DWinSysPT : WinSysPT
48              [dn = DWinSysDN]
49              [inlineECDist = 40]
50              [inlineBCMDist = 40]
51          DWinSysCT : WinSysCT
52              [dn = DWinSysDN]
53              [inlineBCMDist = 40]
54          [dn = DWinSysDN]
55          [pt = DWinSysPT]
56          [ct = DWinSysCT]
57      DWinSysDpl : WinSysDpl
58          [fa = DWinSysFA]
59          [ha = DWinSysHA]
60
61
62  // Passenger Window System
63  PassengerWinSys : System
64      PWinSysFM : WinSysFM
65          [express => DriverWinSys.DWinSysFM.express]
66          [express.expressUp => DriverWinSys.DWinSysFM.express.expressUp]
67      PWinSysFA : WinSysFA
68          [PassengerWinSys.PWinSysFM.express.expressUp <=> PWinSysFA.PinchDetectionFA]
69          DWinSwitch : FunctionalDevice
70              [implementation.hardware]
71              [baseLatency = 10]
72          dWinReq : FunctionConnector
73              [sender = DWinSwitch && receiver = WinArbiter]
74              [messageSize = 1]
75
76          // Timing Chains
77          SwitchToControlDeviceLatency -> integer = WinSwitch.latency + WinArbiter.latency
78          ControlToMotorDeviceLatency -> integer = WinController.latency + WinMotor.latency
79          SwitchToControlCommLatency -> integer = localWinReq.latency + winReq.latency
80          ControlToMotorCommLatency -> integer = winCmd.latency
81          SwitchToMotorEndToEndLatency -> integer = SwitchToControlDeviceLatency + ControlToMotorDeviceLatency + (
                SwitchToControlCommLatency+ControlToMotorCommLatency)/1000
82
83          PositionSensorToControlDeviceLatency -> integer = PositionSensor.latency + PinchDetection.latency
84          PositionSensorToControlCommLatency -> integer = position.latency + object.latency
85          PositionSensorToMotorEndToEndLatency -> integer = PositionSensorToControlDeviceLatency +
                ControlToMotorDeviceLatency + (PositionSensorToControlCommLatency+ControlToMotorCommLatency)/1000
86
87          SwitchToControlLatency -> integer = SwitchToControlDeviceLatency + SwitchToControlCommLatency/1000
88          PositionSensorToControlLatency -> integer = PositionSensorToControlDeviceLatency +
                PositionSensorToControlCommLatency/1000
89          ControlInputDifference -> integer
90          [ControlInputDifference = (max(SwitchToControlLatency.dref, PositionSensorToControlLatency.dref)
91              - min(SwitchToControlLatency.dref, PositionSensorToControlLatency.dref))]
92
93          // End-to-End Timing Constraint(s)
94          [(TimingRequirements.BasicEndToEndLatency) => (SwitchToMotorEndToEndLatency <= TimingRequirements.
                BasicEndToEndLatency)]
95          [(PinchDetectionFA && TimingRequirements.PinchDetectionEndToEndLatency) => (PositionSensorToMotorEndToEndLatency
                <= TimingRequirements.PinchDetectionEndToEndLatency)]
96          // Input Synchronization Contraint(s)
97          [(PinchDetectionFA && TimingRequirements.ControlInputSynchLatency) => ControlInputDifference <= TimingRequirements
                .ControlInputSynchLatency]
98
99          // Timing Margins
100         BasicEndToEndLatencyMargin -> integer ?
101         [if TimingRequirements.BasicEndToEndLatency then (BasicEndToEndLatencyMargin = (TimingRequirements.
                BasicEndToEndLatency - SwitchToMotorEndToEndLatency))
102             else (no BasicEndToEndLatencyMargin)]
103         PinchDetectionEndToEndLatencyMargin -> integer ?
104         [if TimingRequirements.PinchDetectionEndToEndLatency then (
105             PinchDetectionEndToEndLatencyMargin = (TimingRequirements.PinchDetectionEndToEndLatency -
                    PositionSensorToMotorEndToEndLatency))
106             else (no PinchDetectionEndToEndLatencyMargin)]
107
108     PWinSysHA : WinSysHA
109         PWinSysDN : WinSysDN
110             [this.BCM = Car.BCM]
```

```
            [this.EC = Car.EC]
            [this.Switch.numSwitches = 1]
            DSwitch -> SwitchNode
            [DSwitch = DriverWinSys.DWinSysHA.DWinSysDN.Switch]
        PWinSysPT : WinSysPT
            [dn = PWinSysDN]
            [inlineECDist = 130]
            [inlineBCMDist = 130]
        PWinSysCT : WinSysCT
            [dn = PWinSysDN]
            [inlineBCMDist = 130]
            logicalDoorBusJoin : LogicalBusBridge ?
                [bus = (PWinSysCT.logicalLowSpeedBus, DWinSysCT.logicalLowSpeedBus)]
                [gatewayTransferTimePerSize = 10] // This is the time to transfer a unit size over the gateway
                [endpoint in (PWinSysDN.Motor, PWinSysDN.Switch, PWinSysDN.DoorModule, PWinSysDN.BCM.dref, DWinSysDN.Motor
                    , DWinSysDN.Switch, DWinSysDN.DoorModule)]
            logicalDriveSwitchPassSwitch : DiscreteDataConnector ?
                [length = 260]
                [endpoint = (PWinSysHA.PWinSysDN.DSwitch.dref, PWinSysHA.PWinSysDN.Switch)]
            logicalDriveSwitchPassMotor : DiscreteDataConnector ?
                [length = 260]
                [endpoint = (PWinSysHA.PWinSysDN.DSwitch.dref, PWinSysHA.PWinSysDN.Motor)]
            logicalDriveSwitchPassDoorModule : DiscreteDataConnector ?
                [length = 250]
                [endpoint = (PWinSysHA.PWinSysDN.DSwitch.dref, PWinSysHA.PWinSysDN.DoorModule)]
            logicalDriveSwitchBCM : DiscreteDataConnector ?
                [length = 85]
                [endpoint = (PWinSysHA.PWinSysDN.DSwitch.dref, PWinSysHA.PWinSysDN.BCM.dref)]
        [dn = PWinSysDN]
        [pt = PWinSysPT]
        [ct = PWinSysCT]
    PWinSysDpl : WinSysDpl
        [fa = PWinSysFA]
        [ha = PWinSysHA]
        [PWinSysFA.DWinSwitch.deployedTo.dref = PWinSysHA.PWinSysDN.DSwitch.dref]
        [PWinSysFA.dWinReq.deployedTo.dref in (
            PWinSysHA.PWinSysCT.logicalDoorBusJoin,
            PWinSysHA.PWinSysCT.logicalDriveSwitchPassSwitch,
            PWinSysHA.PWinSysCT.logicalDriveSwitchPassMotor,
            PWinSysHA.PWinSysCT.logicalDriveSwitchPassDoorModule,
            PWinSysHA.PWinSysCT.logicalDriveSwitchBCM)]

//---------------------------- Car System Model ----------------------------//
Car
    BCM : DeviceNode ?
        [type = SmartDeviceNode]
        [mass = 408]
        [cost = 460]
        [ppm = 50]
        [replaceCost = 460]
        [speedFactor = 10]
    EC : DeviceNode
        [type = PowerDeviceNode]
        [mass = 0]
        [cost = 0]
        [ppm = 10]
        [replaceCost = 0]


totalCarMass -> integer = sum(DeviceNode.mass) + (sum(HardwareConnector.mass)/1000)
totalCarCost -> integer = sum(DeviceNode.cost) + (sum(HardwareConnector.cost)/1000)
totalCarWarrantyCost -> integer = sum(DeviceNode.warrantyCost)


// Timing Requirements
TimingRequirements
    BasicEndToEndLatency -> integer ?
    PinchDetectionEndToEndLatency -> integer ?
    ControlInputSynchLatency -> integer ?

// Optimization Goals:
// Comment out these goals if optimization should not be perfromed (no other modifications are necessary)
// << minimize totalCarMass >>
// << minimize totalCarCost >>
// << minimize totalCarWarrantyCost >>
```

Appendix D
Central Door Locks

Listing 32. Complete E/E architecture for central door locks case study

```
//------------------- Door Lock Abstract Clafer -------------------------//
// Door Lock Abstract Clafer - This section contains all abstract clafers
// that detail a generic system/component that can be used in the concerete
// system model.
```

```
abstract DoorLockFA : FunctionalAnalysis
    // -------  Core Components ---------------//
    // Cylinder Switches
    DriverDoorCylinderSwitch : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    PassDoorCylinderSwitch : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    driverCylReq : FunctionConnector
        [messageSize = 1]
        [sender = DriverDoorCylinderSwitch && receiver = DoorLockControl]
    passCylReq : FunctionConnector
        [messageSize = 1]
        [sender = PassDoorCylinderSwitch && receiver = DoorLockControl]
    // Door Contacts
    DriverDoorContact : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    PassDoorContact : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    RearRightPassDoorContact : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    RearLeftPassDoorContact : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    driverContactSignal : FunctionConnector
        [messageSize = 1]
        [sender = DriverDoorContact && receiver = DoorLockControl]
    passContactSignal : FunctionConnector
        [messageSize = 1]
        [sender = PassDoorContact && receiver = DoorLockControl]
    rearRightPassContactSignal : FunctionConnector
        [messageSize = 1]
        [sender = RearRightPassDoorContact && receiver = DoorLockControl]
    rearLeftPassContactSignal : FunctionConnector
        [messageSize = 1]
        [sender = RearLeftPassDoorContact && receiver = DoorLockControl]
    // Door Lock Sensors
    DriverDoorLockSensor : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    PassDoorLockSensor : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    RearRightPassDoorLockSensor : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    RearLeftPassDoorLockSensor : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    driverLockPosition : FunctionConnector
        [messageSize = 1]
        [sender = DriverDoorLockSensor && receiver = DoorLockControl]
    passLockPosition : FunctionConnector
        [messageSize = 1]
        [sender = PassDoorLockSensor && receiver = DoorLockControl]
    rearRightPassLockPosition : FunctionConnector
        [messageSize = 1]
        [sender = RearRightPassDoorLockSensor && receiver = DoorLockControl]
    rearLeftPassLockPosition : FunctionConnector
        [messageSize = 1]
        [sender = RearLeftPassDoorLockSensor && receiver = DoorLockControl]
    // Door Lock Control
    DoorLockControl : AnalysisFunction
        [implementation.software]
        [baseLatency = 4]
    driverLockCmd : FunctionConnector
        [messageSize = 1]
        [sender = DoorLockControl && receiver = DriverDoorLockMotor]
    passLockCmd : FunctionConnector
        [messageSize = 1]
        [sender = DoorLockControl && receiver = PassDoorLockMotor]
    rearRightLockCmd : FunctionConnector
        [messageSize = 1]
        [sender = DoorLockControl && receiver = RearRightPassDoorLockMotor]
    rearLeftLockCmd : FunctionConnector
        [messageSize = 1]
        [sender = DoorLockControl && receiver = RearLeftPassDoorLockMotor]
    // Door Lock Motor
    DriverDoorLockMotor : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    PassDoorLockMotor : FunctionalDevice
        [implementation.hardware]
```

```
        [baseLatency = 10]
    RearRightPassDoorLockMotor : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    RearLeftPassDoorLockMotor : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    // Gear Position Sensor
    GearPositionSensor : FunctionalDevice
        [implementation.hardware]
        [baseLatency = 10]
    gearPostion : FunctionConnector
        [messageSize = 1]
        [sender = GearPositionSensor && receiver = DoorLockControl]

    // ----- Optional Fragments/Components --------//
    // Speed Smart Lock FA Components
    SpeedSmartLockFA : FunctionalAnalysis ?
        SpeedSensor : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]
        speed : FunctionConnector
            [messageSize = 1]
            [sender = SpeedSensor && receiver = DoorLockControl]
    // Central or Distributed Lock Switch
    xor DoorLockButtonFA
        IndividualLockSwitchFA : FunctionalAnalysis
            DriverDoorLockButton : FunctionalDevice
                [implementation.hardware]
                [baseLatency = 10]
            PassDoorLockButton : FunctionalDevice
                [implementation.hardware]
                [baseLatency = 10]
            driverDoorLockReq : FunctionConnector
                [messageSize = 1]
                [sender = DriverDoorLockButton && receiver = DoorLockControl]
            passDoorLockReq : FunctionConnector
                [messageSize = 1]
                [sender = PassDoorLockButton && receiver = DoorLockControl]
        CentralLockSwitchFA : FunctionalAnalysis
            CentralLockButton : FunctionalDevice
                [implementation.hardware]
                [baseLatency = 10]
            centralDoorLockReq : FunctionConnector
                [messageSize = 1]
                [sender = CentralLockButton && receiver = DoorLockControl]

    RemoteKeyAccessFA : FunctionalAnalysis ?
        CentralRFAntenna : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]
        CentralRFReceiver : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]
        IDAuthentication : AnalysisFunction
            [implementation.software]
            [baseLatency = 4]

        centralAntennaSignal : FunctionConnector
            [messageSize = 1]
            [sender = CentralRFAntenna && receiver = CentralRFReceiver]
        centralReceiverMsg : FunctionConnector
            [messageSize = 1]
            [sender = CentralRFReceiver && receiver = IDAuthentication]
        authenticationMsg : FunctionConnector
            [messageSize = 1]
            [sender = IDAuthentication && receiver = DoorLockControl]

    PassiveKeyEntryFA : FunctionalAnalysis ?
        DriverOutsideLFAntenna : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]
        DriverLFTransmitter : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]
        PassOutsideLFAntenna : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]
        PassLFTransmitter : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]
        InsideFrontLFAntenna : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]
        InsideCenterLFAntenna : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]
```

```
        InsideRearLFAntenna : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]
        InsideLFTransmitter : FunctionalDevice
            [implementation.hardware]
            [baseLatency = 10]

        driverTransMsg : FunctionConnector
            [messageSize = 1]
            [sender = DriverLFTransmitter && receiver = DriverOutsideLFAntenna]
        passTransMsg : FunctionConnector
            [messageSize = 1]
            [sender = PassLFTransmitter && receiver = PassOutsideLFAntenna]
        insideFrontTransMsg : FunctionConnector
            [messageSize = 1]
            [sender = InsideLFTransmitter && receiver = InsideFrontLFAntenna]
        insideCenterTransMsg : FunctionConnector
            [messageSize = 1]
            [sender = InsideLFTransmitter && receiver = InsideCenterLFAntenna]
        insideRearTransMsg : FunctionConnector
            [messageSize = 1]
            [sender = InsideLFTransmitter && receiver = InsideRearLFAntenna]


        xor OutsideDoorHandleSensor
            ButtonSensor
                DriverDoorButtonSensor : FunctionalDevice
                    [implementation.hardware]
                    [baseLatency = 10]
                PassDoorButtonSensor : FunctionalDevice
                    [implementation.hardware]
                    [baseLatency = 10]
            CapacitiveSensor
                DriverDoorCapacitiveSensor : FunctionalDevice
                    [implementation.hardware]
                    [baseLatency = 10]
                PassDoorCapacitiveSensor : FunctionalDevice
                    [implementation.hardware]
                    [baseLatency = 10]

        PKEControl : AnalysisFunction
            [implementation.software]
            [baseLatency = 4]

        driverDoorHandleReq : FunctionConnector
            [messageSize = 1]
            [sender in (OutsideDoorHandleSensor.ButtonSensor.DriverDoorButtonSensor,
                OutsideDoorHandleSensor.CapacitiveSensor.DriverDoorCapacitiveSensor) && receiver = PKEControl]
        passDoorHandleReq : FunctionConnector
            [messageSize = 1]
            [sender in (OutsideDoorHandleSensor.ButtonSensor.PassDoorButtonSensor,
                OutsideDoorHandleSensor.CapacitiveSensor.PassDoorCapacitiveSensor) && receiver = PKEControl]
        driverPKEReq : FunctionConnector
            [messageSize = 1]
            [sender = PKEControl && receiver = DriverLFTransmitter]
        passPKEReq : FunctionConnector
            [messageSize = 1]
            [sender = PKEControl && receiver = PassLFTransmitter]
        insidePKEReq : FunctionConnector
            [messageSize = 1]
            [sender = PKEControl && receiver = InsideLFTransmitter]
        doorLockControlReq : FunctionConnector
            [messageSize = 1]
            [sender = DoorLockControl && receiver = PKEControl]

abstract DoorLockDN : DeviceNodeClassification
    //---------- Core Device Nodes ----------------//
    DriverDoorLockMotorAssembly : DeviceNode
        [type = EEDeviceNode]
        [cost = 144]
        [ppm = 20]
        [replaceCost = 144]
        [mass = 104]
    PassengerDoorLockMotorAssembly : DeviceNode
        [type = EEDeviceNode]
        [cost = 144]
        [ppm = 20]
        [replaceCost = 144]
        [mass = 104]
    RearRightPassengerDoorLockMotorAssembly : DeviceNode
        [type = EEDeviceNode]
        [cost = 144]
        [ppm = 20]
        [replaceCost = 144]
        [mass = 104]
    RearLeftPassengerDoorLockMotorAssembly : DeviceNode
        [type = EEDeviceNode]
```

```
        [cost = 144]
        [ppm = 20]
        [replaceCost = 144]
        [mass = 104]
    TCM -> DeviceNode
    BCM -> DeviceNode
    EC -> DeviceNode

    // ------- Optional Device Nodes ---------------//
    // Speed Smart Lock Nodes
    CombinationMeter -> DeviceNode ?

    // Central or Individual Lock Nodes
    xor DoorLockButtonDN
        IndividualLockSwitchDN : DeviceNodeClassification
            DriverLockPowerSwitch : DeviceNode
                [type = EEDeviceNode]
                [cost = 23]
                [replaceCost = 23]
                [ppm = 10]
                [mass = 28]
            PassLockPowerSwitch : DeviceNode
                [type = EEDeviceNode]
                [cost = 23]
                [replaceCost = 23]
                [ppm = 10]
                [mass = 28]
        CentralLockSwitchDN
            CenterLockPowerSwitch : DeviceNode
                [type = EEDeviceNode]
                [cost = 23]
                [replaceCost = 23]
                [ppm = 10]
                [mass = 28]

    RemoteKeyAccessDN : DeviceNodeClassification ?
        CentralRFAntennaModule : DeviceNode
            [type = SmartDeviceNode]
            [mass = 91]
            [cost = 57]
            [ppm = 50]
            [replaceCost = 57]
            [speedFactor = 10]

    PassiveKeyEntryDN : DeviceNodeClassification ?
        Transmitter : DeviceNode ?
            [type = EEDeviceNode]
            [mass = 397]
            [cost = 239]
            [ppm = 50]
            [replaceCost = 293]
        PassiveKeyModule : DeviceNode ?
            [type = SmartDeviceNode]
            [mass = 408]
            [cost = 191]
            [ppm = 50]
            [replaceCost = 191]
            [speedFactor = 50]
        xor OutsideDoorHandleSensor
            ButtonSensor
                DriverDoorButtonHandleModule : DeviceNode
                    [type = EEDeviceNode]
                    [mass = 408]
                    [cost = 41]
                    [ppm = 10]
                    [replaceCost = 41]
                PassDoorButtonHandleModule : DeviceNode
                    [type = EEDeviceNode]
                    [mass = 408]
                    [cost = 41]
                    [ppm = 10]
                    [replaceCost = 41]
            CapacitiveSensor
                DriverDoorCapacitiveHandleModule : DeviceNode
                    [type = EEDeviceNode]
                    [mass = 198]
                    [cost = 218]
                    [ppm = 10]
                    [replaceCost = 218]
                PassDoorCapacitiveHandleModule : DeviceNode
                    [type = EEDeviceNode]
                    [mass = 198]
                    [cost = 218]
                    [ppm = 10]
                    [replaceCost = 218]
        InsideFrontLFAntenna : DeviceNode
            [type = EEDeviceNode]
```

```
            [mass = 198]
            [cost = 57]
            [ppm = 10]
            [replaceCost = 57]
        InsideCenterLFAntenna : DeviceNode
            [type = EEDeviceNode]
            [mass = 198]
            [cost = 57]
            [ppm = 10]
            [replaceCost = 57]
        InsideRearLFAntenna : DeviceNode
            [type = EEDeviceNode]
            [mass = 198]
            [cost = 57]
            [ppm = 10]
            [replaceCost = 57]

abstract DoorLockPT : PowerTopology
    dn -> DoorLockDN

    // Motor Load Power
    driverMotorLP : LoadPowerConnector
        [length = 10]
        [source = dn.BCM.dref && sink = dn.DriverDoorLockMotorAssembly]
    passMotorLP : LoadPowerConnector
        [length = 15]
        [source = dn.BCM.dref && sink = dn.PassengerDoorLockMotorAssembly]
    rearRightPassMotorLP : LoadPowerConnector
        [length = 25]
        [source = dn.BCM.dref && sink = dn.RearRightPassengerDoorLockMotorAssembly]
    rearLeftPassMotorLP : LoadPowerConnector
        [length = 30]
        [source = dn.BCM.dref && sink = dn.RearLeftPassengerDoorLockMotorAssembly]

    // Remote Key Access Device Power
    centralRFModuleDP : DevicePowerConnector ?
        [length = 10]
        [source = dn.EC.dref && sink = dn.RemoteKeyAccessDN.CentralRFAntennaModule]

    // Passive Key Entry Device Power
    pkeModuleDP : DevicePowerConnector ?
        [length = 4]
        [source = dn.EC.dref && sink = dn.PassiveKeyEntryDN.PassiveKeyModule]
    transmitterDP : DevicePowerConnector ?
        [length = 5]
        [source = dn.EC.dref && sink = dn.PassiveKeyEntryDN.Transmitter]
    driverCapacitiveSensorDP : DevicePowerConnector ?
        [length = 11]
        [source = dn.EC.dref && sink = dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.
            DriverDoorCapacitiveHandleModule]
    passCapacitiveSensorDP : DevicePowerConnector ?
        [length = 16]
        [source = dn.EC.dref && sink = dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.
            PassDoorCapacitiveHandleModule]

abstract DoorLockCT : CommTopology
    dn -> DoorLockDN

    // Busses
    logicalLowSpeedBus : BusConnector ? // This is the logical bus connecting lower priority ECU's such as in the body
         domain
        [type.LIN || type.LowSpeedCAN]
        [length = 45]
        [endpoint in (dn.BCM.dref, dn.RemoteKeyAccessDN.CentralRFAntennaModule, dn.PassiveKeyEntryDN.PassiveKeyModule)]
    logicalHighSpeedBus : BusConnector // This is the logical bus connecting high priority ECU's such as vehicle control
        [type.HighSpeedCAN || type.FlexRay]
        [length = 30]
        [endpoint in (dn.BCM.dref, dn.TCM.dref, dn.CombinationMeter.dref)]

    // Logical Discrete Wires
    logicalBCMDriverMotorAssemblyDW : DiscreteDataConnector
        [length = 12]
        [endpoint = (dn.BCM.dref, dn.DriverDoorLockMotorAssembly)]
    logicalBCMPassMotorAssemblyDW : DiscreteDataConnector
        [length = 17]
        [endpoint = (dn.BCM.dref, dn.PassengerDoorLockMotorAssembly)]
    logicalBCMRearRightPassMotorAssemblyDW : DiscreteDataConnector
        [length = 27]
        [endpoint = (dn.BCM.dref, dn.RearRightPassengerDoorLockMotorAssembly)]
    logicalBCMRearLeftPassMotorAssemblyDW : DiscreteDataConnector
        [length = 32]
        [endpoint = (dn.BCM.dref, dn.RearLeftPassengerDoorLockMotorAssembly)]
    logicalBCMDriverLockPowerSwitchDW : DiscreteDataConnector ?
        [length = 14]
        [endpoint = (dn.BCM.dref, dn.DoorLockButtonDN.IndividualLockSwitchDN.DriverLockPowerSwitch)]
    logicalBCMPassLockPowerSwitchDW : DiscreteDataConnector ?
        [length = 19]
```

```
        [endpoint = (dn.BCM.dref, dn.DoorLockButtonDN.IndividualLockSwitchDN.PassLockPowerSwitch)]
    logicalBCMCenterLockPowerSwitchDW : DiscreteDataConnector ?
        [length = 3]
        [endpoint = (dn.BCM.dref, dn.DoorLockButtonDN.CentralLockSwitchDN.CenterLockPowerSwitch)]

    logicalBCMDriverCapacitiveSensorModule : AnalogDataConnector ?
        [length = 15]
        [endpoint = (dn.BCM.dref, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.
            DriverDoorCapacitiveHandleModule)]
    logicalBCMPassCapacitiveSensorModule : AnalogDataConnector ?
        [length = 20]
        [endpoint = (dn.BCM.dref, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.
            PassDoorCapacitiveHandleModule)]
    logicalBCMDriverButtonSensorModule : AnalogDataConnector ?
        [length = 15]
        [endpoint = (dn.BCM.dref, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.ButtonSensor.DriverDoorButtonHandleModule)]
    logicalBCMPassButtonSensorModule : AnalogDataConnector ?
        [length = 20]
        [endpoint = (dn.BCM.dref, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.ButtonSensor.PassDoorButtonHandleModule)]

    logicalPKEModuleDriverCapacitiveSensorModule : DiscreteDataConnector ?
        [length = 15]
        [endpoint = (dn.PassiveKeyEntryDN.PassiveKeyModule, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.
            DriverDoorCapacitiveHandleModule)]
    logicalPKEModulePassCapacitiveSensorModule : DiscreteDataConnector ?
        [length = 20]
        [endpoint = (dn.PassiveKeyEntryDN.PassiveKeyModule, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.
            PassDoorCapacitiveHandleModule)]
    logicalPKEModuleDriverButtonSensorModule : DiscreteDataConnector ?
        [length = 15]
        [endpoint = (dn.PassiveKeyEntryDN.PassiveKeyModule, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.ButtonSensor.
            DriverDoorButtonHandleModule)]
    logicalPKEModulePassButtonSensorModule : DiscreteDataConnector ?
        [length = 20]
        [endpoint = (dn.PassiveKeyEntryDN.PassiveKeyModule, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.ButtonSensor.
            PassDoorButtonHandleModule)]

    logicalTransmitterDriverCapacitiveSensorModule : AnalogDataConnector ?
        [length = 15]
        [endpoint = (dn.PassiveKeyEntryDN.Transmitter, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.
            DriverDoorCapacitiveHandleModule)]
    logicalTransmitterPassCapacitiveSensorModule : AnalogDataConnector ?
        [length = 20]
        [endpoint = (dn.PassiveKeyEntryDN.Transmitter, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.
            PassDoorCapacitiveHandleModule)]
    logicalTransmitterDriverButtonSensorModule : AnalogDataConnector ?
        [length = 15]
        [endpoint = (dn.PassiveKeyEntryDN.Transmitter, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.ButtonSensor.
            DriverDoorButtonHandleModule)]
    logicalTransmitterPassButtonSensorModule : AnalogDataConnector ?
        [length = 20]
        [endpoint = (dn.PassiveKeyEntryDN.Transmitter, dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.ButtonSensor.
            PassDoorButtonHandleModule)]

    logicalPKEModuleTransmitter : DiscreteDataConnector ?
        [length = 5]
        [endpoint = (dn.PassiveKeyEntryDN.PassiveKeyModule, dn.PassiveKeyEntryDN.Transmitter)]

    logicalBCMInsideFrontAntenna : AnalogDataConnector ?
        [length = 13]
        [endpoint = (dn.BCM.dref, dn.PassiveKeyEntryDN.InsideFrontLFAntenna)]
    logicalTransmitterInsideFrontAntenna : AnalogDataConnector ?
        [length = 1]
        [endpoint = (dn.PassiveKeyEntryDN.Transmitter, dn.PassiveKeyEntryDN.InsideFrontLFAntenna)]
    logicalBCMInsideCenterAntenna : AnalogDataConnector ?
        [length = 6]
        [endpoint = (dn.BCM.dref, dn.PassiveKeyEntryDN.InsideCenterLFAntenna)]
    logicalTransmitterInsideCenterAntenna : AnalogDataConnector ?
        [length = 4]
        [endpoint = (dn.PassiveKeyEntryDN.Transmitter, dn.PassiveKeyEntryDN.InsideCenterLFAntenna)]
    logicalBCMInsideRearAntenna : AnalogDataConnector ?
        [length = 14]
        [endpoint = (dn.BCM.dref, dn.PassiveKeyEntryDN.InsideRearLFAntenna)]
    logicalTransmitterInsideRearAntenna : AnalogDataConnector ?
        [length = 12]
        [endpoint = (dn.PassiveKeyEntryDN.Transmitter, dn.PassiveKeyEntryDN.InsideRearLFAntenna)]


abstract DoorLockHA : HardwareArchitecture
    dn -> DoorLockDN
    pt -> DoorLockPT
    ct -> DoorLockCT

abstract DoorLockDpl : Deployment
    fa -> DoorLockFA
    ha -> DoorLockHA
```

```
514
515     // Cylinder Switch Deployment
516     [fa.DriverDoorCylinderSwitch.deployedTo = ha.dn.DriverDoorLockMotorAssembly]
517     [fa.PassDoorCylinderSwitch.deployedTo = ha.dn.PassengerDoorLockMotorAssembly]
518
519     // Door Contacts Deployment
520     [fa.DriverDoorContact.deployedTo = ha.dn.DriverDoorLockMotorAssembly]
521     [fa.PassDoorContact.deployedTo = ha.dn.PassengerDoorLockMotorAssembly]
522     [fa.RearRightPassDoorContact.deployedTo = ha.dn.RearRightPassengerDoorLockMotorAssembly]
523     [fa.RearLeftPassDoorContact.deployedTo = ha.dn.RearLeftPassengerDoorLockMotorAssembly]
524
525     // Door Lock Sensors Deployment
526     [fa.DriverDoorLockSensor.deployedTo = ha.dn.DriverDoorLockMotorAssembly]
527     [fa.PassDoorLockSensor.deployedTo = ha.dn.PassengerDoorLockMotorAssembly]
528     [fa.RearRightPassDoorLockSensor.deployedTo = ha.dn.RearRightPassengerDoorLockMotorAssembly]
529     [fa.RearLeftPassDoorLockSensor.deployedTo = ha.dn.RearLeftPassengerDoorLockMotorAssembly]
530
531     // Door Lock Control Deployment
532     [fa.DoorLockControl.deployedTo = ha.dn.BCM.dref]
533
534
535     // Door Lock Motor Deployment
536     [fa.DriverDoorLockMotor.deployedTo = ha.dn.DriverDoorLockMotorAssembly]
537     [fa.PassDoorLockMotor.deployedTo = ha.dn.PassengerDoorLockMotorAssembly]
538     [fa.RearRightPassDoorLockMotor.deployedTo = ha.dn.RearRightPassengerDoorLockMotorAssembly]
539     [fa.RearLeftPassDoorLockMotor.deployedTo = ha.dn.RearLeftPassengerDoorLockMotorAssembly]
540
541     // Gear Position Sensor Deployment
542     [fa.GearPositionSensor.deployedTo = ha.dn.TCM.dref]
543
544     // Speed Sensor Deployment
545     [fa.SpeedSmartLockFA => (fa.SpeedSmartLockFA.SpeedSensor.deployedTo = ha.dn.CombinationMeter.dref)]
546
547     // Power Button Unlock Deployment
548     [fa.DoorLockButtonFA.IndividualLockSwitchFA => (fa.DoorLockButtonFA.IndividualLockSwitchFA.DriverDoorLockButton.
          deployedTo = ha.dn.DoorLockButtonDN.IndividualLockSwitchDN.DriverLockPowerSwitch)]
549     [fa.DoorLockButtonFA.IndividualLockSwitchFA => (fa.DoorLockButtonFA.IndividualLockSwitchFA.PassDoorLockButton.
          deployedTo = ha.dn.DoorLockButtonDN.IndividualLockSwitchDN.PassLockPowerSwitch)]
550     [fa.DoorLockButtonFA.CentralLockSwitchFA => (fa.DoorLockButtonFA.CentralLockSwitchFA.CentralLockButton.deployedTo = ha
          .dn.DoorLockButtonDN.CentralLockSwitchDN.CenterLockPowerSwitch)]
551
552     // Remote Key Access Deployment
553     [fa.RemoteKeyAccessFA => (fa.RemoteKeyAccessFA.CentralRFAntenna.deployedTo = ha.dn.RemoteKeyAccessDN.
          CentralRFAntennaModule)]
554     [fa.RemoteKeyAccessFA => (fa.RemoteKeyAccessFA.CentralRFReceiver.deployedTo = ha.dn.RemoteKeyAccessDN.
          CentralRFAntennaModule)]
555     [fa.RemoteKeyAccessFA => (fa.RemoteKeyAccessFA.IDAuthentication.deployedTo in (ha.dn.BCM.dref, ha.dn.RemoteKeyAccessDN
          .CentralRFAntennaModule, ha.dn.PassiveKeyEntryDN.PassiveKeyModule))]
556
557     // Passive Key Entry Deployment
558     PassiveKeyEntryDpl ?
559         xor OutsideDoorHandleSensor
560             ButtonSensor
561                 [ha.dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.ButtonSensor && fa.PassiveKeyEntryFA.
                        OutsideDoorHandleSensor.ButtonSensor]
562                 [fa.PassiveKeyEntryFA.OutsideDoorHandleSensor.ButtonSensor.DriverDoorButtonSensor.deployedTo = ha.dn.
                        PassiveKeyEntryDN.OutsideDoorHandleSensor.ButtonSensor.DriverDoorButtonHandleModule]
563                 [fa.PassiveKeyEntryFA.OutsideDoorHandleSensor.ButtonSensor.PassDoorButtonSensor.deployedTo = ha.dn.
                        PassiveKeyEntryDN.OutsideDoorHandleSensor.ButtonSensor.PassDoorButtonHandleModule]
564                 [fa.PassiveKeyEntryFA.DriverOutsideLFAntenna.deployedTo = ha.dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.
                        ButtonSensor.DriverDoorButtonHandleModule]
565                 [fa.PassiveKeyEntryFA.PassOutsideLFAntenna.deployedTo = ha.dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.
                        ButtonSensor.PassDoorButtonHandleModule]
566             CapacitiveSensor
567                 [ha.dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor && fa.PassiveKeyEntryFA.
                        OutsideDoorHandleSensor.CapacitiveSensor]
568                 [fa.PassiveKeyEntryFA.OutsideDoorHandleSensor.CapacitiveSensor.DriverDoorCapacitiveSensor.deployedTo = ha.
                        dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.DriverDoorCapacitiveHandleModule]
569                 [fa.PassiveKeyEntryFA.OutsideDoorHandleSensor.CapacitiveSensor.PassDoorCapacitiveSensor.deployedTo = ha.dn
                        .PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.PassDoorCapacitiveHandleModule]
570                 [fa.PassiveKeyEntryFA.DriverOutsideLFAntenna.deployedTo = ha.dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.
                        CapacitiveSensor.DriverDoorCapacitiveHandleModule]
571                 [fa.PassiveKeyEntryFA.PassOutsideLFAntenna.deployedTo = ha.dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.
                        CapacitiveSensor.PassDoorCapacitiveHandleModule]
572
573
574         [fa.PassiveKeyEntryFA.DriverLFTransmitter.deployedTo in (ha.dn.PassiveKeyEntryDN.Transmitter, ha.dn.BCM.dref)]
575         [fa.PassiveKeyEntryFA.PassLFTransmitter.deployedTo in (ha.dn.PassiveKeyEntryDN.Transmitter, ha.dn.BCM.dref)]
576
577         [fa.PassiveKeyEntryFA.InsideFrontLFAntenna.deployedTo = ha.dn.PassiveKeyEntryDN.InsideFrontLFAntenna]
578         [fa.PassiveKeyEntryFA.InsideCenterLFAntenna.deployedTo = ha.dn.PassiveKeyEntryDN.InsideCenterLFAntenna]
579         [fa.PassiveKeyEntryFA.InsideRearLFAntenna.deployedTo = ha.dn.PassiveKeyEntryDN.InsideRearLFAntenna]
580         [fa.PassiveKeyEntryFA.InsideLFTransmitter.deployedTo in (ha.dn.PassiveKeyEntryDN.Transmitter, ha.dn.BCM.dref)]
581
582         [fa.PassiveKeyEntryFA.PKEControl.deployedTo in (ha.dn.BCM.dref, ha.dn.PassiveKeyEntryDN.PassiveKeyModule)]
583
584     // Power Topology Deployment
```

```
    [ha.pt.pkeModuleDP <=> ha.dn.PassiveKeyEntryDN.PassiveKeyModule]
    [ha.pt.driverCapacitiveSensorDP <=> ha.dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.
        DriverDoorCapacitiveHandleModule]
    [ha.pt.passCapacitiveSensorDP <=> ha.dn.PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor.
        PassDoorCapacitiveHandleModule]
    [ha.pt.centralRFModuleDP <=> ha.dn.RemoteKeyAccessDN]
    [ha.pt.transmitterDP <=> ha.dn.PassiveKeyEntryDN.Transmitter]

    // Communication Deployment
    [fa.driverCylReq.deployedTo = ha.ct.logicalBCMDriverMotorAssemblyDW]
    [fa.passCylReq.deployedTo = ha.ct.logicalBCMPassMotorAssemblyDW]

    [fa.driverContactSignal.deployedTo = ha.ct.logicalBCMDriverMotorAssemblyDW]
    [fa.passContactSignal.deployedTo  = ha.ct.logicalBCMPassMotorAssemblyDW]
    [fa.rearRightPassContactSignal.deployedTo = ha.ct.logicalBCMRearRightPassMotorAssemblyDW]
    [fa.rearLeftPassContactSignal.deployedTo = ha.ct.logicalBCMRearLeftPassMotorAssemblyDW]

    [fa.driverLockPosition.deployedTo = ha.ct.logicalBCMDriverMotorAssemblyDW]
    [fa.passLockPosition.deployedTo = ha.ct.logicalBCMPassMotorAssemblyDW]
    [fa.rearRightPassLockPosition.deployedTo = ha.ct.logicalBCMRearRightPassMotorAssemblyDW]
    [fa.rearLeftPassLockPosition.deployedTo = ha.ct.logicalBCMRearLeftPassMotorAssemblyDW]

    [fa.driverLockCmd.deployedTo = ha.ct.logicalBCMDriverMotorAssemblyDW]
    [fa.passLockCmd.deployedTo = ha.ct.logicalBCMPassMotorAssemblyDW]
    [fa.rearRightLockCmd.deployedTo = ha.ct.logicalBCMRearRightPassMotorAssemblyDW]
    [fa.rearLeftLockCmd.deployedTo = ha.ct.logicalBCMRearLeftPassMotorAssemblyDW]

    [fa.gearPostion.deployedTo = ha.ct.logicalHighSpeedBus]

    [fa.SpeedSmartLockFA => (fa.SpeedSmartLockFA.speed.deployedTo in (ha.ct.logicalHighSpeedBus))]

    [fa.DoorLockButtonFA.IndividualLockSwitchFA => (fa.DoorLockButtonFA.IndividualLockSwitchFA.driverDoorLockReq.
        deployedTo = ha.ct.logicalBCMDriverLockPowerSwitchDW)]
    [fa.DoorLockButtonFA.IndividualLockSwitchFA => (fa.DoorLockButtonFA.IndividualLockSwitchFA.passDoorLockReq.deployedTo
        = ha.ct.logicalBCMPassLockPowerSwitchDW)]
    [fa.DoorLockButtonFA.CentralLockSwitchFA => (fa.DoorLockButtonFA.CentralLockSwitchFA.centralDoorLockReq.deployedTo =
        ha.ct.logicalBCMCenterLockPowerSwitchDW)]

    [fa.RemoteKeyAccessFA => (no fa.RemoteKeyAccessFA.centralAntennaSignal.deployedTo)]
    [fa.RemoteKeyAccessFA => (fa.RemoteKeyAccessFA.centralReceiverMsg.deployedTo in (ha.ct.logicalLowSpeedBus))]
    [fa.RemoteKeyAccessFA => (fa.RemoteKeyAccessFA.authenticationMsg.deployedTo in (ha.ct.logicalLowSpeedBus))]

    [fa.PassiveKeyEntryFA => fa.PassiveKeyEntryFA.driverTransMsg.deployedTo in (ha.ct.
        logicalTransmitterDriverCapacitiveSensorModule, ha.ct.logicalTransmitterDriverButtonSensorModule, ha.ct.
        logicalBCMDriverCapacitiveSensorModule, ha.ct.logicalBCMDriverButtonSensorModule)]
    [fa.PassiveKeyEntryFA => fa.PassiveKeyEntryFA.driverPKEReq.deployedTo in (ha.ct.logicalPKEModuleTransmitter, ha.ct.
        logicalLowSpeedBus)]
    [fa.PassiveKeyEntryFA => fa.PassiveKeyEntryFA.passTransMsg.deployedTo in (ha.ct.
        logicalTransmitterPassCapacitiveSensorModule, ha.ct.logicalTransmitterPassButtonSensorModule, ha.ct.
        logicalBCMPassCapacitiveSensorModule, ha.ct.logicalBCMPassButtonSensorModule)]
    [fa.PassiveKeyEntryFA => fa.PassiveKeyEntryFA.passPKEReq.deployedTo in (ha.ct.logicalPKEModuleTransmitter, ha.ct.
        logicalLowSpeedBus)]
    [fa.PassiveKeyEntryFA => fa.PassiveKeyEntryFA.insideFrontTransMsg.deployedTo in (ha.ct.
        logicalTransmitterInsideFrontAntenna, ha.ct.logicalBCMInsideFrontAntenna)]
    [fa.PassiveKeyEntryFA => fa.PassiveKeyEntryFA.insideCenterTransMsg.deployedTo in (ha.ct.
        logicalTransmitterInsideCenterAntenna, ha.ct.logicalBCMInsideCenterAntenna)]
    [fa.PassiveKeyEntryFA => fa.PassiveKeyEntryFA.insideRearTransMsg.deployedTo in (ha.ct.
        logicalTransmitterInsideRearAntenna, ha.ct.logicalBCMInsideRearAntenna)]
    [fa.PassiveKeyEntryFA => fa.PassiveKeyEntryFA.insidePKEReq.deployedTo in (ha.ct.logicalPKEModuleTransmitter, ha.ct.
        logicalLowSpeedBus)]
    [fa.PassiveKeyEntryFA => fa.PassiveKeyEntryFA.driverDoorHandleReq.deployedTo in (ha.ct.
        logicalPKEModuleDriverButtonSensorModule, ha.ct.logicalPKEModuleDriverCapacitiveSensorModule, ha.ct.
        logicalBCMDriverButtonSensorModule, ha.ct.logicalBCMDriverCapacitiveSensorModule)]
    [fa.PassiveKeyEntryFA => fa.PassiveKeyEntryFA.passDoorHandleReq.deployedTo in (ha.ct.
        logicalPKEModulePassButtonSensorModule, ha.ct.logicalPKEModulePassCapacitiveSensorModule, ha.ct.
        logicalBCMPassButtonSensorModule, ha.ct.logicalBCMPassCapacitiveSensorModule)]
    [fa.PassiveKeyEntryFA => fa.PassiveKeyEntryFA.doorLockControlReq.deployedTo in (ha.ct.logicalLowSpeedBus)]


//-------------------- Door Lock System Model ----------------------------//
DoorLockSys : System
    DLockFM : FeatureModel
        Basic : Feature
            IndividualLockSwitch : Feature ? // This feature is to determine if the driver and passenger should have
                individual door lock switches or use a central lock switch.
            SpeedSmartLock : Feature ? // This feature is if the door should lock when the car is above a certain speed.
        RKA : Feature ? // Remote Key Access
        PKE : Feature ? // Passive Key Entry
            xor OutsideDoorHandleSensor
                ButtonSensor : Feature
                CapacitiveSensor : Feature
        [PKE => RKA]
    DLockFA : DoorLockFA
        [DoorLockButtonFA.IndividualLockSwitchFA <=> DLockFM.Basic.IndividualLockSwitch]
        [SpeedSmartLockFA <=> DLockFM.Basic.SpeedSmartLock]
        [RemoteKeyAccessFA <=> DLockFM.RKA]
        [PassiveKeyEntryFA <=> DLockFM.PKE]
```

```
652         [PassiveKeyEntryFA.OutsideDoorHandleSensor.ButtonSensor <=> DLockFM.PKE.OutsideDoorHandleSensor.ButtonSensor]
653         [PassiveKeyEntryFA.OutsideDoorHandleSensor.CapacitiveSensor <=> DLockFM.PKE.OutsideDoorHandleSensor.
                CapacitiveSensor]
654
655         // Timing Chains
656         DriverSwitchToControl -> integer
657             [if (DLockFM.Basic.IndividualLockSwitch) then (
658                 this = DoorLockButtonFA.IndividualLockSwitchFA.DriverDoorLockButton.latency +
659                 DoorLockButtonFA.IndividualLockSwitchFA.driverDoorLockReq.latency/1000
660             ) else (
661                 this = DoorLockButtonFA.CentralLockSwitchFA.CentralLockButton.latency +
662                 DoorLockButtonFA.CentralLockSwitchFA.centralDoorLockReq.latency/1000
663             )]
664         DriverContactToControl -> integer = DriverDoorContact.latency + driverContactSignal.latency/1000
665         DriverLockSensorToControl -> integer = DriverDoorLockSensor.latency + driverLockPosition.latency/1000
666
667         DriverSwitchToMotor -> integer
668             [if (DLockFM.Basic.IndividualLockSwitch) then (
669                 this = DoorLockButtonFA.IndividualLockSwitchFA.DriverDoorLockButton.latency +
670                 DoorLockControl.latency +
671                 DriverDoorLockMotor.latency +
672                 ((driverLockCmd.latency + DoorLockButtonFA.IndividualLockSwitchFA.driverDoorLockReq.latency)/1000)
673             ) else (
674                 this = DoorLockButtonFA.CentralLockSwitchFA.CentralLockButton.latency +
675                 DoorLockControl.latency +
676                 DriverDoorLockMotor.latency +
677                 ((DoorLockButtonFA.CentralLockSwitchFA.centralDoorLockReq.latency + driverLockCmd.latency)/1000)
678             )]
679
680         ControlInputDifference -> integer
681         [ControlInputDifference = (max(DriverSwitchToControl.dref, DriverContactToControl.dref, DriverLockSensorToControl.
                dref)
682             - min(DriverSwitchToControl.dref, DriverContactToControl.dref, DriverLockSensorToControl.dref))]
683
684         PassiveKeyCapacitiveSensorToMotor -> integer ?
685         [if (DLockFM.PKE.OutsideDoorHandleSensor.CapacitiveSensor) then (
686             PassiveKeyCapacitiveSensorToMotor = PassiveKeyEntryFA.OutsideDoorHandleSensor.CapacitiveSensor.
                    DriverDoorCapacitiveSensor.latency +
687             PassiveKeyEntryFA.PKEControl.latency +
688             PassiveKeyEntryFA.DriverLFTransmitter.latency +
689             PassiveKeyEntryFA.DriverOutsideLFAntenna.latency + 50 +
690             RemoteKeyAccessFA.CentralRFAntenna.latency +
691             RemoteKeyAccessFA.CentralRFReceiver.latency +
692             RemoteKeyAccessFA.IDAuthentication.latency +
693             DoorLockControl.latency +
694             DriverDoorLockMotor.latency +
695             ((PassiveKeyEntryFA.driverDoorHandleReq.latency + PassiveKeyEntryFA.driverPKEReq.latency + PassiveKeyEntryFA.
                    driverTransMsg.latency +
696                 RemoteKeyAccessFA.centralAntennaSignal.latency + RemoteKeyAccessFA.centralReceiverMsg.latency +
697                 RemoteKeyAccessFA.authenticationMsg.latency + driverLockCmd.latency)/1000))
698         else (no PassiveKeyCapacitiveSensorToMotor)]
699
700         // Timing Constraints
701         // Driver lock switch to driver motor timing constraint
702         [(DLockFM.Basic.IndividualLockSwitch && DoorLockRequirements.TimingRequirements.BasicIndividualSwitchLatency) => (
703             DriverSwitchToMotor <= DoorLockRequirements.TimingRequirements.BasicIndividualSwitchLatency
704         )]
705         // Central lock switch to driver motor timing constraint
706         [(no DLockFM.Basic.IndividualLockSwitch && DoorLockRequirements.TimingRequirements.BasicCentralSwitchLatency)=> (
707             DriverSwitchToMotor <= DoorLockRequirements.TimingRequirements.BasicCentralSwitchLatency
708         )]
709         // Switch Unlock Input Synchronization Timing Constraint
710         [DoorLockRequirements.TimingRequirements.SwitchUnlockInputSynchLatency => (
711              ControlInputDifference <= DoorLockRequirements.TimingRequirements.SwitchUnlockInputSynchLatency
712         )]
713         // Passive Key Capacitive Sensor to Motor Timing Constraint
714         [(DoorLockRequirements.TimingRequirements.PKELatency && DLockFM.PKE.OutsideDoorHandleSensor.CapacitiveSensor) => (
715             PassiveKeyCapacitiveSensorToMotor <= DoorLockRequirements.TimingRequirements.PKELatency
716         )]
717
718         // Timing Margins
719         BasicIndividualSwitchLatencyMargin -> integer ?
720         [if DoorLockRequirements.TimingRequirements.BasicIndividualSwitchLatency then (BasicIndividualSwitchLatencyMargin
                = (DoorLockRequirements.TimingRequirements.BasicIndividualSwitchLatency - DriverSwitchToMotor))
721             else (no BasicIndividualSwitchLatencyMargin)]
722         BasicCentralSwitchLatencyMargin -> integer ?
723         [if DoorLockRequirements.TimingRequirements.SwitchUnlockInputSynchLatency then (BasicCentralSwitchLatencyMargin =
                (DoorLockRequirements.TimingRequirements.SwitchUnlockInputSynchLatency - DriverSwitchToMotor))
724             else (no BasicCentralSwitchLatencyMargin)]
725         SwitchUnlockInputSynchLatencyMargin -> integer ?
726         [if DoorLockRequirements.TimingRequirements.SwitchUnlockInputSynchLatency then (
727             SwitchUnlockInputSynchLatencyMargin = (DoorLockRequirements.TimingRequirements.SwitchUnlockInputSynchLatency -
                    ControlInputDifference))
728             else (no SwitchUnlockInputSynchLatencyMargin)]
729         PKELatencyMargin -> integer ?
730         [if DoorLockRequirements.TimingRequirements.PKELatency then (PKELatencyMargin = (DoorLockRequirements.
                TimingRequirements.PKELatency - PassiveKeyCapacitiveSensorToMotor))
```

```
731             else (no PKELatencyMargin)]
732
733     DLockHA : DoorLockHA
734         DLockDN : DoorLockDN
735             [BCM = Car.BCM]
736             [TCM = Car.TCM]
737             [EC = Car.EC]
738             [CombinationMeter => CombinationMeter = Car.CombinationMeter]
739             [DoorLockButtonDN.IndividualLockSwitchDN <=> DLockFM.Basic.IndividualLockSwitch]
740             [CombinationMeter <=> DLockFM.Basic.SpeedSmartLock]
741             [RemoteKeyAccessDN <=> DLockFM.RKA]
742             [PassiveKeyEntryDN <=> DLockFM.PKE]
743             [PassiveKeyEntryDN.OutsideDoorHandleSensor.ButtonSensor <=> DLockFM.PKE.OutsideDoorHandleSensor.ButtonSensor]
744             [PassiveKeyEntryDN.OutsideDoorHandleSensor.CapacitiveSensor <=> DLockFM.PKE.OutsideDoorHandleSensor.
                    CapacitiveSensor]
745         DLockPT : DoorLockPT
746             [dn = DLockDN]
747         DLockCT : DoorLockCT
748             [dn = DLockDN]
749         [dn = DLockDN]
750         [pt = DLockPT]
751         [ct = DLockCT]
752     DLockDpl : DoorLockDpl
753         [fa = DLockFA]
754         [ha = DLockHA]
755         [DLockFM.PKE <=> PassiveKeyEntryDpl]
756
757 DoorLockRequirements
758     TimingRequirements
759         BasicIndividualSwitchLatency -> integer ?
760         BasicCentralSwitchLatency -> integer ?
761         SwitchUnlockInputSynchLatency -> integer ?
762         PKELatency -> integer ?
763
764 Car
765     BCM : DeviceNode
766         [type = SmartDeviceNode]
767         [mass = 408]
768         [cost = 261]
769         [ppm = 50]
770         [replaceCost = 261]
771         [speedFactor = 10]
772     TCM : DeviceNode
773         [type = SmartDeviceNode]
774         [mass = 204]
775         [cost = 117]
776         [ppm = 50]
777         [replaceCost = 117]
778         [speedFactor = 10]
779     CombinationMeter : DeviceNode ?
780         [type = SmartDeviceNode]
781         [mass = 198]
782         [cost = 649]
783         [ppm = 50]
784         [replaceCost = 649]
785         [speedFactor = 10]
786     EC : DeviceNode
787         [type = PowerDeviceNode]
788         [mass = 0]
789         [cost = 0]
790         [ppm = 10]
791         [replaceCost = 0]
792
793 totalCarMass -> integer = sum(DeviceNode.mass) + sum(HardwareConnector.mass)/1000
794 totalCarCost -> integer = sum(DeviceNode.cost) + sum(HardwareDataConnector.cost)/1000
795 totalCarWarrantyCost -> integer = sum(DeviceNode.warrantyCost)/1000
796
797 // Optimization Goals:
798 // Comment out these goals if optimization should not be perfromed (no other modifications are necessary)
799 // << minimize totalCarMass >>
800 // << minimize totalCarCost >>
801 // << minimize totalCarWarrantyCost >>
```