CHALMERS

Variability Management and Deployment of Embedded Software

Master's Thesis in Computer Systems and Networks

TOBIAS BOSTRÖM

Department of Signals & Systems
Chalmers University of Technology
Gothenburg, Sweden 2015

Master's Thesis 2015:08

Abstract

For commercial vehicles, like trucks, the customer can have the vehicle customized for the individual needs. However, the possibility to customize the vehicles put demands on the manufacturing system to be able to adapt to handle all possible variants of the vehicle. Product platforms are used to lower the development costs, but due to the high customization needed the manufacturing process is still very complex. Manufacturing Executions Systems (MES) are used to control the manufacturing, and because of the high complexity it can be difficult to understand how these are used. Optimization on how resources, like machines and robots, are used is an important issue in the automotive industry and with high variability of the product computational tools for analyzing how the variability of the product affect the manufacturing system. In 2011 a demonstrator platform was created to educate Volvo employees on how the assembly process works. This platform utilizes all the MES tools used in the factories but with small scale model trucks to keep the size of the assembly line down. With the high variability of the trucks it is important to have a way to model the variability in such a way that all constructed trucks are valid and work correctly. In this thesis an approach to model variability of both the hardware and software in vehicles and the demonstrator platform is upgraded with new software and hardware functionalities to better facilitate the new variability models. An analysis of the strength and weakness with the new approach is provided.

Acknowledgements

This thesis work was carried out between January 2014 and June 2014 in collaboration between Chalmers University of Technology and Volvo Group Trucks Operations. I would like to thank Pierre Johansson, my supervisor at Volvo, as well as all the people at Volvo who helped me during the thesis.

A big thanks also goes to my thesis partner Liang Gong, which whom I would never had been able finish the project without.

Last, but not least, I would like to thank my supervisor at Chalmers Knut Åkesson, for his guidance throughput the project as well as the help with the thesis during the end.

Tobias Boström, Gothenburg 18/11/14

Contents

Acronyms

AAS	Assembly Assurance System
AGV	Automated Guided Vehicle
CAD	Computer-Aided Design
CTH	Chalmers University of Technology
CVL	Common Variability Language
DOID	Diagnostic Object Identifier
DSL	Domain Specific Language
ECU	Electronic Control Unit
GTO	Volvo Group Trucks Operations
IDW	Intentional Domain Workbench
KOLA	Konstruktion lastvagnar, PDM at GTO
LED	Light-Emitting Diode
MES	Manufacturing Executions Systems
MONT	MONTering, assembly in Swedish
UML	Unified Modeling Language
UU	Uppsala University
VR	Virtual Reality
PCB	Printed Circuit Board
PDM	Product Data Management
PROSIT	Production System Integrated Tool
SEWS2	System Engineering Web Server for TEA2
VMCU	Vehicle Master Control Unit

1 Introduction

Product families [1][2] are used to simplify the process of choosing the right product for the customer as well as keeping the cost of manufacturing down. When purchasing a truck this means choosing a product family as well as doing some customizations, such as choosing a suitable engine size. The underlying reason that the costs are kept down when using product families is the use of product platforms. By having a shared product platform between models in the same product family or even separate product families the development cost as well as manufacturing costs can be lowered significantly. Compared to car manufacturing the amount of customizations needed for trucks is larger, which means the variability is also higher. This increase in variability also leads to an increase in complexity of the production line. To be able handle this high variability a way to model the variability is needed. In this thesis the current work practises for this at Volvo Group will be investigated. In addition to this, new methods. By having good variability models, the added complexity to the production line can be minimized.

Because of of the high complexity of the production line, the cost of introducing new functionality is high. By instead testing new functionality on a demonstrator for the production system the costs can be lowered. The demonstrator is a small scale test environment that work in a similar way to the real production line but with LEGO® models instead of full scale trucks. It is also used as tool to educate Volvo Group employees on the production process as well as a way to test new ways to model the variability. As a part of this thesis work, the demonstrator is upgraded to handle software variants. This will make the demonstrator more closely resemble the real production line, which will improve it as a educational platform, as well as make it a better platform for testing new ways of modeling variability. This thesis project was conducted at Volvo Group Trucks at Gothenburg, Sweden. It is a part of the Know4Car project [3], a European research project founded by the seventh framework program. The Know4Car project aims to make knowledge management and collaboration more effective throughout the product life cycle, supporting the capture and the systematic organization of knowledge.

1.1 Background

Many Volvo Group employees have never been to the factory floor. Having a deeper knowledge of the assembly process would help people to get a better overview of the full production process. A demonstrator platform was implemented at Volvo in 2011 [4] for this purpose. It utilize LEGO® bricks and MINDSTORMS® components and together with the Volvo production systems it demonstrates the truck assembly features. As shown in figure 1.1, the platform consisted of five consecutive stations in the spring of of 2014, each corresponding to one step in the assembly process. At each station some form of assembly or testing is carried out. The process at each station depends on the truck that is currently assembled as each truck is built to order. By using the demonstrator it is possible to learn how the assembly process works without having to do the education on the real assembly line in the factory, which would slow down the manufacturing and thus cost a lot of money.

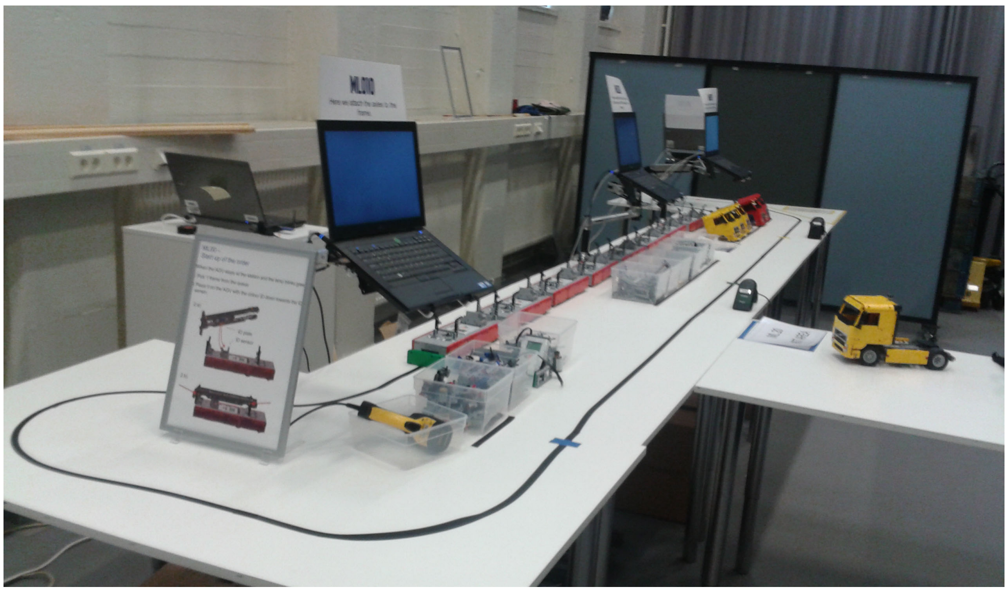


Figure 1.1: Picture showing the assembly line of the original demonstrator setup.

A problem in truck manufacturing is how to model the variability of the trucks as good possible. With the use of the demonstrator it would be possible test out the models without having to base the models on the real production systems. For this to be done as good as possible some additions are needed for the demonstrator. The current platform illustrates the features and key concepts of physical components assembly in truck production. However, this is not a complete view of a modern truck production process and it needs to be further improved. This includes electronic components in the trucks as well as software download features for the trucks. These upgrades to the demonstrator also serve the purpose of improving the experience of assembly training.

1.2 Purpose

The purpose of of this project is to gain knowledge on how variability currently is handled in the industry as well as finding a good way to model the variability of both hardware and software. In addition to this the purpose is to have a demonstrator platform that more closely resembles the real factory assembly line. At the end of this thesis project, the demonstrator platform will be able to illustrate not only physical components assembly, but also the software assembly/download features. Thus, the platform will provide a more complete view of the truck production process.

1.3 Objective

To achieve the purpose of this project, the thesis work will be carried out with the guidance of Volvo Group Trucks Operations (GTO) and Chalmers University of Technology (CTH). The main objective is find the best way to model the variability of Volvo trucks. This will be carried out with the help of the demonstrator. More specifically the following research questions are investigated:

RQ1 How does Volvo Group work with variability for hardware and software?

RQ2 How can the demonstrator be upgraded to handle variability in hardware as well as in software?

The questions are answered in chapter 4 and 5 respectively.

1.4 Contributions

The contribution of this thesis consists of several upgrades to the demonstrator. These contributions are split into three areas:

Hardware The model trucks are upgraded with electronics to facilitate the new software variants.

Testing and Download Client A download client is developed so that the new software functionalities of the trucks can easily be controlled. A testing client is also developed as the new software variants require some upgrades to the testing practises.

Modeling of Variability A new way to model the truck variability is introduced, as well a a workbench which is used to manage the variability models.

1.5 Scope

This project was carried out between January of 2014 and June of 2014 at GTO in Gothenburg. The scope of this thesis was mainly to implement software download features for the demonstrator as well as investigating how variability can be modeled in a better way. The project was carried out in the form of literature studies, user centered design and implementation of new software as well as hardware.

1.6 Limitations

The project has had has certain limitations. One of the biggest challenges has been the significant size difference between LEGO trucks and real trucks. Given the fact that it can be both expensive and time consuming to order customized electronic components for the LEGO truck, only standardized pieces that are available to the general public has been used. At the same time, it is necessary to keep the LEGO truck consistent in appearance to the real trucks. This has lead to a limited amount of choices in terms of hardware as well as software.

The tools that are currently used for software download at the factories were considered for use in the demonstrator but were in the end not be used. This is because it was not possible to integrate these systems without the help of external expertise and this thesis project did not have these kinds of resources.

1.7 Ethical and Sustainability

One of the goals of the demonstrator is to use it for testing of new features. By first testing new features in smaller scale both time and resources can be saved. The environmental effect of discontinuing something that was found faulty on the demonstrator is much less than something that has been introduced in full scale in a factory.

By creating good models and rules for the variants the engineers working with putting orders together can work more efficiently. Because the engineers will be able to do a better job the work load for operators will be lowered as well. This is because the orders will be more correct and the final trucks will have a higher chance of passing all tests when leaving the assembly line.

2 Assembly Demonstrator System

At Volvo Group an assembly demonstrator system has been created as a way to train employees on how the assembly is done as well as demonstrate its features. This demonstrator is also useful as a tool to test new technologies in smaller scale, which would otherwise be to expensive to test on the real assembly line. In section 2.2 the requirements for this demonstrator are shown. Then, in section 2.3 the design decisions and the reasons why a physical demonstrator was chosen are discussed. Finally, in section 2.4, the features of the demonstrator are fully explained.

2.1 Purpose of the Demonstrator

The purpose of the demonstrator is to have a system that can be used to visualize and demonstrate the benefits of the Manufacturing Executions Systems (MES) tools. A second purpose is to give employees who are working with production—but not necessarily on the factory floor—a chance to understand the workflow in the factory. For example, by increasing the knowledge of MES among factory managers they would better understand the benefits of these kinds of systems, which would mean the usage of MES tools would increase even more.

MES is a set of IT tools used to increase increase productivity and quality in manufacturing. In [5] the benefits of MES is shown to be substantial. In this study several manufacturers using MES were investigated to show the benefits of MES. Several benefits arise from using these kind of systems, including decreased cycle time, less paper work and reduced lead time. At Volvo the process of introducing MES has been ongoing for some time now. MES tools are complex and can be expensive to introduce to a factory. It can also be difficult to show the benefit of such systems to people with limited knowledge of these kinds of systems. This is another reason why it is important to have a demonstrator.

The demonstrator was implemented to show current and upcoming features of the

MES system present at Volvo. The key features of the demonstrator are:

- to have a fully integration between professional grade MES tools and flexible and cheap LEGO components.
- to have a demonstrator that looks and handles realistically, and that the models used are realistic in terms of look and feel.
- to have a flexible system that can be adapted to any factory layout and/or specification.

2.2 Requirements

When the original demonstrator was created five requirements were set. These five requirements deal with reproducibility, cost, flexibility, portability and integration. Below each one of these is further explained.

Reproducibility

One of the requirements is to keep the reproducibility of the demonstrator high, so that it is possible to have new demonstrators created when needed. To fulfill this requirement extra care is taken during development to make sure everything is duplicable.

Cost

It is important to keep costs as low as possible. This also integrates with the previous requirement as it is easier to reproduce the demonstrator if the costs are not prohibitive.

Flexibility

The demonstrator should be flexible. This is important because the layout of the demonstrator needs to be adjustable to the factory layout and specifications.

Portability

The demonstrator needs to be as portable as possible. This is because of the importance of being able to move the demonstrator between different locations without having to rebuild parts of the system.

Integration

The last requirement is to have as much integration with the real Volvo systems as possible. By using the Volvo systems on the demonstrator the user experience will get much closer to the real factory experience.

2.3 Design

As mentioned in section 2.1 the purpose of the demonstrator is to show the benefits of MES tools as well as educate employee on how the assembly process works. To achieve this the system needs to be train people on how the assembly is done. This could be anything from training on the real assembly line in the factory to running a simulator software on a regular computer. Training on the real systems would be the best solution, however this would slow down production, which would not be good. This leads to there being two possible training environments left. Either doing the training in a simulator or in a physical training system. The system also need to showcase the benefits of MES which means it is important that these tools are well-integrated.

In [6] virtualized training is discussed. They proposed a game-based approach using a natural user interface. By using this method instead of current methods using mouses and keyboards they hope to make the user experience closer to the real experience. The systems they use (Kinect, WiiMote) offer a better user experience, but are still far from a perfect substitute to the real systems. Another way to improve virtualized training could be through the use of Virtual Reality (VR) headsets [7] which would further increase immersion. This would give the user the experience of being on location in the factory. In the paper they discuss the differences between the different levels of immersions and show that full immersion with VR headsets and advanced controls such as data gloves is needed for the systems to work satisfactory. The problems with doing simulator based training is that it is still very expensive and the technology is not yet mature enough to get full immersion. Another benefit of a physical system is the possibility to directly integrate it with the MES tools used by Volvo. Because of this a physical system was chosen as the best way to do the training. When building a physical system the big question to ask is which parts can be kept intact from the real factory, and what part need to be changed to still adhere to the requirements. By using LEGO for the trucks as well as the Automated Guided Vehicles (AGVs) the flexibility and portability of the system would be high and the costs would be low. Also, because of the use of standardized LEGO parts the the models would also be easily reproducible. However, for the computer systems that represent the MES tools, there is no need for changes, as they only require computer hardware to run the software. This means that all computer interaction can be constructed the exact same way as in the factories.

2.4 Features

The key parts of the demonstrator is its five stations and the AGV. The assembly is done on the the first three stations on the main line as well as the kitting station. On the final station the testing is carried out. To move the trucks between the stations the trucks are put on AGVs. These are carriers that automatically move between the stations. The behavior of the AGVs are controlled by the MONT systems. It secures the assembly process by controlling the shop floor equipment and guiding operators in every step of the assembly process. MONT wirelessly send messages that controls when the AGVs should move between the stations etc. At each each station there is also a system called Assembly Assurance System (AAS) which act as a user interface to MONTering, assembly in Swedish (MONT). The functionalities of AAS include showing the assembly instructions and reading the input from the barcode scanners. It is also responsible for securing quality in the assembly process as well as reporting process and product deviations. These two system are also used by Volvo in the real factories as part of their MES tools. That is the main reasons why those systems were used. The user experience has to be close to the real factory experience, and having the same interfaces helps a lot.

Another important part of the the demonstrator is the variants and variant combinations. A variant is a part that can be handled by the demonstrator, an example of this could be the engine or a wheel. The variant combinations on the other hand is a set of variants that constitutes a full truck. In Figure 2.1 design of two of the variant combinations can be seen. In the real factory there are thousands of different variant combinations, however to keep the demonstrator less cluttered it only contains 16 different combinations. In figure B.1 in the appendix the variant combinations for the demonstrator can be seen. This table also contain the software variants introduced in chapter 5. However, the hardware variants in the table are the same.

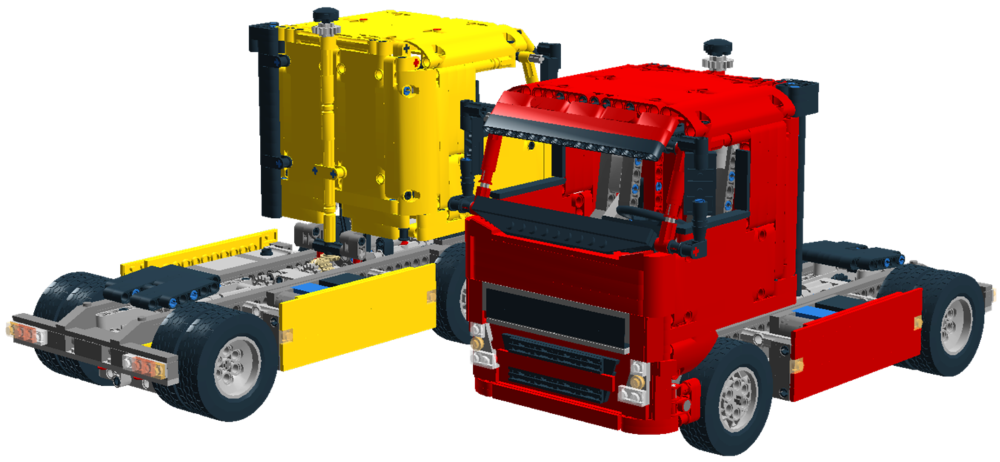


Figure 2.1: Overall design of a yellow and a red tractor truck, both of which are variant combinations that can be constructed on the demonstrator.

In section 2.4.1 the building blocks for the trucks and the AGVs are further explained. In section 2.4.2 the details of the demonstrator setup is explained.

2.4.1 Building Blocks

The model trucks as well as the AGVs are built from LEGO® bricks. The AGVs also use the MINDSTORMS® component for sensors as well as communication. The reason LEGO is used for the demonstrator is its flexibility when building models. Another benefit compared to other professional tools is that LEGO is cheap, modular and reusable [8], which provide a longer life span. Compared to using injection molding for the complete finished product you do not have to remold when creating an updated version. This leads to a shorter development process as well as lower cost. It is also not just a economical aspect, but a matter of sustainability as well. MINDSTORMS was used because of it ease of integration with LEGO, as well as its great community and documentation which makes it easy to get started.

2.4.2 Demonstrator Stations

The demonstrator consist of three assembly stations, a testing station and a kit area situated off the main line. All stations except the kit station are controlled by the MES system MONT. Below the purpose of each station is further explained. The station layout of the whole demonstrator can be seen in figure 2.2.

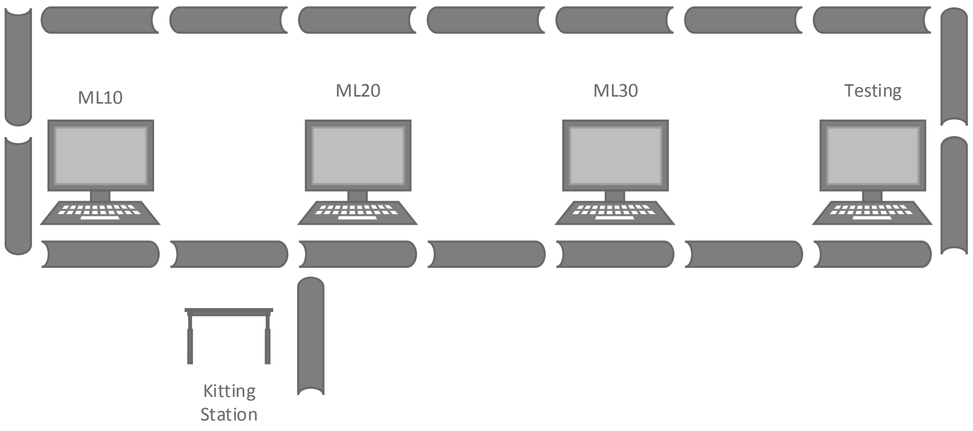


Figure 2.2: Layout of the stations showing the purpose of each station.

ML010

ML010 is the first station on the demonstrator. At this station the carrier gets linked to a specific frame. The carrier has a sensor that can read if it has a frame and which frame it has. If the carrier contains a frame the carrier sends a message to MONT containing the chassis ID. Because the frame is associated with a certain order MONT can then send assembly instructions to the AAS client, which in turn will display the instruction to the assembly workers on the screen.

After picking out the correct frame, the assembly worker is instructed to connect the axle module to the frame. All the components are also marked with barcodes to ensure that the worker always pick the correct components.

When the assembly worker confirms to AAS that all intructions are done AAS sends the results to MONT which then lets the carrier continue to the next station.

Kitting Station

As mentioned in 2.4.2 the kitting station is not part of the main assembly line. Instead it's a stand alone station working alongside the the line. At this station the assembly worker collects a number of components which with help of a visualization system are then assembled into a module. This module is sent to ML020 at the same time the carrier departs from ML010.

ML020

At ML020 the engine, fuel tank, kit and side covers are attached. The fifth wheel is also attached for tractor frames. Here the parts are also scanned to assure that the correct components are attached.

ML030

The wheels and the cab are attached at this station. Normally the cabs are sequenced, however, to simplify the process, the cabs are handled the same way as the other components in the demonstrator.

ML040

The last station is not used to attach any components. Instead it is used perform testing on the different components. This includes testing the driveline, tilting the cab, checking the symmetries of the wheels and looking for scratches on the surface of the cab.

2.5 Conclusion

This chapter has shown that, given the requirement, the best way to do assembly training is through the use of a physical system. Software solutions using motion controls and virtual reality were also considered but these were found to not yet be mature enough. The demonstrator has shown that it is possible to create a assembly training system that is small in size but can still work as a good substitute to training in the real factory. Using the same MES tools as in the factories is one of the main reasons why this was possible. However, there are still some improvements that can be made. This , for example, includes upgrading the demonstrator to handle software variants. This can be seen in chapter 5. By making these upgrades the demonstrator will become both a better platform for assembly training, as well as a better platform for testing the the variability modeling methods, which are also introduced in 5.

3

Modeling of Variability of Software and Hardware

In the introduction of this thesis the concept of product families were introduced. The purpose of the introduction of this concept was the need customization on a mass scale. This chapter will start by discussing different modeling methods in section 3.1 and then discuss different tool implemented using these methods in section 3.2.

3.1 Modeling Methods

When talking about variability modeling, there are different approaches on how to do the modeling. In [9] this was split into three different categories. These three are the case-based, model-based, and rule-based approaches. In the following subsections these three methods will be introduced.

3.1.1 Case-based

Cased-based systems are based on the concept of case-based reasoning [10]. Case-based reasoning is when you base your decisions on previous experiences. In a Case-based manufacturing system this would translate to all previously sold products being stored as cases. Then, when a new product is created, the old cases are used as a basis for the new product. If a new feature need to be added it then has to be manually or automatically added depending on how the system was created. The benefit of case-based system is that there is not a lot of work needed to get started creating a product platform. If new products are created relatively seldom this system is also beneficial as the step that takes up the most resources is when a new product is introduced. However when the scale of production increases this method get less and less viable compared to other methods where the platform is more carefully designed.

3.1.2 Rule-based

Rule-based systems [11] use rules to control what configurations are valid. The rules are presented in the form of simple IF *condition* THEN *consequence* statements. One of the problems with a rule-based systems is that the rule-system is tightly linked to the domain experts knowledge, which leads to a system that can be difficult to understand and use for non-experts.

3.1.3 Model-based

Model-based systems were created to address the limitations seen in cased-based and rule-based systems. The main concept of of a model-based system is that there exists a model that defines how the system is configured. As seen from this, the model-based category is a bit more general than the other two categories and is often split into further sub-categories. These can be based on description logic, features, answer set programming, preference programming, constraints etc. In this thesis we will further explain the constraint-based systems as these are currently used at Volvo Trucks [12].

Constraint-based systems [13] are based on using constraints to control which configurations are valid. A restriction represents which two features are disallowed together. An example of this can how Volvo handles their variability. All the features, which are called variants at Volvo, are grouped together in variant families. A variant family represent a choice during the specification of a truck, this could be the choice of engine for example. A variant family contains at least two variants and usually only one variant is picked. The restriction are then used to restrict which variants can be combined. For example, a heavy truck would be restricted from having a light engine as it would not be powerful enough for the weight of the truck.

3.2 Modeling Tools for Model-based Methods

In this chapter different modeling tools and methods will be introduced. These are useful to understand to better understand the choices made in the subsequent chapters of this thesis.

3.2.1 Feature Models

A feature model [14] is a form a tree structure that represent the members of a product family. Each split in the tree represent a choice. The relationship between a parent and its sub-features can be one out of five different types.

And all sub-features must be selected.

Alternative only one sub-feature can be selected.

Or one or more can be selected.

Mandatory features that required.

Optional features that are optional

A specific product line member is defined as a choices made from the feature model. A feature model is hierarchical which means the choices at the top of the tree is made first. Feature that are the children to a feature that was not picked can thus not be picked.

3.2.2 Common Variability Language

Common Variability Language (CVL) [15] is a language used to model variability. A CVL model consists of two models, a base model and a variability model. For example, this could be a Unified Modeling Language (UML) diagram showing all features available in a software product line. This base model can be created using any MOF-compliant language [16]. The variability model is what controls how features can go together. CVL models consist of a tree structure which can contain three kinds of objects. These are *choices*, *variables* and *restrictions*. A choices work the same way as a choice in a feature model. The variability model and the base model is linked together using variation points. Each variation point links exactly one object in the variability model to exactly one object in the base model. The resolution model, which represent the choices in the variability model, is what control the variation points. When the variation points are set a resolved model can then be generated.

3.2.3 Clafer

Clafer [17] is modeling language that can be used to create meta-models (such as UML models), feature models or a mixtures of both. The main benefit of Clafer is that you can combine features that are normally not available together. For example, you could make a feature model that also incorporates references. The following is a Clafer model of a student:

```
abstract Person
    Name -> string
    Surname -> string
    DateOfBirth -> string
abstract Student : Person
    StudentId -> string
    xor Program
        ComputerScience
        Chemistry
    or Courses
        Calculus
        Programming
        OrganicChemistry
    [ !(Programming && OrganicChemistry) ]
```

As seen in the example the student is represented as a feature model using the Program and Courses as the possible choices in the the tree. There is also also a reference and a restriction in the model. The restriction here makes sure that is not possible to be registered to a programming course and a chemistry course at the same time. Lastly, the reference says each student is also a sub-class to the a person. Another benefit of Clafer is that it also includes some powerful tools for analysis of the models. For example, it is possible to generate the instances of a model when you are modeling variability.

3.2.4 Comparison of Modelling Approaches

When comparing the rule-based approach to the model-based approach it is clear that main difference is that you are less constrained in terms of how the models are created in a model-based approach. For example, with a rule-based approach it is only possible to create simple simple IF *condition* THEN *consequence* statements, while in a model-based language such as CVL it is possible to use any boolean statement. The case-based approach on the other hand, is quite different from the other two. It is the simplest method and requires little initial investments. Although the upfront costs are low, the overall cost will often be higher as the cost of the incremental development is high. However, for small scale production, a case-based approach may still be the best option as the incremental development would not account for such a large part of the costs.

4

Current Work Practices

Research question 1 will be answered in this chapter. It states: *"How does Volvo Group work with variability for hardware and software?"*. More specifically this chapter will show when, where and how variability is handled as well as how Volvo like to work with variability in the future. This reason this was investigated was because it helps in understanding how variability is currently handled and what can be improved. Without having this prior knowledge it is difficult to improve the demonstrator in way where the user experience still resembles the real factory. These questions will be answered through a mixed-method approach. Interviews were carried out with the Volvo employees responsible to the modelling, implementation and testing of software variants. A literature study was also carried out covering the modeling of variability at Volvo. This mostly included internal Volvo documents regarding the different systems and tools used to handle the modeling and variants.

The chapter start by discussing how people at different positions work with and handle variability in section 4.1. Then in section 4.2 the modeling of variability is discussed. This section starts by introducing the system used at Volvo to handle variants and then introduces the rules used to restrict variant combinations. The chapter ends by discussing testing in section 4.3 as well as future work practices in 4.4.

4.1 Handling of Variability

Variability is handled at many levels throughout Volvo Group. In this section the work practices involving variability will be discussed for different users. As seen in this section, a lot of users from different background work with variability. Because of this it is important to have a user interface that is adjustable to a persons preferences or expertise. Methods to improve this is further discussed in chapter 5.The focus of this section will be on product development, manufacturing, testing as this corresponds to the parts that are improved upon in this thesis.

Product Development

The developers handle variability at several levels. When creating a variant it is important to take in to consideration how it would interact with other variants. If a variant is designed in a good way it can be used on more truck variants as well as be combined with more variant. Product development is also responsible for creating the technical rules. These are the rules that control which variants can be used on a specific truck model, as well as how the variants can be combined. When creating the technical rules, it is also very important to design these in a good way as it reduces the complexity. The developers are also responsible for the development and maintaining of the modeling systems. Without a good Product Data Management (PDM) system it difficult keep track of all the variants and the rules.

Manufacturing

During manufacturing the assembly workers need to have the competences needed to handle the high amount of variants used in the factory. The design of factory rules are could also be said to be a part of manufacturing, although these are not created during manufacturing. Even though the assembly workers are not working with the design of variants, they still need to keep track of the variants handled at their station. If they frequently would need to read documentation on how certain variants are installed, then the assembly line would be slowed down significantly. The test operators also has to keep track of the variant tests in a similar way to the assembly workers, as slow testing would also cause the assembly to slow down.

Testing

Because of the high variability, the testing will also vary between different trucks. As the tests are linked to the the variants, the operators are indirectly working with the variability. Because of this it important that the test operator have a good general understanding of of the tests that are required for the different variants.

Marketing

A part of the marketing process is to control which variant combination are valid on which markets. These are combination that are technically possible to create put for certain reasons should not be available at certain markets. With the use of marketing rules the people responsible for marketing can limit range of product in a similar way as the developers do when designing the trucks.

4.2 Variability Modeling

To be able to handle variability satisfactory a way of modeling the variability is needed. As previously mentioned in section 3.1.3 Volvo Group uses constraint-based modeling [13]. The way this works is that you a set containing all possible variant combinations and then use rules to restrict which combinations are not valid. At Volvo Group these models are contained within a PDM [18] system called KOLA [19].

Contrary to a car, a truck is offered with more choices when it comes to configurations. To make this simple, both for the customers as well as engineers the variability needs to be modeled in a way that would make it easy to understand both for the customer as well as the domain experts. From the customer perspective this would mean making a number of choices from a set of features. An example of this can be shown in figure 4.1. When combined with the other choices the customer needs to make, these choices becomes the customers order.

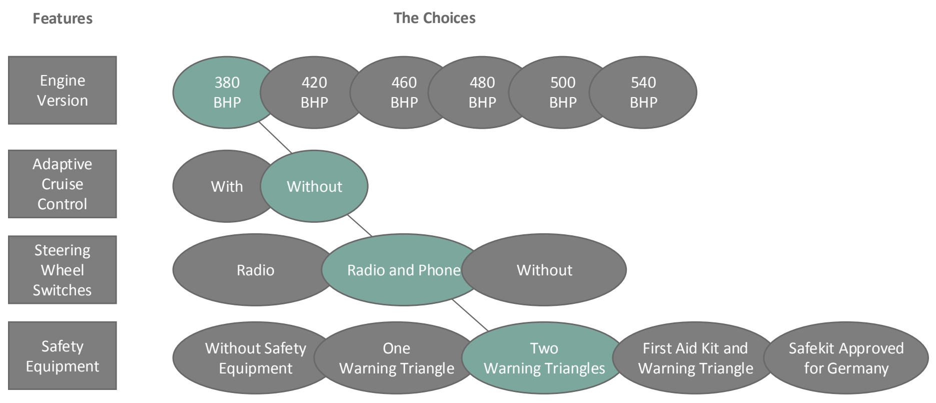


Figure 4.1: Example of possible choices a customer can make.

In KOLA logic, these features are called *variant families* and the choices are called *variants*. This variant structure is shown in figure 4.2. However, this is a technical point of view so it is not necessarily the same features as those visible to the customer. An example of this could be a Hot Environment feature. This would contain several technical features such as cooling systems, special batteries and battery chargers. In the Variant Structure, these are represented by Variants, not the summed up customer feature. However, each one of these technical features are represented by a variant family which reflects the perspective of the customer. For example a variant family for the size of a radiator would be called Cooling Capacity instead of Radiator Size.

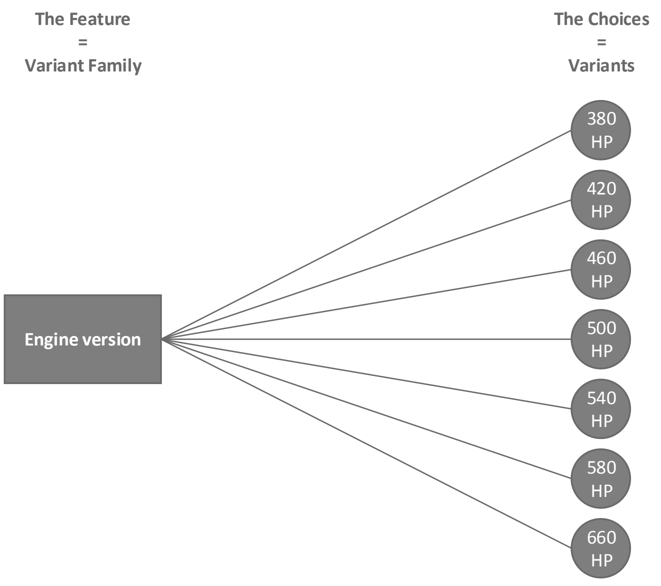


Figure 4.2: The Variant Structure.

A problem still exist with this way of modeling the variability. That is the problem of knowing which variants can be combined and which can not. Two types of rules are used for this. The first type is *authorizations* and the second type is *exclusions*. Authorizations are general rules while the exclusions are adjustments of the authorizations. These can be created from a technical, a market or a factory point of view. What is meant by general and adjustment rules is that a general rule is defined for specific variants while the adjustments affect how the variants are handles together.

The authorizations restricts:

1. which variants are requested and are available on which truck models.
2. which variants can be sold where.
3. which variants can be managed in which factories.

In equation 4.1 a logical expression of an authorization can be seen. $vf1.v1$ here represents the variant $v1$ in the variant family $vf1$. What this expression does is that it restricts the usage of variant $v1$ in variant family $vf1$ by negating it, which means $\neg vf.v1$ is true if and only if the variant is not chosen.

$$\neg vf1.v1 \tag{4.1}$$

The Exclusions restricts:

1. which variant combinations are not requested or cannot be used.

2. which variant combinations cannot be sold on specific markets.

3. which variant combinations cannot be managed in specific factories.

In equation 4.2 a logical expression can seen which models an exclusion. The variants here are represented the same way as in equation 4.1. In the expression a conjunction is made between the two variants $vf1.v1$ and $vf2.v2$ which is then negated. This means the expression is true for all cases where the variants are not used simultaneously.

$$\neg(vf.v1 \land vf2.v2) \tag{4.2}$$

There is also a third rule type, called inclusions. With inclusions it is possible to group together variants and then define rules for these a a group. The combined variants are called upper variants while the building blocks are called lower variants. The benefit of the inclusions is that it can used as a way to structure the variants as well as a way to reduce the number of rules. The logical representation of an inclusion between two variants can be seen in equation 4.3. What this expression says is that if a certain variant $vf1.v1$ is selected then the variant $vf2.v2$ must also be selected. An example of when an inclusion could be used is with the case of the variants that are grouped together at the kitting station on the demonstrator. For example, if one of the parts in the kit is select all other parts should also be selected. A benefit of of using this rule is when a kit is mounted at a certain spot, then that spot can not be used for other components. Instead of separately putting this rule on all variants that are a part of the kit, these are instead grouped together.

$$vf1.v1 \Rightarrow vf2.v2 \tag{4.3}$$

These rules describe how to deal with variants from three points of view. From an engineering point of view the technical rules are the most important rules. For example the technical authorizations show which variants are available for which product types. A technical restriction, which is an exclusion rule, shows which variant combinations are forbidden from a technical standpoint. These technical rules are created by the designer. There are also equivalent rules for marketing as well as rules regarding the factories. The marketing rules are created by product planning while the factory rules, which show which variant can be handled in a specific factory, are created by the factory manager.

As previously explained, a variant family is a group of variants representing possible choices for a feature. As was also mentioned earlier the features are designed to have names that are easy to understand for the customer. For a customer to be able to make the choices the variants also needs to be easy to understand. Take the example of the variant family Steering Wheel Switches. This contains the variants Radio, Radio and Phone and Without. These variants are built from other variants using the inclusion tool discussed earlier. These upper variants are also named to be easy to understand while the lower variants may have more technical names. The rules associated with these upper and lower variants are what controls which component will be part of the truck after a variant has been chosen. For example, depending on the size of the cab the radio might be situation at different positions which would mean a different components is needed to fasten it.

4.2.1 Software

The way to model variants as described in section 4.2 works great for physical components. However to use the same system for the software some modification to the underlying implementation of the software variants is needed. The way this is done in KOLA is that software variant is made to looks the same as physical variants, but beneath the surface it works differently. Each software variant consist of a number of Diagnostic Object Identifiers (DOIDs). These DOIDs act as a kind of constant in the software. They are used as a way to configure different algorithms or controlling functional settings (switch case, for loop, if statement etc.) in the Electronic Control Unit (ECU) software. By doing it this way it is possible to have the same software for all ECUs of the same type, with a table that controls which functionalities are activated. For each ECU the set of possible DOIDs are defined in a tool called System Engineering Web Server for TEA2 (SEWS2). For the Vehicle Master Control Unit (VMCU) there are over 200 DOIDs. By taking the software variants as input SEWS2 can then generate a table for a specific ECU. Then when the ECU is about to be programmed both the software and the table are downloaded into the unit.

This way of modeling the software so that all variants can work the same in KOLA has its benefits. For example, it is possible continue using KOLA the same way without doing any major modification to. Another benefit is that the same software is always be downloaded, as it is only the table controlling the DOIDs that is unique. However a disadvantage to using this method is that the software packages gets large. This can lead to long download times as well as a need for high storage capacity on the ECUs.

4.3 Testing

At Volvo Group several levels of testing is done. This includes everything from simulations to physical test on the finished trucks. At the beginning, before the trucks are assembled, the software variants on the ECUs are tested. Later, for the finished trucks tests are also run, but in this step only a selection of the variants are tested. This limit is due to time constraints on the assembly line. As soon as the battery is connected it is possible to start running these tests. This is done by connecting a diagnostics tool that will run the tests while the truck is being assembled.

When the truck is fully assembled these software test will continue as well as some physical tests that will be run on the truck. In addition to these functional tests some environmental tests will be run. This includes for example vibration endurance tests and environmental tests on electrical components and sensors. These are normally performed in different test rigs or complete trucks.

4.4 Future Work Methods

In this section future work methods at Volvo Group will be discussed. These will be split up in two categories which are short and long term goals. The short term goals are goal that could currently be implemented, while the long term goals still need other other technologies before they can be implemented.

4.4.1 Short Term Goals

Some factories has adopted the paperless assembly instruction system that is also used by demonstrator. A big goal is to make this is used in all factories. The demonstrator is also part of the short term goals. Its main purpose is to use it for assembly training, but it will also used to test new features. By first testing new features on the demonstrator these features could then, if successful, be introduced in small scale to factories, and eventually full scale. By incrementally moving features up this scale risks could be lowered significantly. In figure 4.3 this can be seen. By having the possibly to test in small scale, new features that otherwise would be part of the long term goals, could be tested much faster.

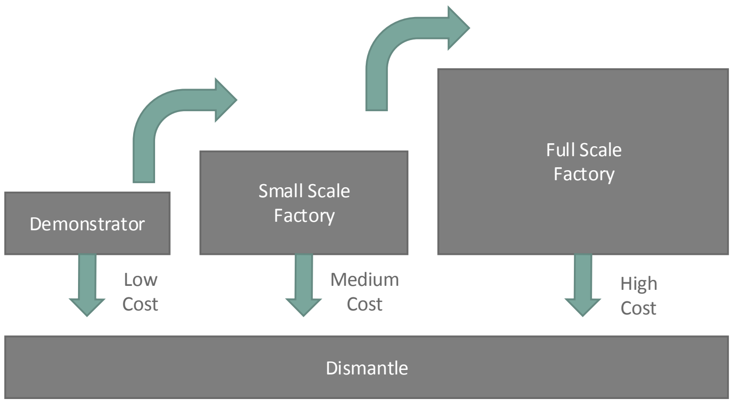


Figure 4.3: Illustration of the iterative implementation process.

4.4.2 Long Term Goals

In the long term new systems working together with KOLA could be introduced. A possible solution for simplifying the interaction with KOLA is presented in Chapter 5. This new system was created to be both easy to use as well as flexible in terms of upgrades. The benefit of this system is that it is able to present its data in a designated way for each user group, without making it too simplistic or removing necessary functionality. The system presented is not yet connected to KOLA but could, with some additions, be connected.

As explained in chapter 4 all software for a type of ECU is downloaded into each unit, with certain variables controlling the behavior of the code. By being able to generate specific software for each ECU the code could be customized for each ECU as well as a lot of overhead being removed by not having to download all the software each time. Depending on the benefits gained from these changes it might be possible to move the software download to the main line, as it would be so fast that it would not slow down the line.

Another long term goal is to further improve the testing. As seen in section 4.3 only a selection of the software variants are tested when the trucks are assembled. To improve this, fully automated test rigs with configurable simulation environments are being researched. The goal of this would be to find faults that would otherwise not happen if the ECUs are only tested individually. These rigs would be used to find the faults in the customer configurations before the trucks have built.

4.5 Conclusion

In this chapter it has been shown how variability is handled throughout Volvo Group. It has also been shown that variability is a part of a lot more than just the trucks design. Because variability is not only part of the design process, it is even more important than was previously believed. This means there are many things that need to be taken into consideration when creating the variability models. For example how the testing is affected as well as what impact the modeling has on the assembly process. There a many way of modeling variability and the method that is used by Volvo Group is a constraint-based method. The details of this approach is further explained in chapter 3. One of the drawbacks of this method is that you are restricted to only a small set of rules. This can make the models both difficult to write as well as understand. It can be likened to the difference between a high-level programming language and a low-level language. A high-level modeling language can be easier to use, as well as make the models simpler and more understandable. In chapter 5 a new way of modeling the variability is introduced using high-level modeling language.

5

Implementation of Updated Demonstrator Platform

In this chapter the research question ”How can the demonstrator be upgraded to handle variability in hardware as well as in software?” gets answered. This research question was chosen as it combines the requirement from Volvo Group of upgrading the demonstrator with with variability modeling. As seen in the research question the demonstrator has already been chosen as the way to do assembly training at Volvo [4]. However, there is still value in discussing other possible solutions.

The methods used to to answer the research question mostly consists of a literature review. This literature contains some scientific papers, but due to the nature of the subject a lot of the the material consists of data sheets and technical information regarding languages and libraries etc. Some of the design decisions were also based upon the information gained during the interviews mentioned in chapter 4. Contributions were made to the demonstrator in three areas.

Hardware The model trucks were upgraded with electronics to facilitate the new software variants.

Testing and Download Client A download client was developed so that the new software functionalities of the trucks could easily be controlled. A testing client was also developed as the new software variants required that the testing station was upgraded.

Modeling of Variability A new way to model the truck variability was introduced, as well a a workbench which is used to manage the variability models.

The chapter starts by introduing the requirement in 5.1 and then discusses the overall design decisions in section 5.2. Then in section 5.3 through 5.5 the three areas of contributions are discussed.

5.1 Requirements

The requirements set for the original iteration of the demonstrator still apply. These 5 requirements deal with reproducibility, cost, flexibility, portability and integration. In section 2.2 in depth information on these requirements are given. To answer the research question some further requirements had to be set. The first new requirement is that demonstrator has to be able to handle software variants in addition to the hardware variants. This requirement is based on the requirement set by Volvo which was that they wanted software download capabilities on the demonstrator. This, combined which the goal of wanting the demonstrator to exhibit a wider range of the capabilities of the real assembly line lead to the addition of this requirement. The second new requirement is that the demonstrator should handle all the steps of production. This means that all steps from the design of truck specifications to the final testing should be part of the demonstrator.

5.2 Design Decisions

To fulfill the requirements set in 5.1 the demonstrator need at least two upgrades. Firstly, the demonstrator needs to be physically upgraded to handle software. This include upgrading the model trucks to handle software variants as well as creating the software for the trucks. Secondly, software download capabilities needs to be added to be able get the software onto the trucks. These are the basic upgrades needed to make the demonstrator capable of handling software variants.

When upgrading the demonstrator to handle software variant, several choices need to be made. One option is to upgrade all the systems that handle software variants as well as adding software capabilities to the trucks. This means it would be possible to look at a finished truck, and based on the differences in behaviour of truck, see what software variant it contains. The second option would be to only upgrade the systems that handle the variants, but not the trucks. By choosing this option, it would only be possible to have dummy software download and testing. This leads to the user experience not being as good as there can be no feedback from the download and testing. Because of this it was chosen to fully implement the software features in the trucks.

These software functionalities on the trucks were chosen be controlling lights and sounds as that would make it possible to create several different variant combination, which at the same time would easy for the user to differentiate. Another important design decision was whether to put all components inside the cab or spread them out on the chassis. Having the component spread out would make it possible to have lights outside the cab, however because of the difficulty of fitting cables and components without them being visual it was chosen to put all components inside the cab. The upgrades to the cab are further explained in 5.3. Another benefit of having all the components at the same place was that there was no need to create a communication network between the different modules.

For the download of the software, a system called PROSIT had been used in the factory for downloading and testing. Originally, this was considered to be used on the demonstrator as well. However because of the difficulties in porting this to to the demonstrator environment without external expertise, it was chosen to create a new set application mimicking the behaviour of PROSIT system. The design of these application are discussed in section 5.4.

The second new requirement for the demonstrator is that it should show the full process from creating specifications to the final testing. The thing that needs to be added to meet the requirement the modeling and handling of the variants. For this, the workbench [20] used in [21] was chosen to be used as a way to add better variability management and modeling to the demonstrator. In this workbench a the Clafer language is used to model the rules that control the variability. The reason this was chosen to model the variability in the demonstrator was that instead of a standard set of rules as seen in KOLA, Clafer offer a more flexible way to design the rules. The workbench also offer great management of variability. What makes the workbench good at this is that user input and output can be adapted to each users preferences. In section 5.5 the variability modeling and the full implementation of the rules for the trucks is discussed. Later in section 5.5.3 the workbench as a whole is further discussed.

5.3 Hardware Upgrades

Many different upgrades of the cab has been considered. Mainly adding Light-Emitting Diodes (LEDs) to parts of the cab and the chassis. In the beginning several ECUs were considered but in the end it was chosen to only have one ECU. The reason for this was that it was deemed unnecessary to make the trucks too complex. This was because the main objective was not to make the trucks as feature complete as possible, but to have a demonstrator that was user friendly and captured the user experience of the assembly process in the factory. Thus, in the end all features were chosen to be added to the cab. The cab was chosen as the best place to put the features as it is the spot with the most space on the truck. In figure 5.1 a truck with the upgraded cab can be seen.

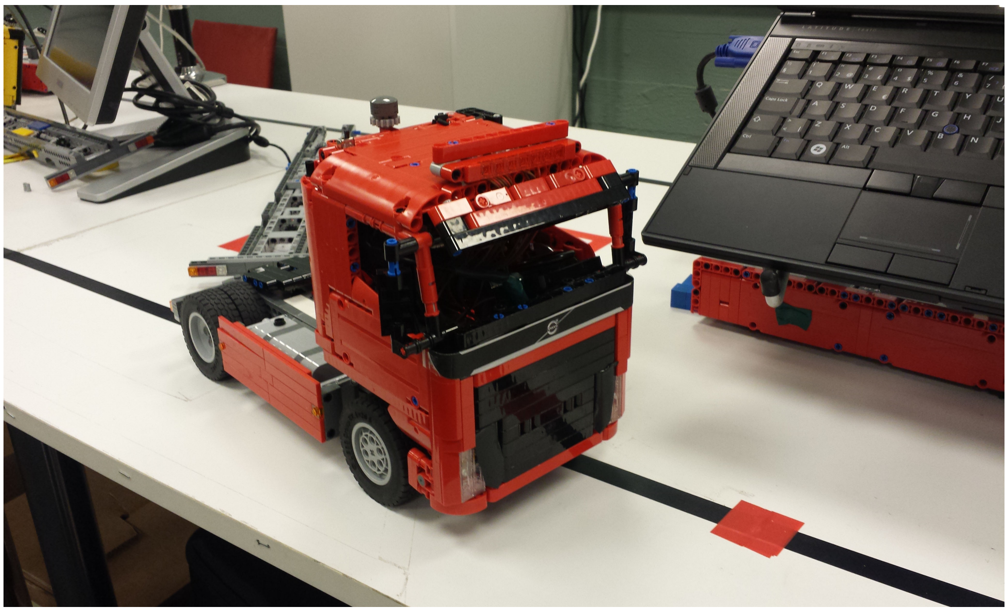


Figure 5.1: Picture showing the truck model with the upgraded cab.

Two main functionalities has been added to the cab. These are sound and light capabilities. The LEDs are used as the truck lights while the sound is used for the engine and horn sounds. A miniature 8 Ω speaker is used for the sound capabilities. For the lights a set of white and orange LEDs are used. An Arduino board is used to control the LEDs and the speakers. The Arduino platform was chosen because of the ease of use of both the hardware and the associated libraries.

To be able to control these functionalities with the Arduino board some intermediary components are needed. For the sound to work properly an amplifier as well as capacitors were added. In figure 5.2 the circuit diagram for the solution is shown. The amplifier is connected to the Arduino through the DAC0 port. The properties of the LM386 amplifier can be found in [22].

Because audio was needed for the engine sounds the Arduino Due [23] board was chosen. This board is larger and more expensive than the standard Arduino Uno [24], but since the Arduino Uno does not contain any analog output the Arduino Due had to be chosen instead.

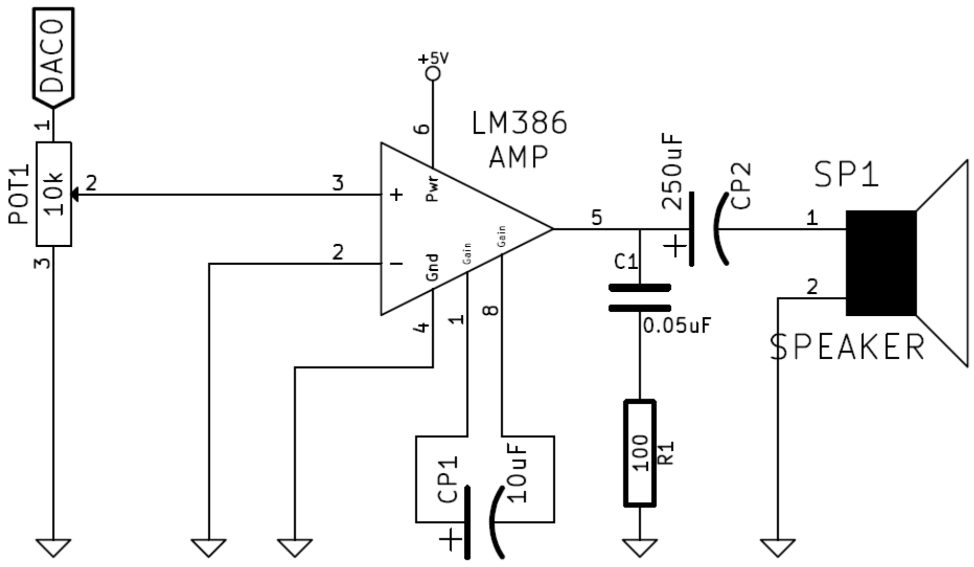


Figure 5.2: Sound amplifier circuit diagram

Neither the Arduino Due nor the Uno contain enough flash memory to store sound files so the Arduino ethernet shield [25] was added because of its built in microSD reader. With the shield connected to the Arduino it is possible to store all the sound files on an microSD card instead of the small onboard flash.

Although a small amount of LED can be powered off of the I/O pins, the amount needed for the cab exceeds the maximum current of the I/O pins. Figure 5.3 shows the circuit used to power the LED.

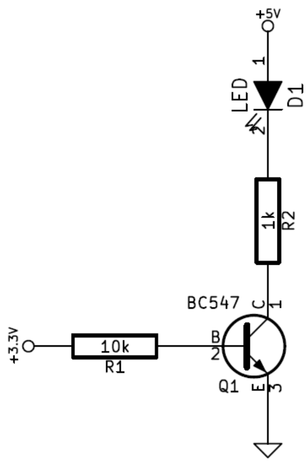


Figure 5.3: Circuit diagram showing the transisor logic used to power and control the LEDs

When a logical zero is supplied to the I/O pin on the circuit there is no flow of current between the base and the emitter which results in no flow between the collector and emitter. By instead switching the voltage to a logical 1 current would flow between the collector and emitter. As shown in the circuit diagram this would result in the 5V pin powering the LED instead of the I/O pin on the Arduino. The 10 kΩ resistor between the I/0 pin on the Arduino and the base results in a low current drawn from the I/O pin, while the the 1 kΩ resistor between the LED and the collector results in a high enough current to power the LED. The 1 kΩ resistor can be changed to an even lower resistance if the LED requires a higher current.

In the cab there is 15 different LEDs. These include:

- 7 white LEDs for the beamer lights.
- 2 white front lights.
- 2 orange indicator lights.
- 2 white interior lights.
- 2 white working light

Each of LEDs require one of the circuits shown in figure 5.3. However, the resistors and transistors has been combined on one Printed Circuit Board (PCB) to save space. The LEDs that are close to each other are also sharing 5V cables.

5.4 Download and Testing Client

The download client is used to download the correct software to the cab, and the testing client is used to test the software. They have been created to mimic the behaviour of the PROSIT application used at Volvo Group. The PROSIT application contain more functionalities than these applications. However, to keep the demonstrator as simple as possible, the functionalites of the download and testing clients have been limited to only containing the essential functionalities.

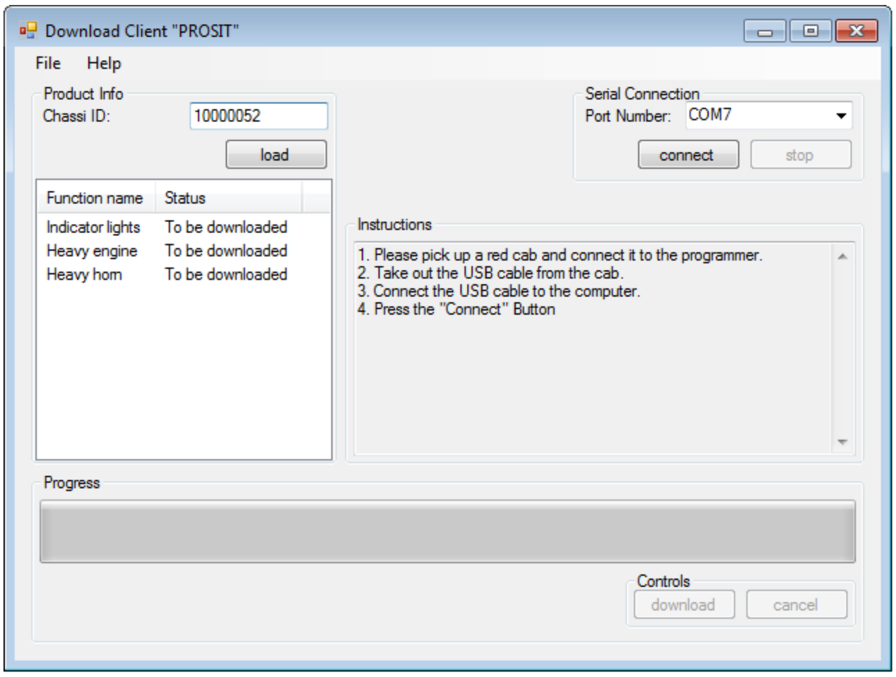
These clients were chosen to be written in C# mainly because of three reasons. Firstly, the client was to be run on the windows platform which meant that C# was one of the logical choice. Secondly, the .NET framework platform contains libraries that can be used add all the functionalities needed in the application. The last reason is that the updated version of the AAS client is also written in C#. As these two systems are going to be managed together having them be written in the same language is good.

5.4.1 Download Client

In figure A.1 in the appendix the flowchart corresponding to download client can be seen. When starting the download client the client print a message that says the user should scan the barcode located on the cab. This action triggers a number of events. Firstly a config file associated with the chassis ID is read. This config file contains the string mentioned in 5.5.2 as well as information on what cab should be picked for that specific truck. An example of this config file can be seen below.

```
rABCDEGI
red
```

The list of functionalities is also added to a list view in the client, giving the user an overview of the functionalities associated with that specific truck. In Figure 5.4 the visual appearance of the client can be seen.



Chassi ID: 10000052

Port Number: COM7

Function name	Status
Indicator lights	To be downloaded
Heavy engine	To be downloaded
Heavy horn	To be downloaded

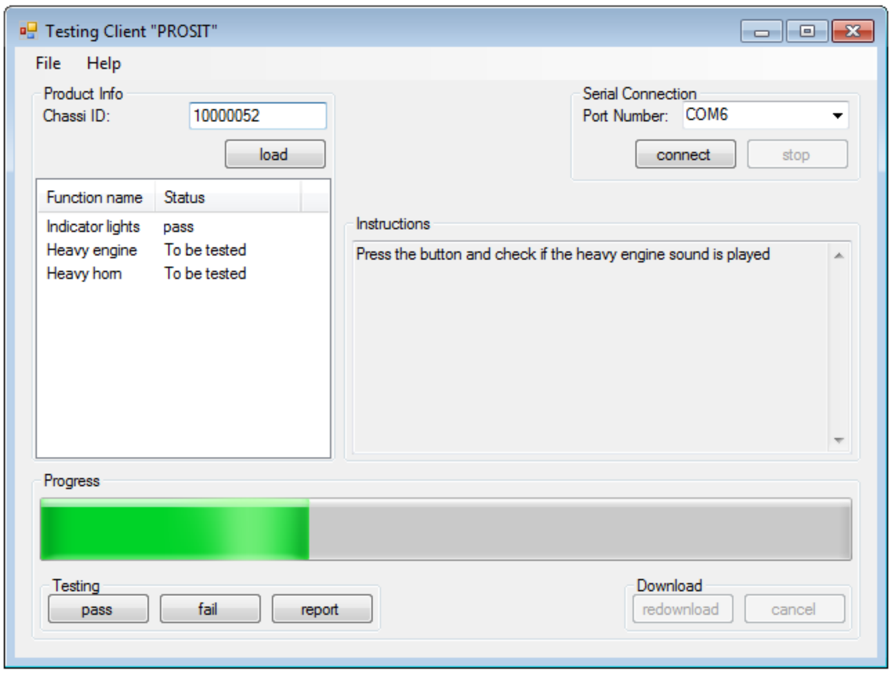
Instructions

1. Please pick up a red cab and connect it to the programmer.
2. Take out the USB cable from the cab.
3. Connect the USB cable to the computer.
4. Press the "Connect" Button

Figure 5.4: Screenshot of the download client after the chassis ID has been scanned.

5.4.2 Testing Client

The main purpose of the testing client is to guide the user through the process of testing the software that has been downloaded with the download client. Figure 5.5 shows a view of the testing client. The testing client can also redownload the software if necessary, as well as save error reports. Because of the redownloading the testing client contains a lot of the same functionalities as the download client. The behavior of the testing client can be seen in the flowchart in figure A.2 in the appendix.



Chassi ID: 10000052

Port Number: COM6

Function name	Status
Indicator lights	pass
Heavy engine	To be tested
Heavy horn	To be tested

Press the button and check if the heavy engine sound is played

Figure 5.5: Screenshot of the testing client showing the view during a test run.

5.5 Variability Modeling

This section will showcase a new way of modeling truck variability. It will start by discussing how this method was chosen, and then in section 5.5.1 the implementation of this model is explained. This new model uses the new software variants which are introduced in 5.5.2. Lastly, in section 5.5.3, the workbench is introduced, which is used to handle the variants and then new models.

In KOLA the variability is modeled using variants that can be combined into variant combinations that represented the trucks. In addition to this rules are used to remove variant combinations that are not wanted. For the current demonstrator the handling of variability consists of a list of a variants and a feature diagram showing which variant combination are valid and what variants they consist of. This is not a good method of modeling the variability as it requires each possible variant combination to be defined separately, instead of just defining which variants can not be used together etc.

CVL was considered as the modeling method for the demonstrator, however, the main functionality needed for the modeling was not the visualization. Another problem with CVL was that the models easily get cluttered when modeling a system with high variability. This is because the UML part of the CVLs model always has to contain all possible variants. The language that was used instead is called Clafer [17]. In comparison to KOLA rules can be used to define which variant combinations are valid. However for Clafer the rules work somewhat differently. With Clafer, instead of starting with all variant combinations and then removing unwanted combination, you define a set of conditions that must be met. These rules are written using logical expressions. The benefit of this is that rules can be written however you want instead of being locked into a set of predefined rules as used in KOLA. This is also another reason why Clafer was picked over CVL, which is much more restrictive on how the model are created. In section 5.5.1 the design of the rules for the demonstrator is further discussed.

5.5.1 Truck Model

The visualization gained from using CVL is great, however, because the main focus was modeling of variability and not visualization another modeling language was chosen. Compared to a feature diagram, a clafer model can be created with the variant rules being the main focus. When having the focus on the rules it is easy to see why certain variant combinations exist and why some do not exist. From the model created in clafer it is also possible to generate a feature diagram if needed.

In the following example a Clafer rule can be seen:

```
[(Heavy_engine && Heavy_horn) || (Light_engine && Light_horn)]
```

What this line of code does is that it defines a logical expression that must be fulfilled for all combination. The meaning of the expression is that a truck must either have a heavy engine and a heavy horn, or a light engine and a light horn. Although, the rules are not enough to have a full Clafer model, all variants also need to be defined. The variants are defined in features groups.

The following is the software variants for a truck on the upgraded demonstrator:

```
xor Horn_sound
        Heavy_horn
        Light_horn
xor Engine_sound
        Heavy_engine
        Light_engine
or  lights
        Interior_lights
        Working_lights
        Beamer_lights
        Rotational_lights
        Front_lights
        Indicator_lights
```

As seen in the example above each group has a prefix. This is a simple rule that controls which variants in a group can be used together. A full Clafer file contains these group definitions as well as a number of rules.

5.5.2 Software Variants

In Table 5.1 the 12 different software variant combinations can be seen. These variant combinations were created by setting up certain rules that restricts which variant combinations are valid. The rules are then analysed by the Clafer tool which gives the variant combination. In section 5.5.1 the process to create these rules are further explained. For comparison, in the real Volvo systems, variant combinations are instead restricted by using inclusions and exclusion. These are discussed in chapter 4.2.

Variant Combinations		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Unit Number	**Software Variants**												
18	front lights	x	x	x	x	x	x	x	x	x	x	x	x
19	interior lights	x	x			x	x			x	x		
20	indicator lights	x	x	x	x	x	x	x	x	x	x	x	x
21	working lights	x		x		x		x		x		x	
22	beamer lights	x	x	x	x								
23	rotational lights					x	x	x	x				
24	heavy engine	x	x	x	x	x	x	x	x				
25	light engine									x	x	x	x
26	heavy horn	x	x	x	x	x	x	x	x				
27	light horn									x	x	x	x

Table 5.1: Table showing all valid variant combination

Each of the software variants control either some LEDs or the speaker. The behaviour of these variants are defined in software on the Arduino Due. In the following list the behavior of each variant is listed:

front lights steady light from two white lights.

interior lights steady light from two white lights.

indicator lights 2 blinking orange lights used when turning.

working lights steady light from two white lights.

beamers lights 7 white lights with orange filters showing a steady light.

rotational lights 7 white lights with orange filters showing running lights.

heavy engine Plays a ”heavy” engine sound.

light engine Plays a "light" engine sound.

heavy horn Plays a slow horn sound.

light horn Plays a faster high pitch horn sound.

These software variants are stored on the Arduino Due board. Because the software variants are stored on the board a solution is to activate and deactivate them is needed. This consists of a table that controls which variants should be activated. To update the table a simple message is sent to the Arduino board through serial communication. The message looks like this: *rABCDEGI*. The "r" tells the Arduino to reset and each other letter controls which variants should be activated. In the Volvo system, these letters would correspond to the DOIDs discussed in section 4.2.1. However, the way it is used here is not as advanced as it only controls whether the variants are activated or deactivated.

The on board software is written using the C programming language. The way it is implemented is that each variant is written as a separate function that showcases a specific variant using the appropriate LEDs and sounds. To showcase each variant there is a loop that cycles through these functions. If a function is marked as activated, then that function will be run, otherwise the next function is run instead. The button on the Cab is used to step between these functions. This makes it easy to go through the variants during testing.

5.5.3 Modeling in a Language Workbench

The purpose of a language workbench [20] is to manage data. The benefit of a language workbench is that data can be controlled using different Domain Specific Languages (DSLs)[26]. For example, a user can to enter data in a way that suits their expertise. This might be a database expert using the DSL SQL when entering data. The data can then be represented in various ways depending on the expertise of that user. A business user might prefer to visualise the data using excel-like spreadsheets.

At Chalmers a workbench software called Intentional Domain Workbench (IDW) [27] has been used to test new ways of handling variability [21]. What they did was to use the IDW as a way to model the variant and rules of the original demonstrator. However, to be able to use the IDW to model the variability for of upgraded demonstrator, some upgrades had to be done. This consisted of creating upgraded variability models that encompassed the new software variants, as well as integrating them in IDW.

This modeling was done using Clafer. When these rules are added to the IDW, the IDW can then use the Clafer tool to generate the truck instances. After this is done the instances are then imported back into the IDW, and can then be viewed according to the expertise of the user. This includes views of the separate instances as well as feature diagrams showing all variant combinations. It is also possible to show the rule models in different views. If needed, it also possible to integrate new ways of viewing the data. In figure B.2 the variant view in the workbench can be seen. As seen in this figure the workbench also contains meta data for the variants. This can be mnemonics, pictures or other information regarding the variants.

5.6 Conclusion

Upgrades were made to the demonstrator in three different areas. The hardware upgrades made it possible to handle a wider range of variants while the new variability model on the other hands show a new improved way of managing variability. Lastly, the new download and testing clients allows the demonstrator to handle the new software variants, which was made possible because of the previously mentioned upgrades. Because of these upgrades the demonstrator now has a wider range of variants as well as better way to handle them.

Although the the demonstrator does not perfectly represent the real assembly line, because of the improvement in these three areas, the demonstrator is close enough to the real system so that it can showcase what improvements could be made to the assembly line as well as what improvements could be made to the variability modeling.

6
Discussion

In this thesis, the modeling of truck variability has been investigated. The current practices regarding variability at Volvo were investigated as a part of this to see what parts of the variability modeling could be improved. With the help of state of the art modeling tools as well as the knowledge gained from the current Volvo systems a new way of handling the variability has been designed. This new modeling method was implemented on a demonstrator that was created at AB Volvo [4]. The demonstrator work as a small scale version of the truck assembly line. The benefits of modeling a small scale system are high as the costs, both in terms of time and resources, are much lower compared to if it would have been implemented on the real production line. For the system to simulate the behavior of the real system as good as possible some features had to be added. This included upgrading the model trucks with new software and hardware features as well as adding a software download station to the assembly line. The original purpose of the demonstrator was to educate Volvo employees on how the assembly process works. The demonstrator is still being used for this, thus these upgrades were created in such a way that the user experience still reflects the real factory practices. The demonstrator and the variability modeling is separate from each other which means a link between the two is needed. This link is responsible for taking the variability model and then generating the lists of variants that the demonstrator requires. A language workbench that had been developed at Chalmers [21] was used for this. The benefit of using this workbench is that it makes it is easy to manage the variability. This is because the output as well as the input can be specialized to the users expertise. For example, the workbench can not only output the list of variants needed by the demonstrator, but it can also present the models in different way, such as feature diagrams.

A benefit of using Clafer [17], which is the language used for the variability modeling, is that it contains a lot of useful modeling features. This makes it possible to use the simple modeling used in this thesis, but still being able to upgrade the modeling in the future. Another benefits of Clafer is that it also contains tools for generating instances of models, which is useful when the list of variants is created for the demonstrator. The benefits of the modeling used in this thesis compared to the one used at Volvo lies mostly in the design process. Because the designer is not restricted to constraint-based modeling it is possible to make models which are both easier to write as well as understand.

Because of limitations in time as well a resources certain parts did not get as much focus as others. For example, only one of the model trucks were upgraded with the new features which meant it was not possible have several station active with the new features at the same time. Another part that could have been further improved was the workbench, but because of limited access to the workbench software it was not possible to improve it further. Because of this there is still a number of improvement that could be made to the workbench. It already contains the most important part which is the variants information and the Clafer-model. One possibility would be to add meta-data, such as Computer-Aided Design (CAD) drawings for the physical variant as well as well as the sofware source code. Using small increments the platform could be used to handle more and more part of the manufacturing process. By adding the code to the the workbench it could also be used to generate the complete software packages for the ECUs. By having small packages which only contains the specific software needed for each unique ECU the software download process could be become signficantly faster.

Bibliography

[1] M. M. Tseng, J. Jiao, M. E. Merchant, Design for mass customization, CIRP Annals-Manufacturing Technology 45 (1) (1996) 153–156.

[2] J. R. Jiao, T. W. Simpson, Z. Siddique, Product family design and platform-based product development: a state-of-the-art review, Journal of Intelligent Manufacturing 18 (1) (2007) 5–29.

[3] Know4Car consortium, Know4car (2011).
URL `http://www.know4car.eu/`

[4] R. Oliveira, P. Johansson, Manufacturing execution system demonstrator platform, Master's thesis, Chalmers University of Technology (2011).

[5] MESA, The benefits of mes: A report from the field, 1997.
URL `https://services.mesa.org/ResourceLibrary/ShowResource/38dd4c5a-d0ee-4a5b-b0eb-1cf2a7c74636`

[6] A. Stork, N. Sevilmis, D. Weber, D. Gorecky, C. Stahl, M. Loskyll, F. Michel, Enabling virtual assembly training in and beyond the automotive industry, in: Virtual Systems and Multimedia (VSMM), 2012 18th International Conference on, IEEE, 2012, pp. 347–352.

[7] T. Mujber, T. Szecsi, M. Hashmi, Virtual reality applications in manufacturing process simulation, Journal of materials processing technology 155 (2004) 1834–1838.

[8] M. Ferrari, G. Ferrari, Building robots with lego mindstorms, Syngress, 2001.

[9] D. Sabin, R. Weigel, Product configuration frameworks-a survey, IEEE intelligent systems 13 (4) (1998) 42–49.

[10] J. L. Kolodner, An introduction to case-based reasoning, Artificial Intelligence Review 6 (1) (1992) 3–34.

[11] F. Hayes-Roth, Rule-based systems, Communications of the ACM 28 (9) (1985) 921–932.

[12] P. Lindroth, Product configuration from a mathematical optimization perspective, Chalmers University of Technology, 2011.

[13] F. Rossi, P. Van Beek, T. Walsh, Handbook of constraint programming, Elsevier, 2006.

[14] D. Batory, Feature models, grammars, and propositional formulas, Springer, 2005.

[15] Ø. Haugen, A. Wąsowski, K. Czarnecki, CVL: common variability language, in: Proceedings of the 16th International Software Product Line Conference-Volume 2, ACM, 2012, pp. 266–267.

[16] S. S. Iyengar, Metadata driven system for effecting extensible data interchange based on universal modeling language (uml), meta object facility (mof) and extensible markup language (xml) standards, uS Patent 6,874,146 (Mar. 29 2005).

[17] K. Bąk, K. Czarnecki, A. Wąsowski, Feature and meta-models in clafer: mixed, specialized, and coupled, in: Software Language Engineering, Springer, 2011, pp. 102–122.

[18] T. Bilgic, D. Rock, Product data management systems:..., in: Proceedings of the 1997 ASME Design Engineering Technical Conferences and Computers in Engineering Conference, paper No. DETC97/EIM-3720, Citeseer, 1997.

[19] Volvo Group Trucks Technology, Volvo Group PDM, (KOLA) Guidelines (2014).

[20] M. Fowler, Language workbenches: The killer-app for domain specific languages?, 2005, URL http://martinfowler. com/articles/languageWorkbench. html.

[21] A. H. Ebrahimi, P. E. Johansson, K. Bengtsson, K. Åkesson, Managing product and production variety–a language workbench approach, Procedia CIRP 17 (2014) 338–344.

[22] National Semiconductor, LM386 Low Voltage Audio Power Amplifier (8 2000).

[23] Arduino SA, Arduino due (2013).
URL `http://arduino.cc/en/Main/ArduinoBoardDue`

[24] Arduino SA, Arduino uno (2013).
URL `http://arduino.cc/en/Main/ArduinoBoardUno`

[25] Arduino SA, Arduino ethernet shield (2012).
URL `http://arduino.cc/en/Main/ArduinoEthernetShield`

[26] M. Fowler, Domain-specific languages, Pearson Education, 2010.

[27] Intentional Software, Intentional white paper, 2014.
URL http://www.intentsoft.com/wp-content/uploads/2014/05/ISC_WhitePaper.pdf

A

Flow Charts

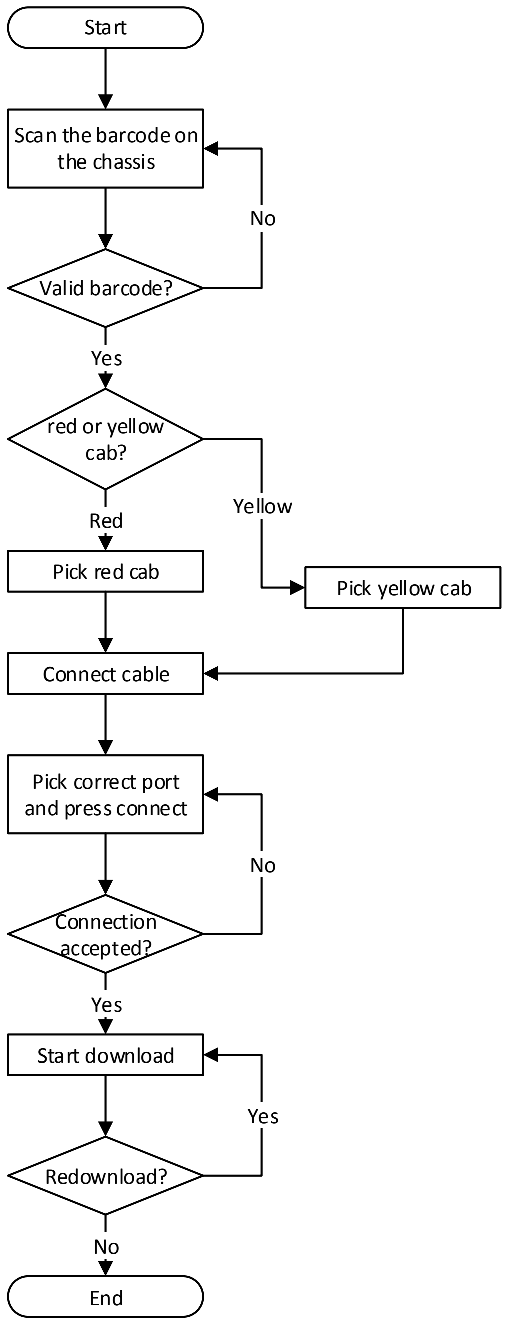


Figure A.1: Flow chart for the download client

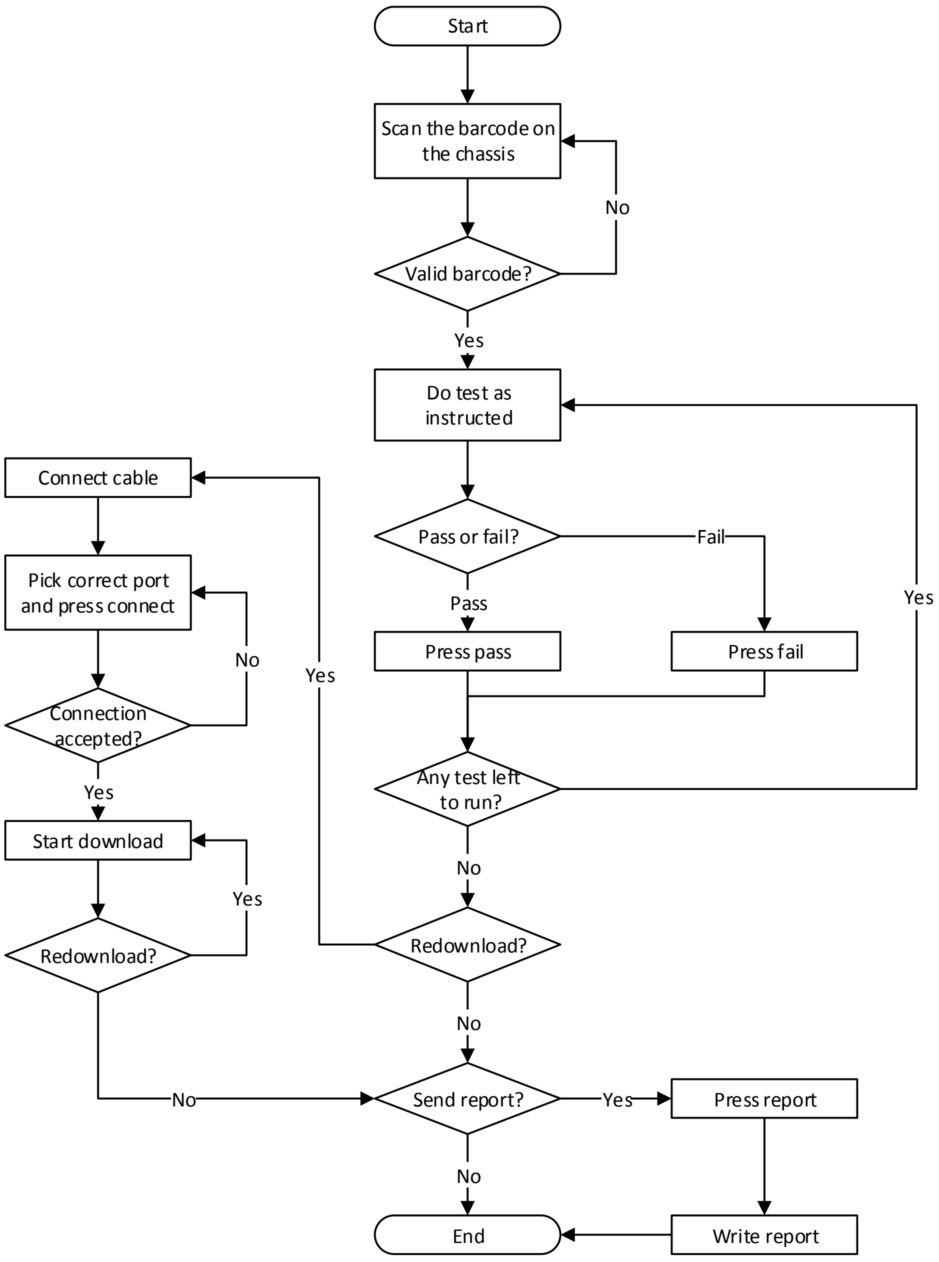
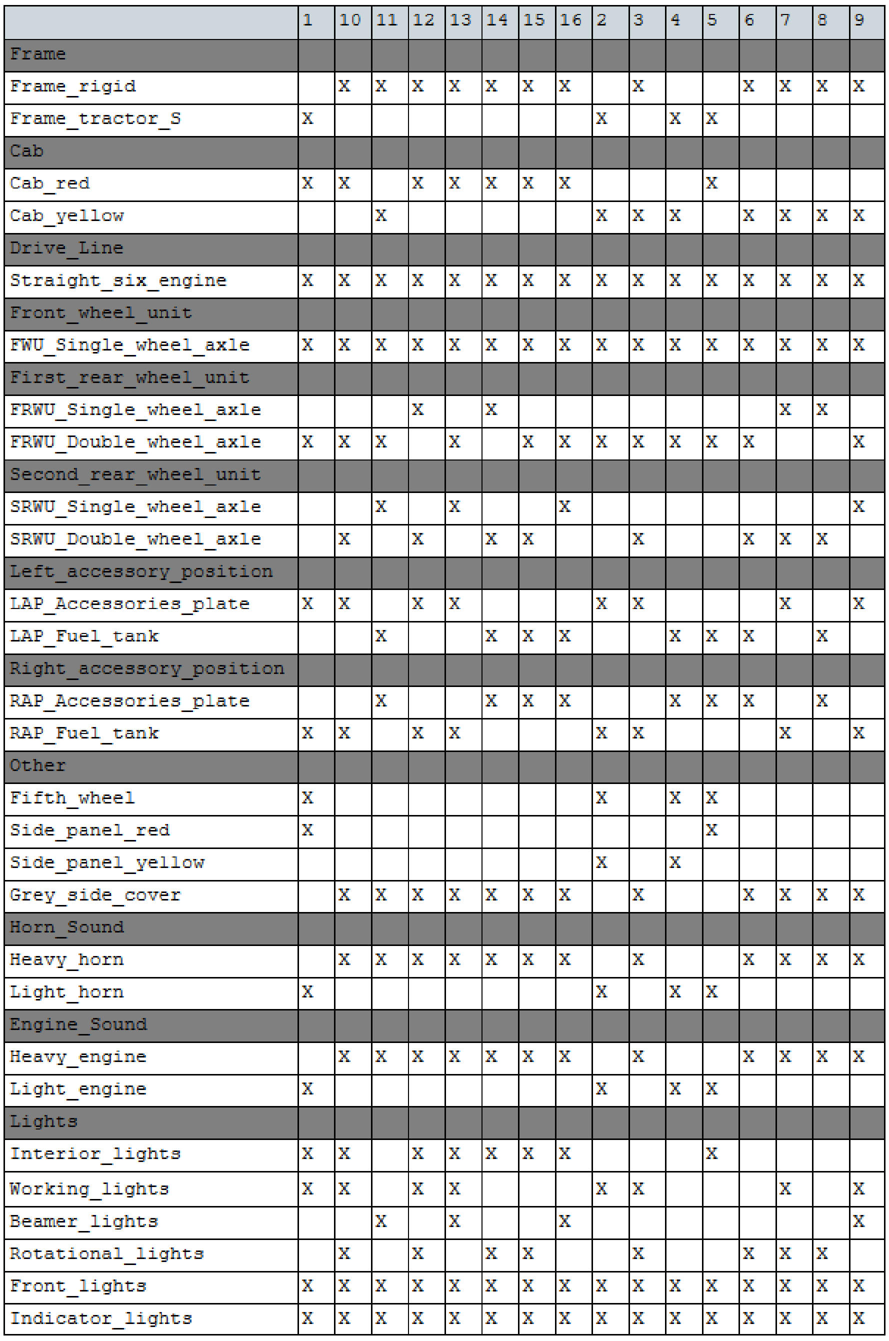


Figure A.2: Flow chart for the testing client

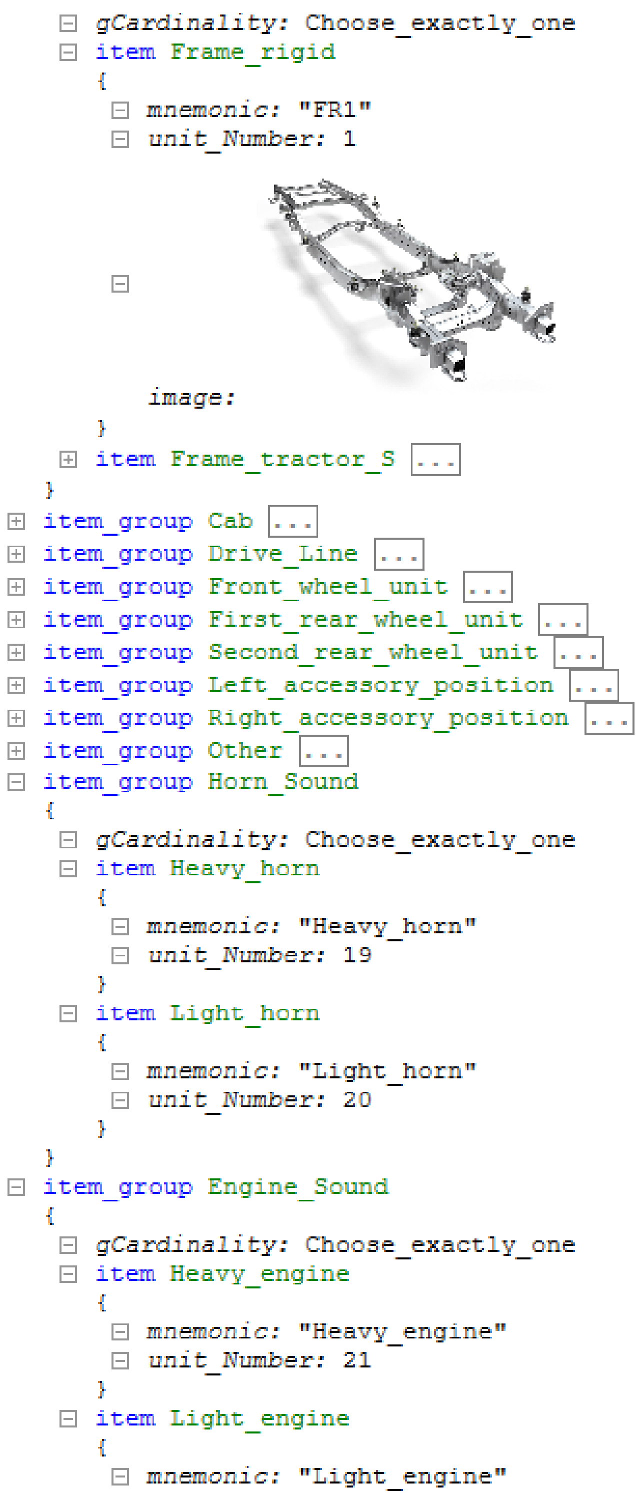
B

Workbench



	1	10	11	12	13	14	15	16	2	3	4	5	6	7	8	9
Frame																
Frame_rigid		X	X	X	X	X	X	X		X			X	X	X	X
Frame_tractor_S	X								X		X	X				
Cab																
Cab_red	X	X		X	X	X	X	X				X				
Cab_yellow			X						X	X	X		X	X	X	X
Drive_Line																
Straight_six_engine	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Front_wheel_unit																
FWU_Single_wheel_axle	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
First_rear_wheel_unit																
FRWU_Single_wheel_axle				X		X								X	X	
FRWU_Double_wheel_axle	X	X	X		X		X	X	X	X	X	X	X			X
Second_rear_wheel_unit																
SRWU_Single_wheel_axle			X		X			X								X
SRWU_Double_wheel_axle		X		X		X	X			X			X	X	X	
Left_accessory_position																
LAP_Accessories_plate	X	X		X	X				X	X				X		X
LAP_Fuel_tank			X			X	X	X			X	X	X		X	
Right_accessory_position																
RAP_Accessories_plate			X			X	X	X			X	X	X		X	
RAP_Fuel_tank	X	X		X	X				X	X				X		X
Other																
Fifth_wheel	X								X		X	X				
Side_panel_red	X											X				
Side_panel_yellow									X		X					
Grey_side_cover		X	X	X	X	X	X	X		X			X	X	X	X
Horn_Sound																
Heavy_horn		X	X	X	X	X	X	X		X			X	X	X	X
Light_horn	X								X		X	X				
Engine_Sound																
Heavy_engine		X	X	X	X	X	X	X		X			X	X	X	X
Light_engine	X								X		X	X				
Lights																
Interior_lights	X	X		X	X	X	X	X				X				
Working_lights	X	X		X	X				X	X				X		X
Beamer_lights			X		X			X								X
Rotational_lights		X		X		X	X			X			X	X	X	
Front_lights	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Indicator_lights	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure B.1: Generated feature diagram from the workbench



```
        ⊟ gCardinality: Choose_exactly_one
        ⊟ item Frame_rigid
          {
            ⊟ mnemonic: "FR1"
            ⊟ unit_Number: 1

            ⊟

              image:
          }
        ⊞ item Frame_tractor_S [...]
      }
    ⊞ item_group Cab [...]
    ⊞ item_group Drive_Line [...]
    ⊞ item_group Front_wheel_unit [...]
    ⊞ item_group First_rear_wheel_unit [...]
    ⊞ item_group Second_rear_wheel_unit [...]
    ⊞ item_group Left_accessory_position [...]
    ⊞ item_group Right_accessory_position [...]
    ⊞ item_group Other [...]
    ⊟ item_group Horn_Sound
      {
        ⊟ gCardinality: Choose_exactly_one
        ⊟ item Heavy_horn
          {
            ⊟ mnemonic: "Heavy_horn"
            ⊟ unit_Number: 19
          }
        ⊟ item Light_horn
          {
            ⊟ mnemonic: "Light_horn"
            ⊟ unit_Number: 20
          }
      }
    ⊟ item_group Engine_Sound
      {
        ⊟ gCardinality: Choose_exactly_one
        ⊟ item Heavy_engine
          {
            ⊟ mnemonic: "Heavy_engine"
            ⊟ unit_Number: 21
          }
        ⊟ item Light_engine
          {
            ⊟ mnemonic: "Light_engine"
```

Figure B.2: View of the variants in the workbench