



Minnesota State University, Mankato
Cornerstone: A Collection of Scholarly
and Creative Works for Minnesota
State University, Mankato

All Theses, Dissertations, and Other Capstone
Projects

Theses, Dissertations, and Other Capstone
Projects

2019

Managing Variability in Assembly Lines

Sumanth Gokapai
Minnesota State University, Mankato

Follow this and additional works at: <https://cornerstone.lib.mnsu.edu/etds>



Part of the [Manufacturing Commons](#), and the [Operations Research, Systems Engineering and Industrial Engineering Commons](#)

Recommended Citation

Gokapai, S. (2019). Managing variability in assembly lines [Master's thesis, Minnesota State University, Mankato]. Cornerstone: A Collection of Scholarly and Creative Works for Minnesota State University, Mankato. <https://cornerstone.lib.mnsu.edu/etds/917/>

This Thesis is brought to you for free and open access by the Theses, Dissertations, and Other Capstone Projects at Cornerstone: A Collection of Scholarly and Creative Works for Minnesota State University, Mankato. It has been accepted for inclusion in All Theses, Dissertations, and Other Capstone Projects by an authorized administrator of Cornerstone: A Collection of Scholarly and Creative Works for Minnesota State University, Mankato.

MANAGING VARIABILITY IN ASSEMBLY LINES

By
Sumanth Gokapai

A Thesis submitted in partial fulfillment of the
Requirements for the degree of
Master of Science
In
Manufacturing Engineering Technology

MINNESOTA STATE UNIVERSITY, MANKATO
MANKATO, MINNESOTA

May 2019

April 05th, 2019

Managing Variability In Assembly Lines

Sumanth Gokapai

This thesis has been examined and approved by the following members of the student's committee.

Dr. Kuldeep Agarwal, Associate Professor

Dr. Shaheen Ahmed, Assistant Professor

Dr. Winston Sealy, Associate Professor

ABSTRACT

Every successful company have a diversified product portfolio based on their customer demand. Though products are branded as different variants, versions etc. most of them are similar products with little variations in their features or functions they perform. With the traditional, product centric approach to handle variation, new techniques like Clone and own were implemented in which engineers pick the most similar one available, copy it, make changes to it and present it as a new variant. Even though this approach employs reuse, but the savings occur once and only once.

This project discussed how companies can exploit the commonality between the products, manage variability and get benefitted using concepts like feature based modelling, materialization etc. Feature based modelling was extensively being used in software industry, this paper focuses on using the same methods in manufacturing industry and improve the product lines for better efficiency. This project specifically investigates the assembly lines and how they can be programmed and documented to handle the portfolio of products efficiently through the entire Product Life Cycle using pure::variants, a leading variant-management tool. The use feature based modelling and variability management techniques through the assembly lines greatly reduces the evolution and development time of variants, improves cycle time and with project management documents like instruction Manuals, Bill of materials etc.

Keywords: Central Variability Model (CVM), Feature based modelling, Pure Variants, Assembly Lines, Product Life Cycle.

ACKNOWLEDGMENTS

Firstly, I would like to express my deepest gratitude to my advisor Dr. Kuldeep Agarwal, for his patience, support, valuable guidance, and encouragement throughout the entire journey of my thesis completion. I whole heartedly thank him for providing me with all required guidance and support to pursue my internship abroad. I would like to thank the members of my committee, Dr. Shaheen Ahmed and Dr. Winston Sealy for their valuable discussions and accessibility.

Secondly, I would like to thank my supervisor Dr. Bart Meyers and manager Davy Maes at Flanders Make, Belgium for guiding and encouraged me to complete the thesis. Lastly, I am eternally grateful to my family and friends for their love and support.

Table of Contents

Introduction	1
Literature Review	3
Background	6
Variability	6
Assembly Lines	7
Operator Instruction Manuals	10
Methodology	11
Software Product Line Engineering	11
Variability Management	13
Tool Chain	18
Variability Management Tools	18
Binding Time	34
New Toolchain - Static Binding	35
Case Study – Duck Maker	37
Raspberry Pi	39
Brick Pi	42
Tool Chain - Static Binding	47
Tool Chain - Dynamic Binding	51
Conclusions and Further work	55
References	56

List of Figures

<i>Figure 1 - Tool chain of existing assembly lines</i>	<i>8</i>
<i>Figure 2 - Overview of SPLE</i>	<i>11</i>
<i>Figure 3 - Overview of variability management</i>	<i>13</i>
<i>Figure 4 - Example of feature model</i>	<i>15</i>
<i>Figure 5 - Proposed toolchain for assembly lines</i>	<i>18</i>
<i>Figure 6 - Variation types and description</i>	<i>19</i>
<i>Figure 7 - Feature model and Variant description model in pure::variants</i>	<i>20</i>
<i>Figure 8 - Ribbon Tab of pure::variants plugin in MS Word.....</i>	<i>23</i>
<i>Figure 9 - Modelling master instructional manual</i>	<i>25</i>
<i>Figure 10 - Activity diagram.....</i>	<i>28</i>
<i>Figure 11 - State machine diagram.....</i>	<i>29</i>
<i>Figure 12 - Creating feature model.....</i>	<i>Error! Bookmark not defined.</i>
<i>Figure 13 - Model transformation instructions</i>	<i>32</i>
<i>Figure 14 - LL Embedded Engineer installation - 1</i>	<i>33</i>
<i>Figure 15 - LL Embedded Engineer installation - 2</i>	<i>33</i>
<i>Figure 16 - LL Embedded Engineer - 3.....</i>	<i>34</i>
<i>Figure 17- Toolchain (Static Binding).....</i>	<i>35</i>
<i>Figure 18 - Implementation of SPLE techniques.....</i>	<i>38</i>
<i>Figure 19 - Duck maker assembly line</i>	<i>39</i>
<i>Figure 20 - Brick Pi case assembly</i>	<i>43</i>
<i>Figure 21 - Brick Pi powering guide.....</i>	<i>44</i>

<i>Figure 22 - Toolchain for Duck Maker (Static Binding)</i>	47
<i>Figure 23 - Commonality identification</i>	48
<i>Figure 24 - Variability modelling</i>	48
<i>Figure 25 - Product family State machine diagram</i>	49
<i>Figure 26 - Product variant State machine diagram</i>	50
<i>Figure 27 - Toolchain for Duck Maker (Dynamic Binding)</i>	51
<i>Figure 28 – Transformation of MS Word document</i>	53
<i>Figure 29 - Toolchain of DuckMaker</i>	53

Introduction

Manufacturing companies have different ways to meet the market demand, some companies deliver end products from their inventory to the customers. Other companies, instead of keeping the finished goods inventories, they assemble or manufacture end products only after they receive the customer order. However, producing finished goods to order does not necessarily mean that the manufactured item is tailored to a specific customer's requirements. For instance, if the end products inventory costs are high, suppliers choose to manufacture only when the order was placed.

Now a days, companies have more need to look towards customization more than ever, because of customers unique set of demands in their orders. In order to satisfy customer needs, companies have to supply products which address the wants of the consumers which made companies to diversify their product portfolio. While diversifying the portfolio, companies are making different products in order to address the customization problem at hand. Though products are branded as different variants, versions etc. most of them are similar products with little variations in their features or functions they perform.

While assembling these products, most companies use automated assembly lines but because of the variations within the portfolio, the assembly lines has to be programmed repeatedly and consumes a lot of time. Today, the development and validation of these controller software variants require a lot of manual effort. This results in long market introduction cycles for each

new variant and therefore leads to some reluctance to bring new products to the market, leaving business opportunities unused.

Therefore, in this research study, we proposed a simpler method to considerably shorten the time needed for the development and evolution of new variants using techniques like feature based modelling, materialization and pure::variants software.

Literature Review

As a result of rise in individualization of customer products in numerous industries during recent times, much of the effort has been focused to increase the flexibility and versatility of assembly lines by planning and developing decision models for the mixed model assembly lines [Boysen, Fliedner & Scholl, 2009][1]. Shtub & Dar- El, 1989[2] developed a methodology to select an assembly system by ranking possible alternatives in regards to subjective cost or benefit criteria and, determined for assembly of low volume and high diversifiable products, high degree of specialization of labor and its associated effects can be used to get most benefited.

Kull Hans, 2015[3], in his book Mass Customization Opportunities, Methods, and Challenges for Manufacturers, describes mass customization as automated manufacturing of bespoke products. Joneja, Ajay & Lee, 1998[4] describes mass customization as a trend that is being followed undoubtedly by many industries. Tseng, 1996[5] proposed a product design methodology called Design for mass customization for efficient manufacturing of product variants. In order to achieve this, Tseng suggested to maximize the modularity in product design. Suh, 1990[6] proposed an axiomatic approach and introduced product family structure as design basis to define similar products.

Schmid & Santana, 2013[7] describes Product Line Engineering (PLE) as one of the few industry ready techniques to manage reuse & variability, definably and hence bring software development to further developed stage. The goal is to deliver exact product variants with quick response times at an economical life-cycle cost with high quality. PLE is implemented both in large companies like Philips, Lucent, Boeing and Toshiba, and many small companies. PLE consists of two life cycles namely Domain engineering which creates customizable software and provides

required assets to individual projects and application engineering uses the software provided as basis for developing required product variant.

Automotive manufacturers are under pressure to deliver a wide range of product variants in order to satisfy customers demands and managing these product variants has become a major concern in automotive industry [Tarek & Hoda, 2010][8]. Flores, Krueger & Clements, 2013[9] defined Second Generation PLE (2GPLE) to handle the product complexity, richness of variation in their product family at General Motors. Using 2GPLE, GM has successfully managed their difficulty to deal with complex and numerous product variants. Not only Automotive industry, many other industries have been benefited by imparting PLE into their product lines specifically defense in their satellite ground control systems[10], weapon test ranges[11], helicopter avionics systems[12], submarine combat systems[13] etc.

Semi- Automated assembly lines are equipped with human operators and human factor proves to be crucial in production systems due to their cognitive abilities, versatility and flexibility demonstrated in facing unplanned events [Mura, Dini & Failli, 2016][14]. Many other factors known as Performance Shaping Factors (PSFs) [15], can effect and cause errors in operator's performance, e.g., operator factors[16] such as worker's memory, mental and physical abilities, skills, training level, experience etc. Assembly line errors caused by human operators must be decreased in order to reduce the cycle time which results in high productivity and efficiency.

In summary, there are many techniques to handle variability among the product variants and this project discusses another reasonable method to handle and manage variability along with techniques for aiding operators by better managing instruction manuals.

Background

Variability

Every company has to offer different types of products in order to satisfy wide range of customer's needs. Lately, customers' demands are increasing and because of this, industries are shifting towards customer-focused production allowing customers to specify product components and features they offer. There are many problems associated with this transition because production management systems are usually designed based on a limited number of product variants [17]. Bachman & Clements, 2005 [18] defines variability as a ability of a system, an asset, or a development environment to aid the production of a group of artifacts that vary from each other in a predetermined fashion.

In assembly lines, depending upon the complexity of the product, the number of parts needed to be assembled increases which increases the complexity of assembly procedure. An automotive system has huge number of components and allows customers to specify options such as motor type, motor size, body color, interior and many add-ons. The number of different product variants will be more in this type of scenarios and becomes very difficult to manage. The traditional approach is to maintain a BOM (Bill of Materials) for each variant and this approach will only work, if there are less number of product variants. A car in general has 20,000 to 30,000 parts [19] and this makes impossible to different product variants using traditional approach.

Assembly Lines

Assembly line is one of the greatest invention in the 20th century. It was so beneficial and efficient that the companies refuse to adapt this invention soon became extinct. Later Henry Ford improved the assembly line concept and introduced the moving platforms of a conveyor system in which the chassis of the car is moved from one station to another allowing the workers to assemble the parts and finishing the assembly. GM installed its own robotic arm that could move and assemble parts in a continuous pattern, repeated work is automated by the robots and the operators are eliminated which resulted in better productivity and efficiency. Today robotics are reaching a new level of advancement and companies are developing robots to adapt manufacturing process and making them to work next humans. Along with robots, assembly lines are equipped with Visual Aids, to assist the operators in complex assemblies to eliminate unnecessary errors.

There are different types of assembly lines and they can be broadly classified into three categories, namely:

- Manual assembly lines
- Automated assembly lines
- Semi-automated assembly lines

Manual assembly lines

An assembly line that consists of a sequence of workstations where the assembly tasks are performed by human workers. Workers start with base parts and adds components along the line to progressively build the product. Sometimes, workers uses portable powered tools to do

specific operations and production rate is relatively slow and error prone compared to automated and semi-automated assembly lines.

Automated assembly lines

An automated assembly line consists of series of workstations performing predetermined operations linked by a transfer system to move from one workstation to another controlled by electrical control system. A fully automated assembly line doesn't require human operators directly involved to make an assembly, every process is taken care by machines and automated systems. In automated assembly lines, human operators are used for supervision, system design and monitoring the process rather than operating.

Semi-automated assembly lines

In semi-automated assembly lines, both human workers and automated systems work together in assembling the product. Human workers compliment automated systems by doing certain operations which are in capable of automated systems. The picture below shows the basic tool chain of semi-automated assembly line.

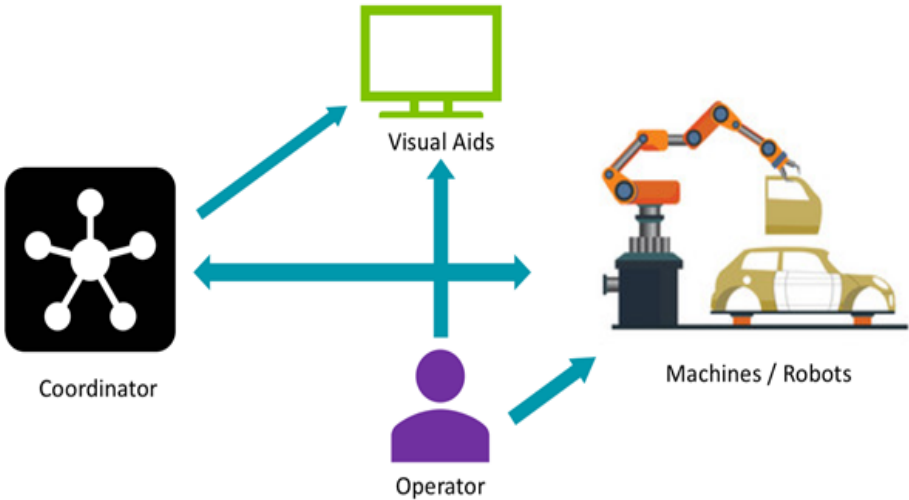


Figure 1 - Tool chain of existing assembly lines

- **Coordinator:** Coordinator serves as brain of the Assembly line, instructs every other member in the assembly line on what to do and when to do.
- **Machines / Robots:** These are the ones which does the maximum work and follows the instructions of the Coordinator. Robots does not have idea of the process but follows the instructions of Coordinator with in its capabilities.
- **Operator:** Robots are usually restricted by their capabilities and if a complex step occurs in the assembly process which is out of Robots capabilities, then the operator performs that specific task and enables the assembly process.
- **Visual Aids:** These can be as simple as pdf documents, Display unit or Virtual Reality depending upon the complexity of the procedure and company's budget. These are used to aid the operator in the assembly process in various tasks.

Semi-automated assemblies are handling the assembly process efficiently without any fail and provide high productivity rates according to the customer orders. But in some cases, companies has to respond quickly to the varying customer demands especially when they are orders with high variety and low volume. In this scenario, the coordinator has to be programmed specifically for the order again due to the variety and requires a lot of manual effort. This results in long market introduction cycles for each new variant and therefore leads to some reluctance to bring new products to the market, leaving business opportunities unused.

Operator Instruction Manuals

Instructions for the workers in assembly line during assembly process plays a vital role and has a significant impact on the production rates and had to be given high importance. Because of numerous product variants, it is impossible for the operators to remember the instructions and even maintain countless BOM and documents required for the assembly. Since, a better method to manage these instruction manuals is required to aid the operators during assembly process for better production rates.

Methodology

The objective of this project is to offer software product line methods, techniques and tools for the development of mechatronic software controller variants considerably shorten the time for the development and evolution time of new variants. This framework will strongly decrease the manual efforts that are needed to implement controller software for existing product variants.

This will be achieved by selecting, linking and configuring the appropriate feature-related components.

Software Product Line Engineering

Software Product Line Engineering (SPLE) focuses on developing software-intensive systems using platforms and mass customization [20-21]. Software product lines (SPLs) are the most successful methods to increase productivity, lower costs and shorten time-to-market in software engineering businesses [22]. SPLs facilitate organizations to reuse of software by developing a family of products that share some commonality, but differ in several aspects.

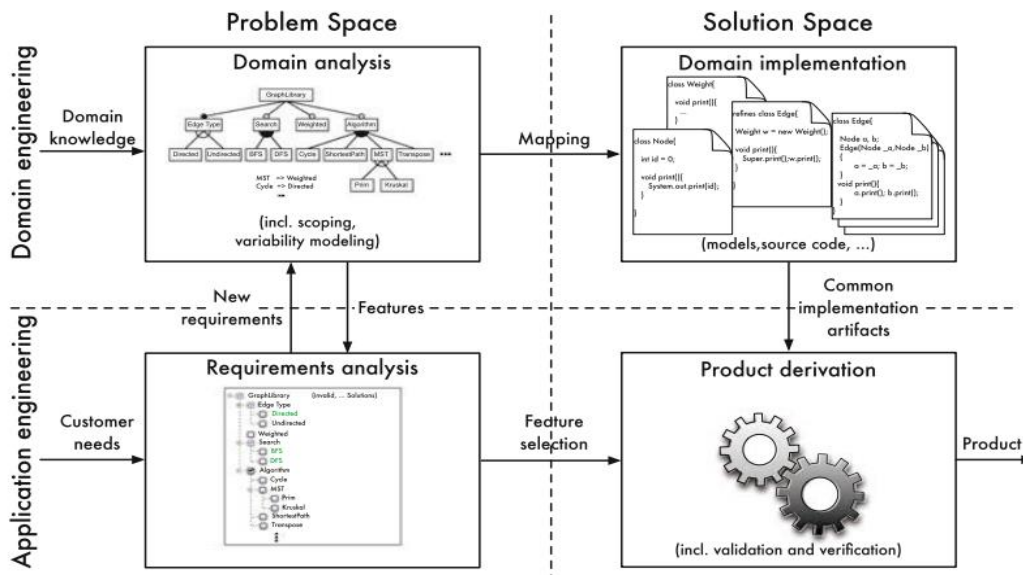


Figure 2 - Overview of SPLE

Domain Engineering

Domain Engineering [23] focuses on the development for reuse, where one develops artifacts for reuse and these doesn't result in a specific product. These artifacts can be used in developing specific products.

- **Domain analysis:** Where we define all our features, variability and constraints which define all family of products including scoping and variability modelling.
- **Domain implementation:** In this module, one develops all the domain specific source code, tools, models which are used to produce or used for all family of products rather than a specific product.

Application Engineering

This focuses on the developing specific product according to the needs of a particular customer.

This is connected to the Domain Engineering space and the specific product requirements are derived from the artifacts developed for reuse.

- **Requirement analysis:** This module offers the customers with options for selecting required features according to their needs and requirements.
- **Product Derivation:** Specific product according to the requirements of the customer is derived including validation and verification of the constraints associated with requirement analysis.

As an overview, Domain analysis and Requirement analysis can be constituted as Problem Space, which focuses on perspective of the stakeholder's problems, requirements and gives an idea on the entire family of products with features and capabilities they offer. While Domain implementation and Product Derivation can be constituted as Solution space, which focuses on

design, implementation, validation and verification of features and their combinations so that systematic reuse is availed [23].

Variability Management

Variability management is a key activity in software product-line engineering [24]. There has been a lot of studies going on since 1990s [25]. There are a lot of approaches have been introduced to the industry from last two decades. Variability management involves specific activities to handle and manage variability. Some of the key activities are Commonality Identification, Variability Modelling, Materialization, and Variability Evolution.

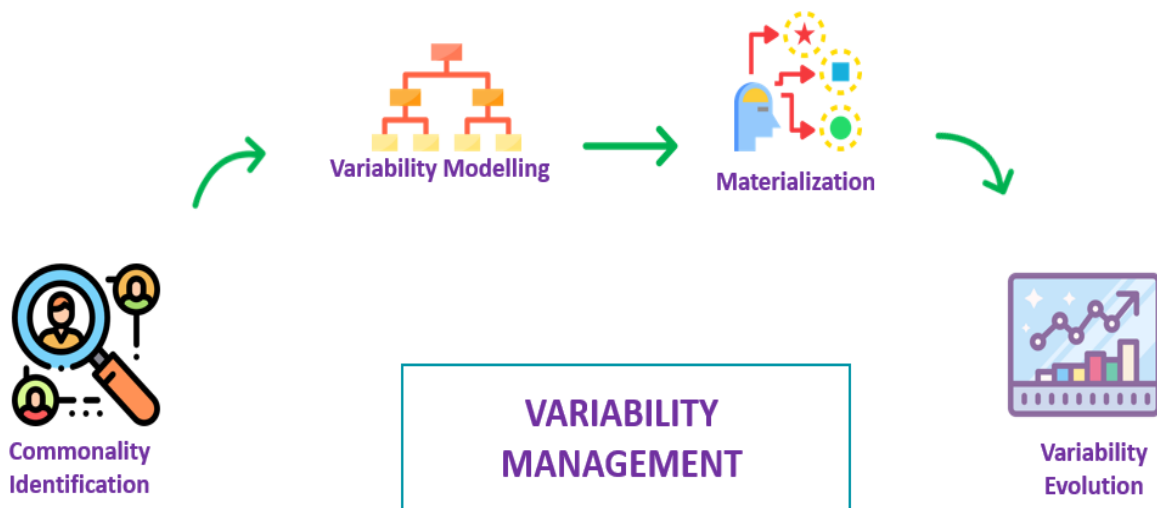


Figure 3 - Overview of variability management

Commonality Identification

SPLE exploits the commonalities of the products that belong to a product family and handles the variation among the products. Commonality is a feature that is shared by all products of the product line [26]. The main aim of this activity is to identify all the commonalities within product family.

Variability Modelling

Variability is expressed by feature models, variability expressed in relation a base model [27][28]. Commonality and variability of a product family can be modelled in many ways based on different viewpoints [29]. In problem space, user goals, objectives, constraints are modelled in product line engineering. There are two modelling techniques which are popular in the industry. They are Feature Modelling and Decision Modelling. Feature Modelling is the one which is extensively used in software product lines compared to Decision Modelling.

Feature Modelling

Feature Modelling was first introduced in the Feature-Oriented Domain Analysis (FODA) by Kang et al. (1990), since then it has been widely used in software reuse methodologies along with Software Product Line Engineering (SPL) as means for modelling variability in product line i.e., a family of products. Feature modelling is an analysis technique which is used to define product line i.e. family of products [30]. Feature is defined as a distinctive quality or characteristic of a software system or system. An example of a Feature Model is shown below:

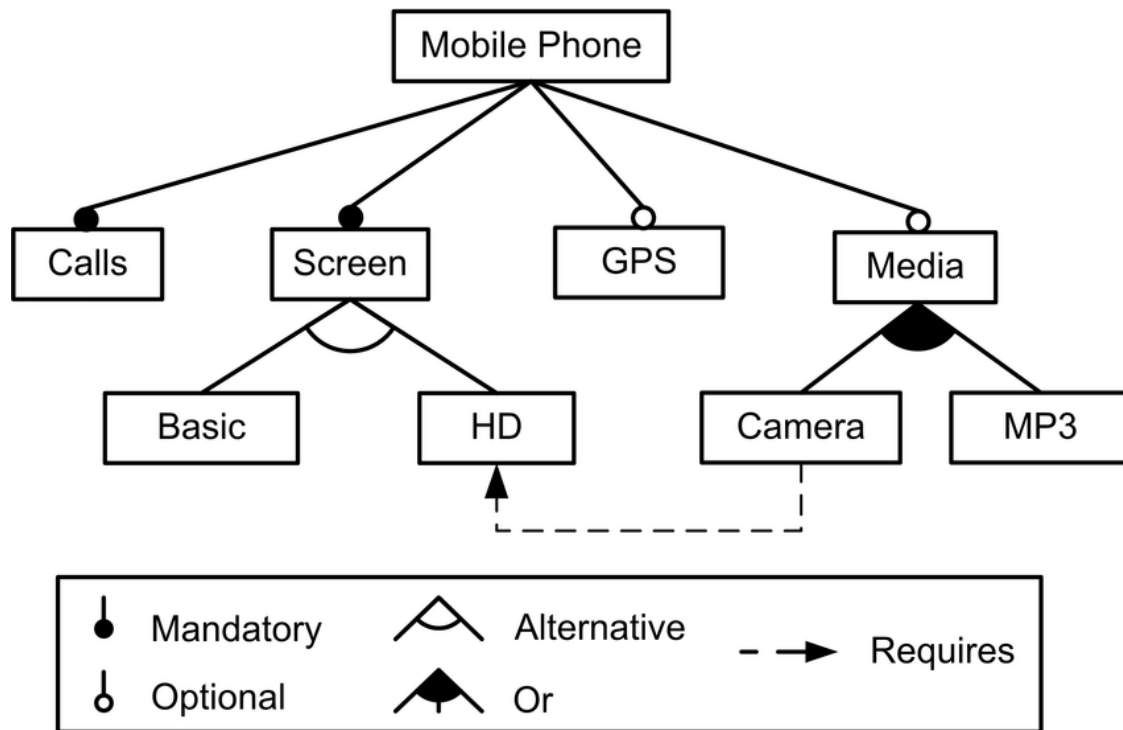


Figure 4 - Example of feature model

The relationship among features can be divided into [31]:

- **Mandatory.** If a feature is defined with a mandatory relationship with its parent feature, it should be included in all the combinations in which its parent feature appears. In the above Figure, all mobile phones should support calls.
- **Optional.** If a feature is defined with a optional relationship with its parent feature, it may or may not be included in all the combinations including its parent feature. For example, GPS is defined as an optional feature, so it may or may not be a feature of mobile phones.
- **Alternative.** A set of child features are defined with alternative relationship with their parent feature where only one of them can be selected when its parent feature is part of the combination. In Figure, mobile phones might provide support for Basic or HD (High Definition) screen, but not both of them at the same time.

- **Or.** A set of child features are defined as an or-relationship with their parent when one or more than one of them can be included in the combinations in which its parent feature is included. In Figure, software for mobile phones can provide support for Camera, MP3 or both in the same combination.

In addition to the hierarchical relationships between features, a feature model can also contain constraints which makes the model robust and helps to define the model completely. These are usually of the form:

- **Requires.** If a feature X requires a feature Y, if feature X is included in a combination implies that feature Y should be included in such combination. In Figure 1, mobile phones including the feature Camera must include support for a HD screen.
- **Excludes.** If a feature X excludes a feature Y, both features cannot appear in the same combination.

Product Feature Model (PFM)

PFM describes the variability of family of products in terms of product features, which can be classified into parts, capabilities, domain technologies, operating environments and implementation techniques as defined in FORM [32]. The most common usage of feature diagram is to represent a Product Feature Model [33].

Materialization / Product Derivation

Product derivation [34] in software product line engineering is the process of constructing a product based on the user selection of features defined. Feature selection is determining the optimal choices that satisfy product goals and quality requirements [33]. Based on the selection, only selected features are derived and the information is carried out for production or next steps.

The product derivation is generally automated process depending the tools (Enterprise Architect, Pure Variants etc.,) one uses.

Variability Evolution

Product lines are long-lived software systems supporting companies with changing customer requirements [\[35\]](#). Variability models evolve and grow together with the evolution and growth of the product line itself [\[36\]](#). Whenever a new feature is added to the product line or a new product is added to the family of products, instead of recreating the feature model with new arrangements, feature models are constructed to support evolution of variability. New features are added to the existing feature model which significantly reduces the development time for new variants.

The whole variability management process is automated and aids the cycle times of product lines to be faster. Different software are used to help the variability management to be managed easily and intuitively.

Tool Chain

Variability Management tools like Pure Variants, Magic Draw (for modelling), Lieber Lieber Embedded Engineer (for code generation) are used to manage the variability and make the assembly line capable of producing different products on a single assembly line.

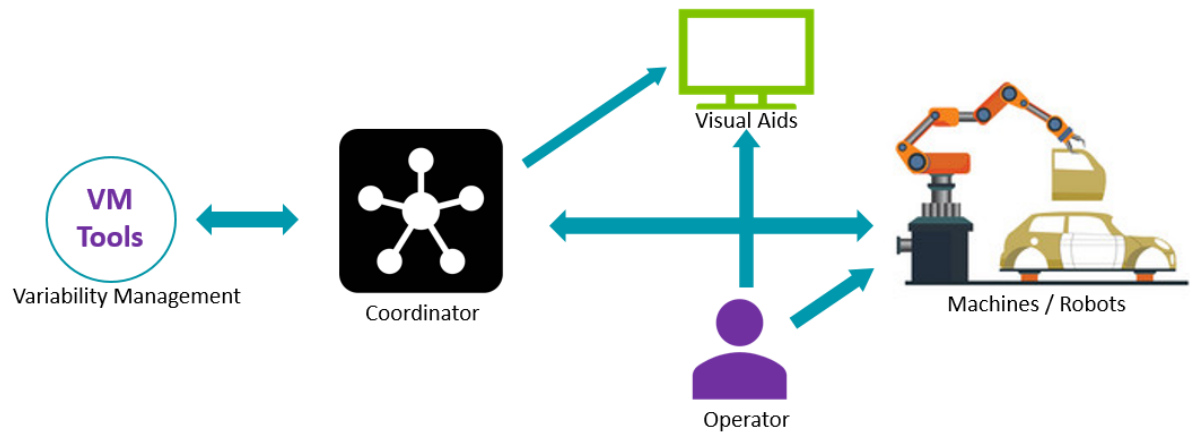


Figure 5 - Proposed toolchain for assembly lines

Variability Management Tools

Pure Variants

The pure::variants [37] is a commercial variants and variability management suite. The main goal of pure::variants is to provide a systematic way to express and define variability and variant information throughout the life cycle of a product line. It is designed to complement the existing development tools and provide the required links between the tools in order to aid efficient development of variant-rich systems.

The pure::variants tool uses a uniform meta-model to define variability [38] in the problem space (feature models) and solution space (family models) and a related meta-model to describe

variant combinations (variant description models). In a typical use case portfolio in pure::variants, product managers define the planned and intended product commonalities, features and variabilities in feature models. Out of the defined features in feature model, one selects the required for a particular product and assign the features of products in variant models. Different variant models for specific variant of a product and there can be as many as variant model for a single feature model. Each variant model has a single product's configuration.

Feature Models

Feature model are used to express commonalities and variabilities efficiently. A feature models explains features and their relations. A feature is a property with respect to the commonalities or variation between the products in the product family. The relations define the connections between features whether they are optional, alternative or mandatory.





Short name	Variation Type	Description	Icon
mandatory	<i>ps:mandatory</i>	A mandatory element is implicitly selected if its parent element is selected.	
optional	<i>ps:optional</i>	Optional elements are selected independently.	
alternative	<i>ps:alternative</i>	Alternative elements are organized in groups. Exactly one element has to be selected from a group if the parent element is selected (although this can be changed using range expressions). pure::variants allows only one <i>ps:alternative</i> group for the same parent element.	
or	<i>ps:or</i>	Or elements are organized in groups. At least one element has to be selected from a group if the parent element is selected (although this can be changed using range expressions). pure::variants allows only one <i>ps:or</i> group for the same parent element.	

Figure 6 - Variation types and description

With respect to the features and relations, a feature model is constructed which defines product family by expressing the commonalities and variability.

Variant Description Model (VDM)

Variant description model allows the users to select the intended features of the desired product from the feature model.

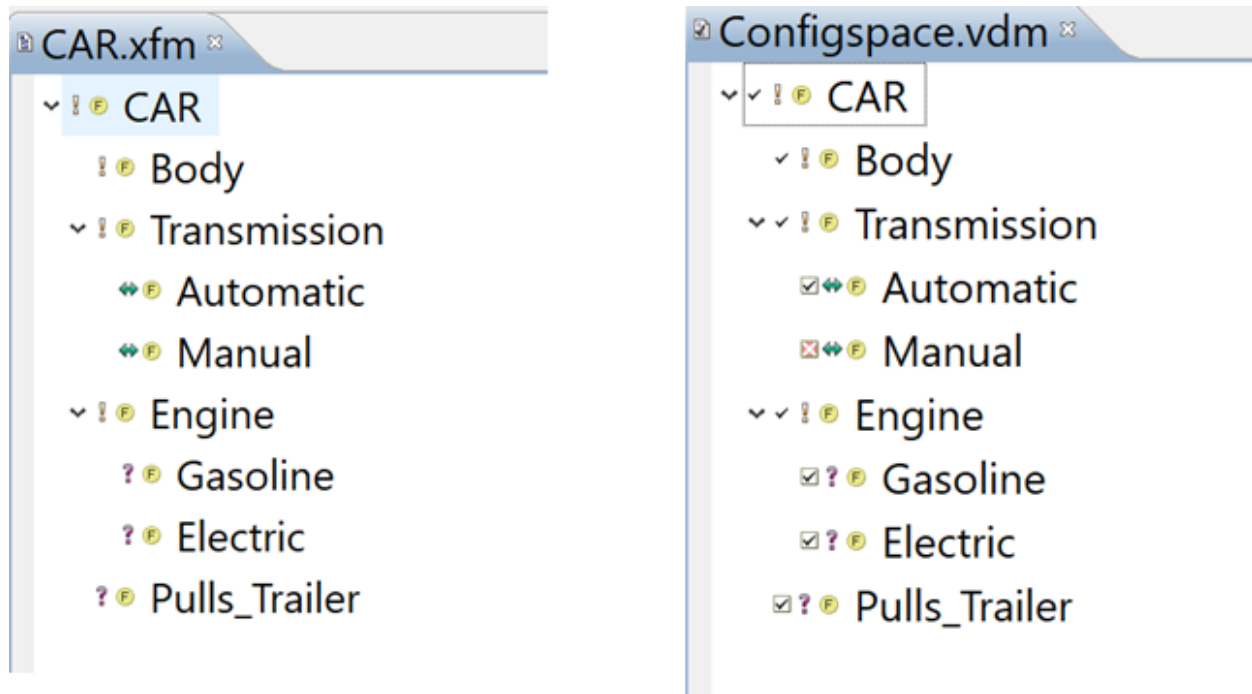


Figure 7 - Feature model and Variant description model in pure::variants

The figure above shows both feature model (.xfm) and variant description model (.vdm) of a product CAR. The feature model CAR.xfm has a product CAR with mandatory features Body, Transmission, Engine and an optional feature Pulls_Trailer. Also Transmission has child attributes Automatic and Manual as alternatives so that user has an option to select only one option either Automatic or Manual. Mandatory feature Engine also has two child attributes defined as an optional feature, Gasoline and Electric providing user with option to select either one of them or both.

Transformation Modules

There are many transformation modules available in pure variants, which helps the variant management system to be efficient and complete, can be found in [\[39\]](#). With respect to this project, there are two transformation modules which helped the tool chain to be complete and aid the variant management process to be automated. They are:

- JavaScript Transformation Module
- Microsoft Transformation Module

JavaScript Transformation Module

The pure::variants offers JavaScript transformation module for generating a product variant.

There are no special requirements required to use the transformation module and one use it right away. The main goal of the JavaScript transformation module is to access the features in the feature model. If a user selects his desired features in the variant description model, this module helps to map those selections and helps to create a product variant. Following are the instructions to create a JavaScript transformation module:

- Open the transformation configuration page in the Configuration Space properties.
- Add the JavaScript Transformation module using the Add button. Name it for instance Execute JavaScript.
- The module parameters can be changed on next page.
- Enter the path to the script file you want to execute as value of the JavaScript file parameter.
- An (optional) output file can be specified using the output file parameter.
- Press Finish to finish setup of the JavaScript transformation

The script consists of three main functions. These three functions will be called by the transformation module.

- **init()** This Method is optional. Necessary work can be done here, before transformation starts, like initializing the script. Gets necessary information from transformation module, like the used variant model, the used models in this variant, some variables and the transformation parameters. All this information can also be retrieved from the JavaScript transformation module using getter functions.
- **work()** Does the whole transformation work.
- **done()** This method is optional. After transformation is finished, this function is called, to provide possibility to do some work after transformation.

Microsoft Word Transformation Module

The pure::variants Connector for Microsoft Office enables the use of product line variability concepts in Microsoft Office Word and Excel documents [\[40\]](#). It allows to maintain one master document from which different document variants are created automatically by selecting features from Feature Models in pure::variants. So instead of having to merge changes in slight variations of the base documents, the change is applied once to the master document and then all relevant variants are automatically generated by pure::variants.

The pure::variants connector can be installed in the Microsoft Word as an plugin, which makes it easy for the users to create a single master document and define the content according to the feature model.

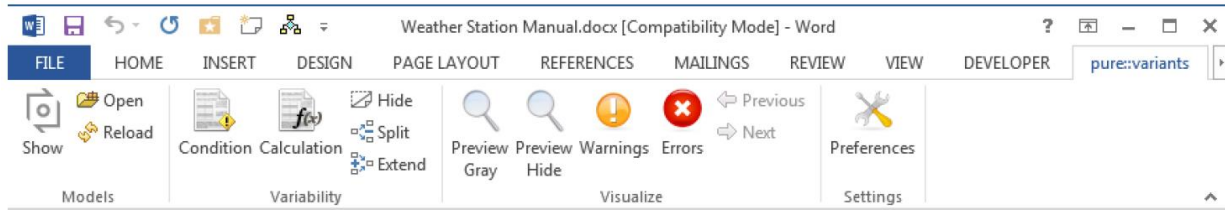


Figure 8 - Ribbon Tab of pure::variants plugin in MS Word

The figure above represents the pure::variants Ribbon Tab with user interface elements after installing the pure::variants plug in Microsoft word application. The user interface elements function are explained below:

- Show: Shows or hides the pure::variants task pane.
- Open: Opens a configuration space or other pure::variants models (.xfm, or .ccfm). See the section called “Using the pure::variants Task pane” for details.
- Reload: Reloads all loaded pure::variants models and refreshes the visualization.
- Condition: Adds a condition to the current text selection.
- Calculation: Adds a calculation to the current text selection.
- Hide: Hides all variability information.
- Split: Splits the commented area of an existing condition or calculation at the current text cursor location. The text cursor must be inside exactly one commented area.
- Extend: Extends a commented area of an existing condition or calculation with the current selection. If the current selection overlaps with exactly one commented area, the overlapped area is extended with the selection
- Preview Grey: Preview visualization, which grays out all elements that would not be included in a variant produced with the currently loaded variant model

- Preview Hide: Preview visualization, which hides all elements that would not be included in a variant produced with the currently loaded variant model
- Warnings: Visualization that highlights all conditions and calculations that contain semantic errors in the pvSCL expression, such as unknown names of features or attributes
- Errors: Visualization that highlights all conditions and calculations that contain syntactic errors in the pvSCL expression
- Previous: Jump to previous faulty condition or calculation.
- Next: Jump to next faulty condition or calculation.
- Preferences: Opens the preferences dialog

Editing Variability

Users can create a master document like normal document in Microsoft Word application. The plugin integration provides an editor, which features auto completion, syntax highlighting and checking for errors. There are two ways one can define variability in this transformation module either by conditions or calculations [41].

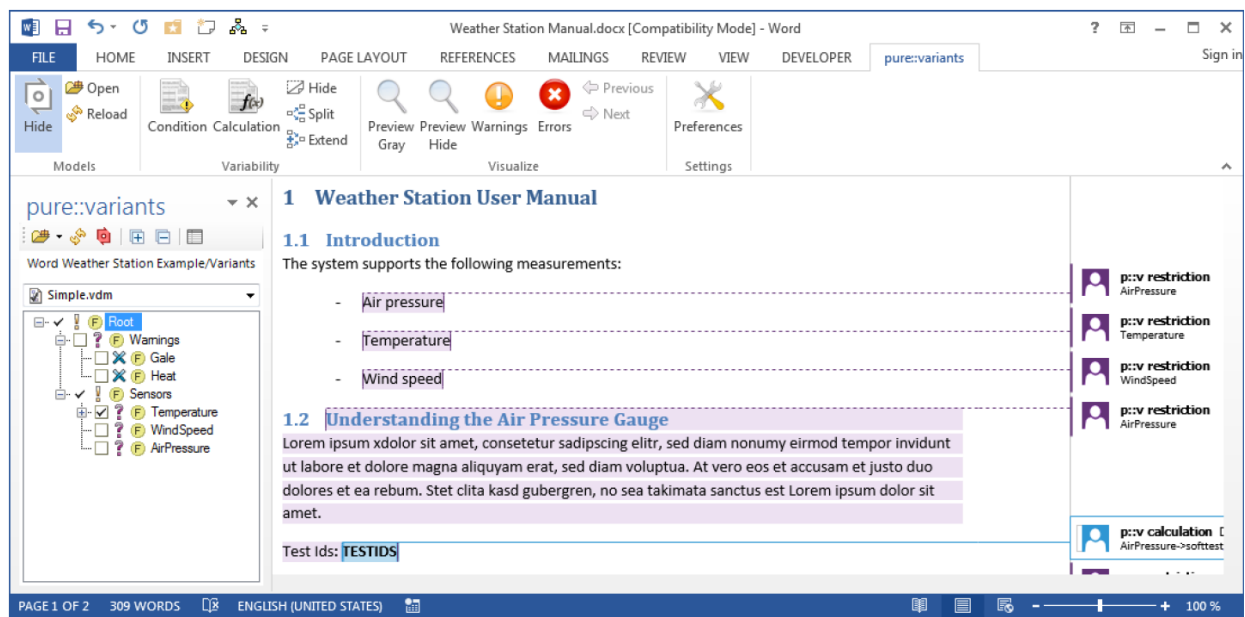


Figure 9 - Modelling master instructional manual

- **Conditions:** Conditions are the pvSCL expressions that return with a Boolean value, either true or false, which decides whether the annotated text should be in the resulting variant or not. To annotate the text and assign a condition, select the desired text and click **Condition** button. An editor dialog box opens, enter the condition within the box and it also has features like auto-completion and syntax highlighting.
- **Calculations:** Calculations are the pvSCL expressions that return a value, which replaces the entered text during transformation.

Magic Draw

Magic Draw is the award-winning business process, architecture, software and system modeling tool with teamwork support. Designed for Business Analysts, Software Analysts, Programmers, QA Engineers, and Documentation Writers, this dynamic and versatile development tool facilitates analysis and design of Object Oriented (OO) systems and databases. It provides the industry's best code engineering mechanism (with full round-trip support for Java, C++, C#, CL (MSIL) and CORBA IDL programming languages), as well as database schema modeling, DDL generation and reverse engineering facilities[\[42\]](#).

The Unified Modelling Language (UML) is widely used in the development of software development and also for its customizations, for computer based software, business process modelling and system design [\[43\]](#). Instead of using UML as a documentation tool, we tried to use it for defining the system and generate code from it using Magic draw (defining model) and Lieber Lieber Embedded Engineer (code generation). The following are the steps to create a model with activity diagrams and state machine diagrams.

Creating a model according to the use requirements from scratch

To create a model from scratch according to the requirements, one has to do the mandatory steps in order to lessen the hassle during code generation.

- Create a new project
- Create a new package in the project
- Now open up a existing and working project(Guessing_Game)

- Copy the following folders under the package of Guessing_Game
EE_PrimitiveTypes_Package
Usings
EA Profile
Embedded Engineer
- Go to the Package and also copy the folder FSM and Class signals.
- Check all the inner elements of FSM and FSM_State and their data types. (Note :Having correct data types for all the declarations helps us eliminate a bunch of errors during Code Generation times)
- Now start with a class and build your own model.
- When defining the operations and Opaque Behaviors inside the class make sure that both operation and Opaque behavior are connected to each other by directing them in Specification(Opaque Behavior) or Method (Operation)
- It is advisable to create both Operations and Opaque behaviors by going into specification of certain Class

Example for creating a new Operation

- Right Click on the class where you want to create and go to Specification
- On the left hand side there'll be a containment tree, choose Operations
- In order to create a new operation, click on create on the lower right hand side
- Give all the required details and press OK

Similarly go to inner elements in order to create an Opaque behavior.

- If we create the Operations and Opaque Behavior in the containment tree itself sometimes it might not create the code and give errors like, the function was not defined. So it is advisable to create the Operations and Opaque Behaviors as described in Step above.

Creating an Activity Diagram

To create an activity diagram [44] for certain activity user can go to diagrams in the menu bar and create an activity diagram or Right Click on the class under which the activity is going to be and go to diagrams and then activity diagram.

- Start with giving the activity diagram a name
- Draw the diagram according to the requirements s enclosed with Start and Finish nodes
- Use Opaque actions to do a certain action and write the appropriate code in Body or Body and Language section.
- One can also call the operations and Opaque Behaviors from the Opaque Action in activity diagram
- To distribute the flow from one action to two or more actions use Fork Horizontal rather than drawing multiple control flow lines from the same activity.
- Drawing multiple control flow lines from one action gives errors (like multiple exits) during the code generation process

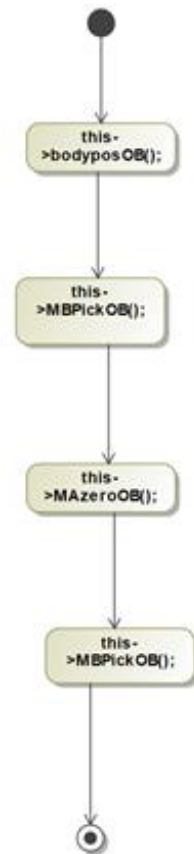


Figure 10 - Activity diagram

- While calling the functions/methods/operations in the activity diagram make sure that the name of the method was called correctly and use this keyword. Ex (this->printfunction());

Creating a State Machine Diagram

State diagram helps the user to control the total project flow and also helps to integrate all the methods and diagrams created before.

- Start with naming the diagram as “State Machine”, since the name was defined similarly in the FSM and FSM_State Classes (Even in the main function of Guessing_Game)

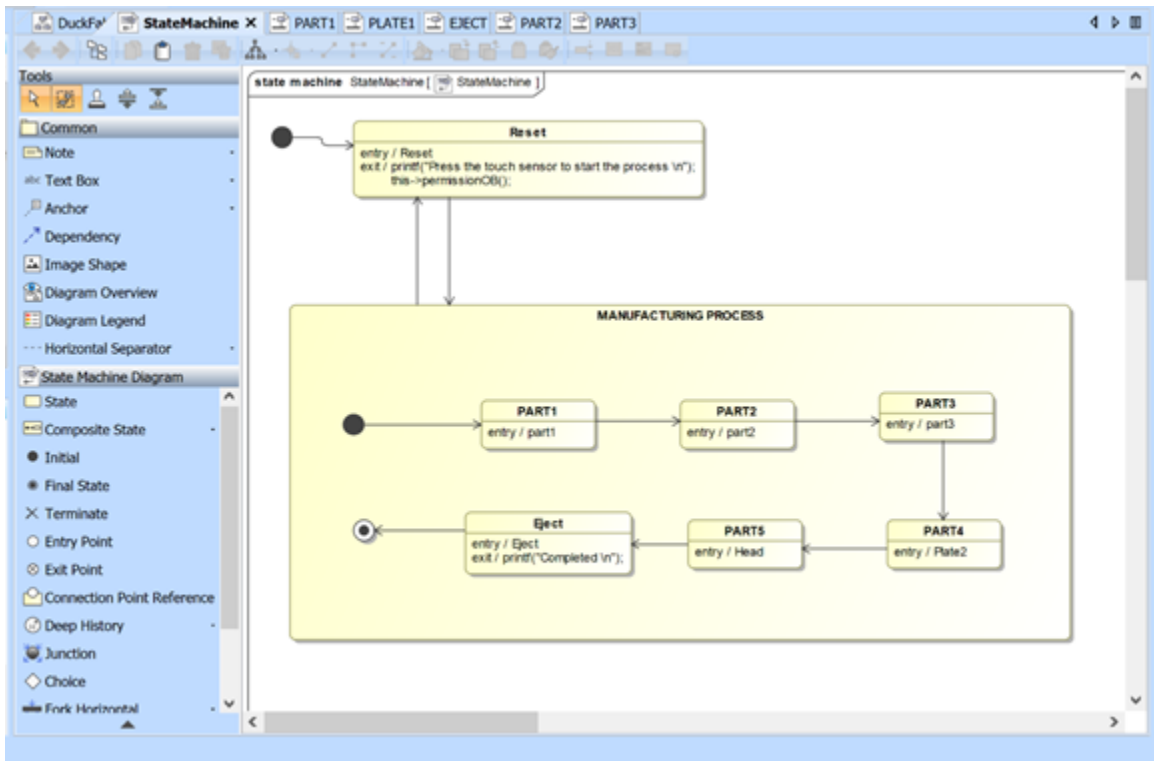


Figure 11 - State machine diagram

- Create different states according to the requirements and to call the Opaque Behaviors defined before. Go to Specification of the state -> Select Behavior type from Entry/ Do/ Exit and select Opaque Behavior and call the methods from Body and Language.

- Attaching an activity diagram to the state is a bit tricky. Go to specifications of the state. Select Behavior type as Activity. Now double click on the Behavior element which takes to the new screen. Go to node and add the “Call Behavior action” and now select the behavior add direct to the activity diagram you want to link.

Integrating Magic Draw with Pure Variants Enterprise

Integration

To integrate the existing Pure Variants Enterprise project with Magic Draw project, follow these instructions:

- After creating the model in Magic Draw, to integrate the pure variants to add variability, Go to File -> Use Project -> Local Project -> Select Pure variants Profile.mdzip and click OK

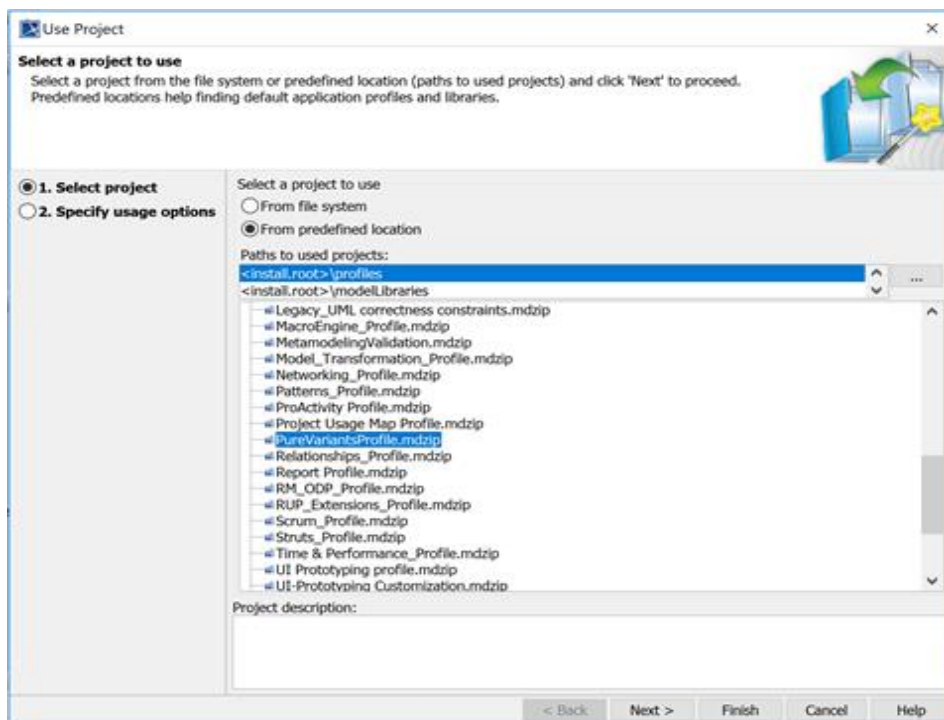


Figure 12 - Integrating Magic Draw with pure::variants

- To check that you have done this process correctly, there'll be a pure variants window opened either on the right side or at left hand bottom corner or you can right click on any state or activity and find Pure Variants option where you can select the type of variability.
- Go to the pure variants window and open the existing feature model (.xfm) from the project you want to integrate.
- After opening the feature model, add variability according to the requirements to activity diagrams, flows, states etc.
- Now select the variant model (.vdm) from the project.
- After selecting the variant model from the project you can observe the effect of the added variation points to the model.
- Go to Pure Variants window in Magic Draw and click on the “Enable Transformation Preview” on Task bar and you can observe that all the variation points’ changes color to red or yellow according the variation points defined.
- Enable Transformation preview gives an overview how the model will look after Model Transformation.
- Check all the diagrams and change the variation points if necessary and once everything is fine and we are ready to transform the model.

Model Transformation

After completing defining all the variation points required and linking it to the Pure Variants model, we are ready to transform the current model as the variant model as per requirements.

To transform the model Go to Tools in Magic Draw -> Model Transformation, a Model Transformation Wizard will open and there are four steps:

1. Select transformation type -> Select Variant Realization -> Next
2. Select source/ destination -> Select your Package -> Next
3. Check mappings -> Do nothing -> Next
4. Specify transformation details -> check Clean Variation Point Data to true (if necessary) -> Finish.

This will transform the current model (150% model) into required (variant model) and we can generate the code for this model now.

Note: Don't get confused with two Model Transformation options in the Tools menu. Please select the first available Model Transformation.

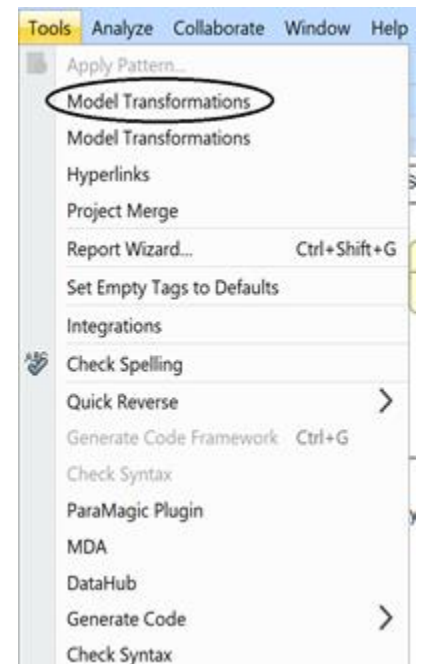


Figure 13 - Model transformation instructions

Lieber Lieber Embedded Engineer

LL Embedded Engineer [\[45\]](#) is a plugin which can be integrated into Magic Draw for generating code for the model drawn in Magic Draw.

Compatibility

In order to generate the code in C or C++, one can use Magic Draw 18.2 or any 18 version but if you are planning to use the Magic Draw 19.0 for generating C++ code, the code will be

generated but the generated code is in C language rather than C++.So, please pay attention while using the language and the Magic Draw version while generating the code.

Installation

Now that we have decided about the version of Magic Draw to use, we are going to look on “How to install the LL Embedded Engineer” plugin in the Magic Draw. The procedure of installation of the plugin is similar in either of the Magic Draw versions.

Please follow the following steps to complete the installation:

- Go to Help and find the Resource/ Plugin Manager:

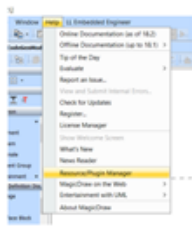


Figure 14 - LL Embedded Engineer installation - 1

- It opens up a new window with all the available plugins and now scroll down to the LL Embedded Engineer in the list.



Figure 15 - LL Embedded Engineer installation - 2

- Select the plugin and press Download/Install to install the plugin.

Note: It'll ask for admin permission to do so.

- After installing the plugin, you have to see the LL Embedded Engineer option on your Ribbon Tabs as shown below in the figure.



Figure 16 - LL Embedded Engineer - 3

- If you see LL Embedded Engineer tab, you have installed the plugin successfully.

Generating code

After Model Transformation, we can generate the code by right clicking on the Package in the containment tree -> LL Embedded Engineer -> Generate Code.

NOTE: It is always very important to save the project after transforming the model because if we haven't saved the project, the changes will not be reflected. So, it's mandatory to save the project every time after Transforming the model to generate the code for the intended 100% model.

Binding Time

Binding time is the time at which an implementation decision is made to create a binding [\[46\]](#).

Binding is an association between a name and the thing that is named. There are two types of binding namely Static Binding and Dynamic Binding.

- **Static Binding:** A binding is static, if binding occurs before run time and remains unchanged throughout the execution
- **Dynamic Binding:** A binding is dynamic, if binding occurs during run time and can change during execution

New Toolchain - Static Binding

A new tool chain is proposed to handle the variability, where the implementation occurs before the run time. The tool chain can be defined as below:

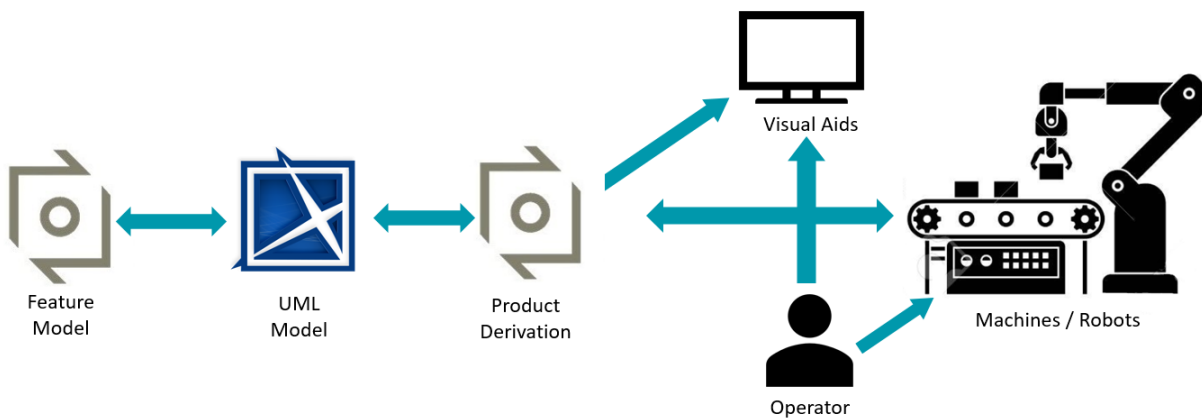


Figure 17- Toolchain (Static Binding)

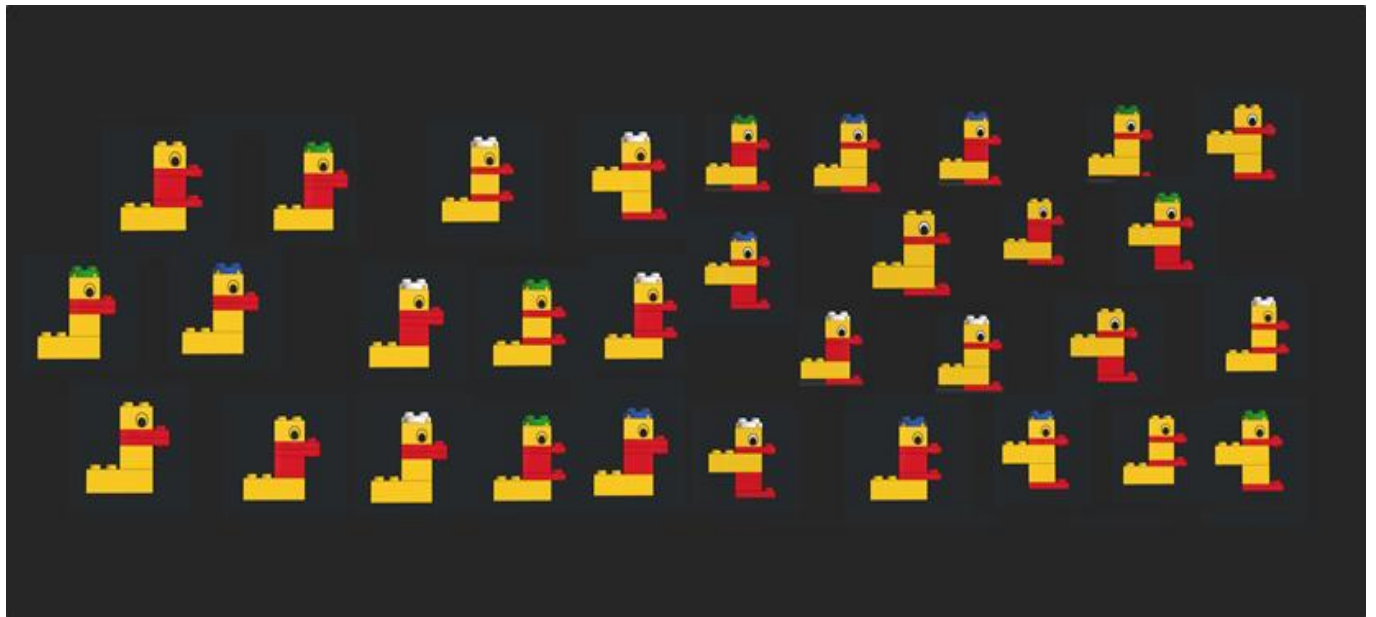
- Initially, companies define their product family with all their features, commonalities and variabilities.
- Once the variability modelling was done and now UML Model will be constructed according to the feature model in Magic Draw software to perform the desired actions on Machines/ Robots.
- After the UML Model, one can validate the model inside Magic Draw which helps to detect the problems with the UML Model.

- Now pure::variants feature model can be imported into Magic Draw software and integrate the UML Model with feature model and add all necessary constraints. Once this is done, now the tool chain is ready to take orders from customers/ consumers.
- Customers can open the Magic Draw software and select desired features within the feature model.
- After selecting the required features from the feature model, customers can preview the UML Model.
- This preview gives an idea on how the UML Model is transformed from family of products to required product variant and this gives a chance to change their selection, if required.
- After feature selection is finalized, transformation of model can be done by clicking Transform Model, this transforms the UML Model to required product variant along with MS Word transformation of operator instructions.
- After the UML Model is transformed, executable code can be generated by Lieber Lieber Embedded Engineer plugin with the MagicDraw software.
- This executable code is loaded on to the machines/robots to perform the required actions.

The proposed new tool chain can manage variability to reduce the development times of new variants.

Case Study – Duck Maker

A case study in assembly lines is done for testing out the proposed new tool chain. Brick Pi is chosen for constructing an assembly line and also for the products. Brick Pi has been widely used in research, education and teaching projects for its versatility and affordability (e.g., [47] [48]). In Discrete Event Systems (DES) and Automated Manufacturing Systems (AMS), BRICK PI has been used for research purposes (e.g., [49] [50] [51]). After trying out different combinations with the available BRICK PI bricks, we came up with 32 different styles of ducks as product variants.



The concepts of SPLE were used in developing the process for handling the variability. The figure below explains how these techniques will be implemented in order to successfully manage the variability:

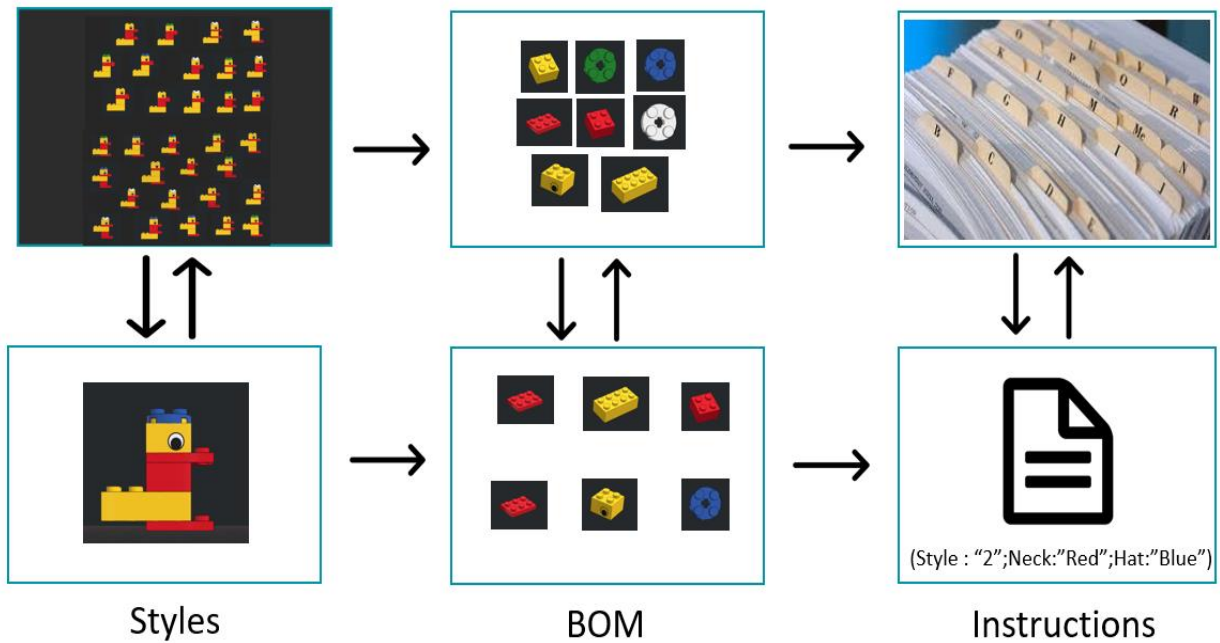


Figure 18 - Implementation of SPL techniques

Out of the all the available products, user selects desired features on product variant and automatically the bill of materials is transformed and the necessary instructions will be generated and communicated to assemble the required product variant.

Every duck needs five or six Brick Pi bricks to be assembled. Brick Pi bricks of different size, color and orientation are used. In order to assemble these Brick Pi ducks, an assembly line is made up of BRICK PI bricks which is controlled by microcontroller Raspberry Pi and motor board Brick Pi. After going through a lot of online Brick Pi forums and tutorials, an assembly line was made up of entirely BRICK PI bricks from BRICK PI Educational Set.

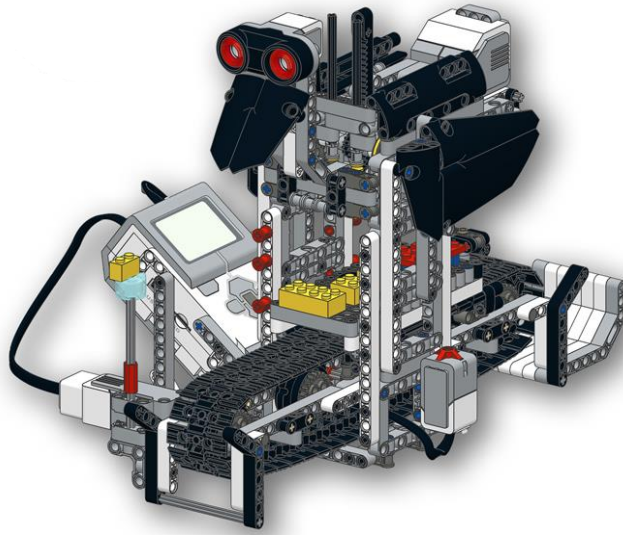


Figure 19 - Duck maker assembly line

Raspberry Pi

Single board computers became popular due to their affordability and capabilities they offer. Raspberry Pi as one of the first popular single board computers has gained popularity in the scientific community because of its flexibility, compactness, affordability and also with a huge support from community of creators with a lot of online resources. There are numerous cited scientific articles on using Raspberry Pi in solving engineering challenges and many of these applications required to be low cost, low mass and offer software flexibility [52-57]. This project uses a Raspberry Pi kit to control the assembly line made out of BRICK PI bricks.

Overview

Raspberry Pi kit comes with the following components:

- Raspberry Pi 3 Model B+ Board
- 16GB Micro SD Card – NOOBS (OS)

- Official Raspberry Pi Case
- Official Raspberry Pi Univ 2.5 A Power Supply

In order to start working with Raspberry Pi, we may need the following to start with:

- Raspberry Pi 3 Model B+ Board
- Keyboard
- Mouse
- HDMI Cable
- Official Raspberry Pi Univ 2.5 A Power Supply
- Monitor or TV
- 16GB Micro SD Card – NOOBS (OS)

Once we have all the required Hardware ready. Connect all the devices to Raspberry Pi board and if everything setup we can see the Home Screen of Raspbian OS. NOOBS is an easy operating system installation manager for the Raspberry Pi which consists of following operating systems:

- Raspbian
- LibreELEC
- OSMC
- Recalbox
- Lakka
- RISC OS
- Screenly OSE
- Windows 10 IoT Core

- TLXOS

Only Raspbian is installed by default in NOOBS. The others can be installed with a network connection.

Raspbian Os

Raspbian is an official operating system for Raspberry Pi devices, supported by Raspberry Pi Foundation. There are three most popular versions of Raspbian, widely spread among users now: Wheezy, Jessie and Stretch. All of them are based on Debian: Debian 7 Wheezy, Debian 8 Jessie and Debian 9 Stretch respectively. The Stretch is the latest one. It contains a lot of useful stuff: Chromium browser, Sonic Pi, RealVNC, NodeRED, Blue and Greenfoot Java IDE, Geany, Python, Scratch, Wolfram

Connecting to Wi-Fi:

Since we are using a Raspberry Pi3, there is built in Wi-Fi but if using an earlier version of the Raspberry Pi, then we'll need to connect a USB Wi-Fi dongle to one of the four USB ports in Raspberry Pi.

Troubleshooting Wi-F

Sometimes, Wi-Fi symbol will not appear as expected. This mostly happens when we first connect the Raspberry Pi board, in that case:

Click on the Raspberry Pi button on the top left of the screen -> Preferences -> Raspberry Pi configurations -> Localization tab -> Set Wi-Fi Country.

Following these steps would rectify this problem.

Running C++

If the users want to write code in C++, they can download applications like Code Blocks from the terminal itself.

Go to terminal and run the following code:

```
sudo apt-get install codeblocks
```

If the user is connected to internet, the codeblocks is downloaded to the Raspberry Pi.

We can also run C++ code on terminal but first we have to download the g++ compiler, to do that write the following code in the terminal:

```
sudo apt-get install g++
```

Once the compiler gets downloaded we can create a new C++ file and proceed.

Brick Pi

Brick Pi helps you do more with BRICK PI® MINDSTORMS® by connecting BRICK PI sensors and motors to the Raspberry Pi.

We can connect the Brick Pi to Raspberry Pi by mating the Brick Pi's header to the Raspberry Pi's GPIO header and we can attach the servo motors and sensors available to perform desired functions.

Note: The original Brick Pi and the Brick Pi+ are not supported on Raspbian Stretch. You will need to use an older version of Raspbian, such as Jessie or Wheezy

But if you are using Brick Pi3 and Raspberry Pi 3 its fine and they are compatible

Setting up the SD card

There are three ways in which we can set our SD Card so that it can be compatible with Brick Pi.

This information is available at [\[58\]](#)

Since we are using the Raspberry Pi 3 Model B+, we already have the required Raspbian OS and all we need to do is “Install the Brick Pi Repository”.

```
sudo curl -kL dexterindustries.com/update_Brick Pi3 | bash
```

This following command installs the Brick Pi Repository and allows us to use the Brick Pi

3.After installing, it needs a reboot. Enter the following command to complete the installation process: *sudo reboot*

Assembling the Case

Brick Pi comes with an acrylic case which holds both Raspberry Pi and the Brick Pi together and slots made within the acrylic allows the user to connect the sensors, motors and the power to the Brick Pi [\[58\]](#).



Figure 20 - Brick Pi case assembly

Powering the Brick Pi

There are 3 ways to powering the Brick Pi and each way has its own Pros and Cons.

Select the appropriate method to powering the Brick Pi from the options below[58]:



Figure 21 - Brick Pi powering guide

Connect to the Brick Pi

We can connect the Brick Pi to your PC (Windows), Mac or you can control without an computer.

If you want to connect the Brick Pi to your PC (Windows) or Mac, the detailed procedure on how to connect the Brick Pi is available at [\[59\]](#).

If you want to control the Brick Pi without a computer, all you need is a monitor, HDMI Cable to connect to you monitor, keyboard and a mouse.

Connect all the accessories to the Raspberry Pi board in the allotted slots and turn on the power.

As soon as we done this correctly, the Raspberry Pi starts booting up and we can see the Start screen of Rasbian OS.

Testing for the connection

In order to make sure that the Brick Pi is connected to the Raspberry Pi, run the following python code from the command window and if everything is right then it prints the following:

“Brick Pi3 connected and running”

The python code for testing whether the Brick Pi3 is connected to Raspberry Pi or not is given below and can be found at [\[59\]](#):

```
from __future__ import print_function
from __future__ import division
import time # import the time library for the sleep function
import Brick Pi3 # import the Brick Pi3 drivers
try:
    BP = Brick Pi3.Brick Pi3() # Create an instance of the Brick Pi3 class. BP will be the
Brick Pi3 object.
    print("Brick Pi3 connected and running")
except Brick Pi3.FirmwareVersionError as error:
    print(error)
except:
    print("Communication with Brick Pi 3 unsuccessful")
```

In order to make sure that the sufficient voltage is supplied to the Brick Pi board, we can monitor that using the following python program and can be found at [\[60\]](#):

```

from __future__ import print_function
from __future__ import division
import time
import Brick Pi3
BP = Brick Pi3.Brick Pi3()
try:
    while True:
        print("Battery voltage: %6.3f 9v voltage: %6.3f 5v voltage: %6.3f 3.3v voltage: %6.3f"
              % (BP.get_voltage_battery(), BP.get_voltage_9v(), BP.get_voltage_5v(), BP.get_voltage_3v3()))

        time.sleep(0.02)
except KeyboardInterrupt:
    BP.reset_all()

```

This program should show the voltages reading at different positions of the Brick Pi board.

Sometimes, the motor is connected to the Brick Pi board and able to communicate its encoder readings but not able to move. This is due to insufficient power supply from the batteries. So, in order to make sure that the sufficient power is supplied, Battery voltage from the above program should read voltages between 9V-12V.

The power status of Brick Pi can be explained by the nature of LED's on it. The following explanation regarding LED's on Brick Pi board helps us understanding the powering the Brick Pi:

- The green LED indicates that the Brick Pi3 is powered.
- The yellow LED is normally controlled by the Brick Pi3 firmware, but can be manually controlled by the user program. The user program can set the LED brightness to a level of 0-100%, or to -1 to return control of the LED back to the firmware. When the firmware is controlling the LED, it flashes it according to the status of the Brick Pi3. Flashing once

per second means that the battery voltage is at least 7.2v (the duty cycle will be scaled from 10% at 7.2v to 90% at 9v). Flashing twice per second means that the battery voltage is below 7.2v. Flashing four times per second means that the battery voltage is below 6v, or the battery voltage is below 6.8v and Raspberry Pi voltage below 4.85v (at which point the motors are automatically disabled). Flashing rapidly means that the Brick Pi3 firmware has encountered an error. This is most likely caused by the Brick Pi 3 not being properly connected to the Raspberry Pi.

Tool Chain - Static Binding

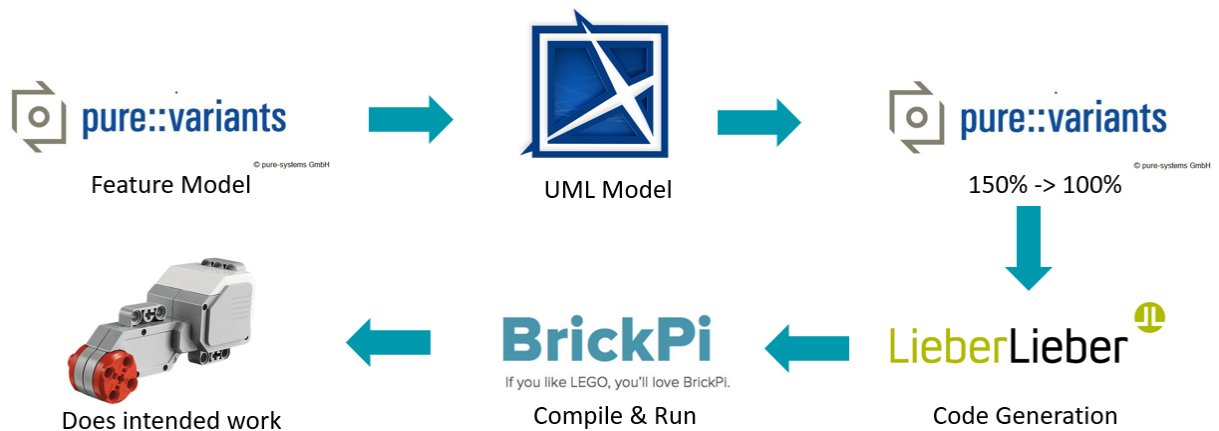



Figure 22 - Toolchain for Duck Maker (Static Binding)

In order to manage variability within the 32 different types of ducks, variability management principles are followed and the above tool chain is proposed. Variability management starts with commonality identification.

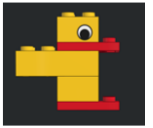
Commonality Identification:

All the 32 different types of ducks are examined and we find out there are a lot of similarities in the design of 32 different types of ducks. Even though they look different, all 32 different models of ducks are of 4 styles and has two options for legs and neck with yellow and red colored BRICK PI bricks. Even the duck has an optional hat which can be selected from three different colors namely white, blue and green.


- Styles :




Style1





Style2



Style3



Style4
- Legs & Neck



- Hats (Optional):








Figure 23 - Commonality identification

Variability Modelling

Based on the observations from commonality identification, feature model was modelled in pure::variants software and the constraints were defined to make the feature model robust.

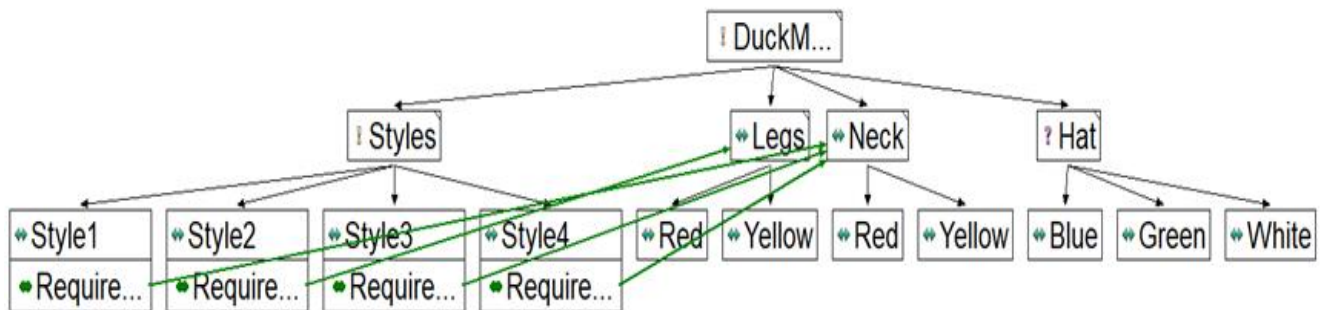


Figure 24 - Variability modelling

Based on feature model, UML Model was drawn in MagicDraw software for assembling the duck and the variation points are used to integrate the feature model from pure::variants software to the UML Model.

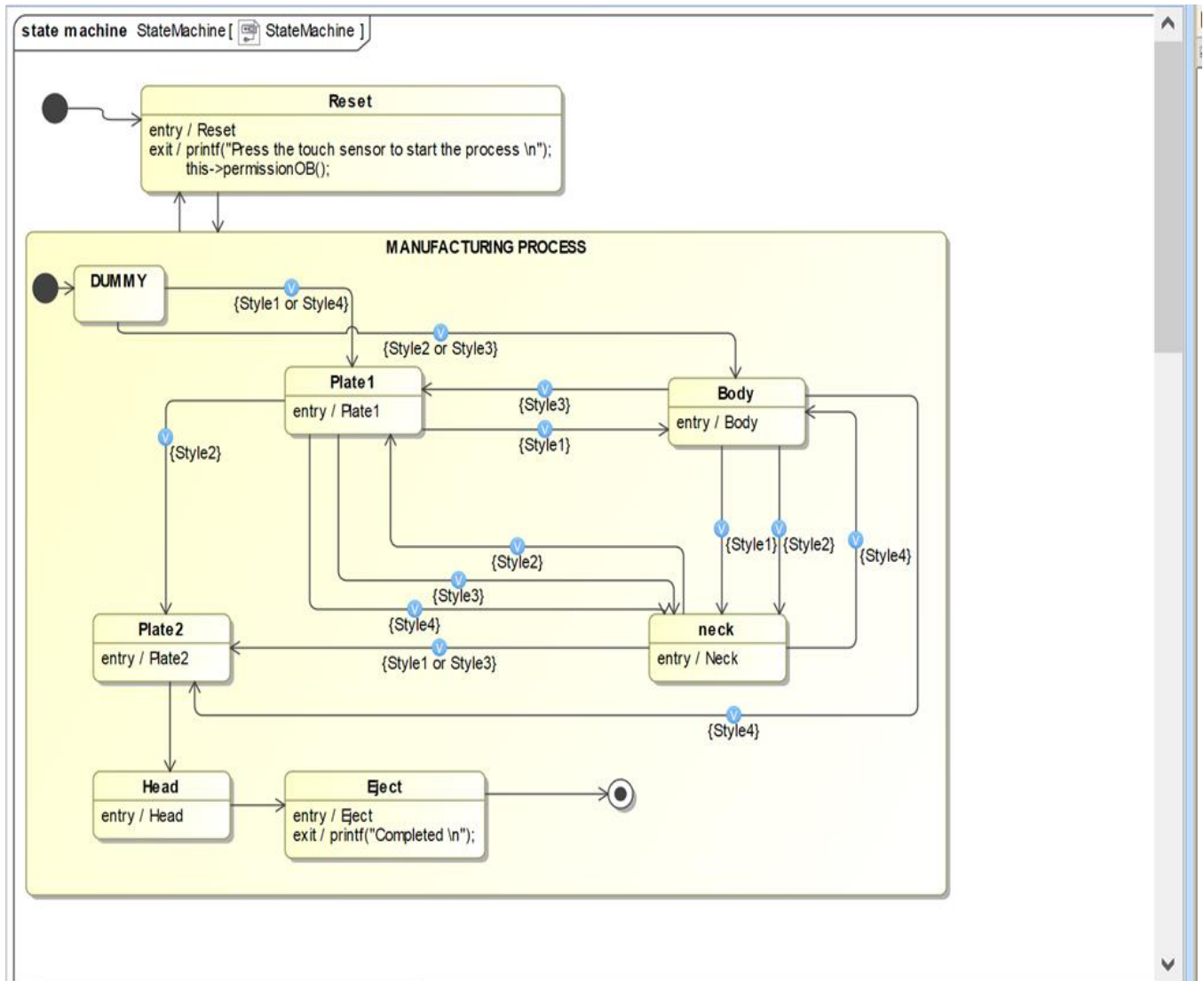


Figure 25 - Product family State machine diagram

Materialization

Now the customer can choose the desired features and style on the product variant duck and transform the model. Now the UML Model will be transformed as below, removing all the unwanted features and ready for code generation.

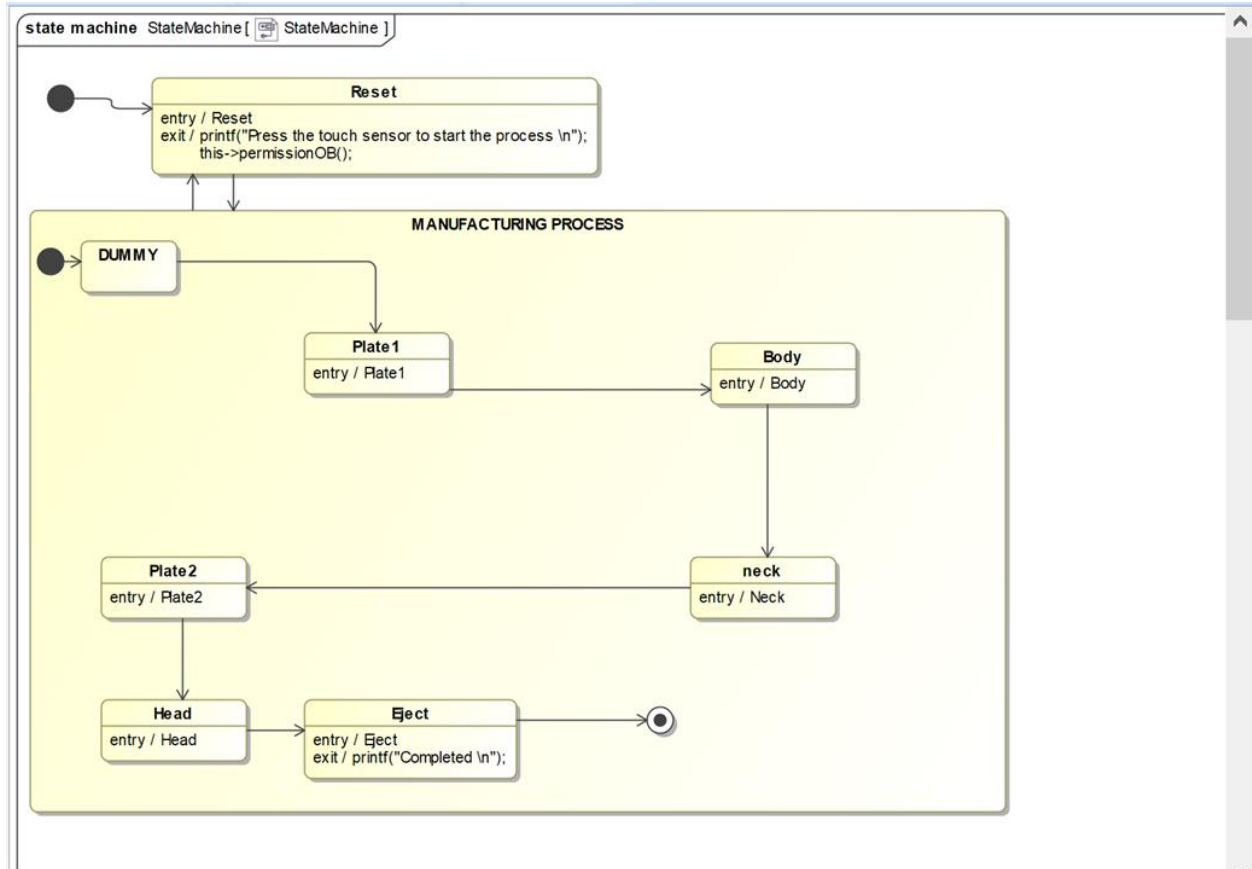


Figure 26 - Product variant State machine diagram

After transformation, code will be generated through Lieber Lieber Embedded Engineer plugin installed on MagicDraw and then the executable code will be loaded on to the Raspberry Pi and the generated code will be executed and run to control the BRICK PI assembly line to assemble the required duck with desired features.

This method manages the variability efficiently but there are some drawbacks since this method involves static binding. The code generated was loaded onto the Raspberry Pi all at once which

gives no control over the actions of the assembly line. Also there are two actions associated with variability management, which forces the user to interact with the interface two times and perform the actions on the software before moving to the assembly line.

Tool Chain - Dynamic Binding

A new tool chain was proposed addressing the limitations and errors in the previous tool chain. Binding time for this new tool chain is during Runtime. Coordinator was introduced to keep track of all the actions and to coordinate the assembly process. Instead of sending all the instructions at once, coordinator will send step by step instruction to the robots and machines during runtime. The flow of control step by step gives greater authority and control over the assembly tool chain. MagicDraw was removed from the tool chain since code generation was not that effective for this case study and also write application related syntax is taking more time and effort than writing the program manually. Considering all these factors a new tool chain for which binding happens during runtime is proposed as below:

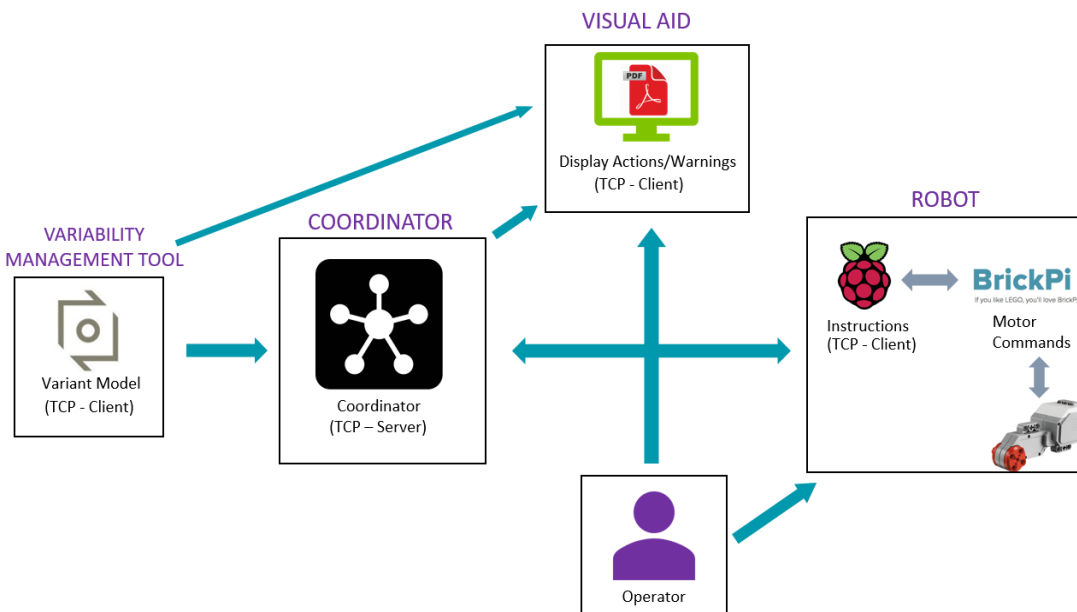


Figure 27 - Toolchain for Duck Maker (Dynamic Binding)

The communication between pure::variants, coordinator and robot is done through TCP/IP communication.

TCP/IP

TCP/IP is a set of protocols that enable communication between computers [61]. TCP is the most commonly used transport protocol on the internet because of its reliable transfer of data [62]. The Transmission Control Protocol is defined in RFC 793 [63], published in 1981, which gives a sign of how well this transport protocol is established. In this project TCP/IP communication protocols are used to interconnect network devices on the internet, e.g., coordinator, visual aid and robot are connected through TCP/ IP protocol.

WORKING PROCEDURE:

TCP/IP uses the client and server model of communication in which a machine (a client) is provided a service (like sending instructions) by another machine/ computer (a server) in the network. A single server can have multiple clients and server has the ability to communicate and coordinate all the clients.

In this case study, coordinator acts as a server and Display (Visual Aid) and Robot (BRICK PI Assembly line) acts as clients and the communication is carried in successfully assembling the required product variant. Server side code is written in python language due to its simplified programs to create a server. Executing coordinator starts server and opens a port for clients to connect and waits for clients (Display and Assembly line) to connect. Client side code is written in C++ and running the assembly line starts the client and connects to the server port and the communication is enabled.

Initially server side and client side code is written and then the feature model is designed in pure::variants and ready for user selection. A master document for operator instructions is written using pure::variants plug in MS Word and integrate the document with feature model.

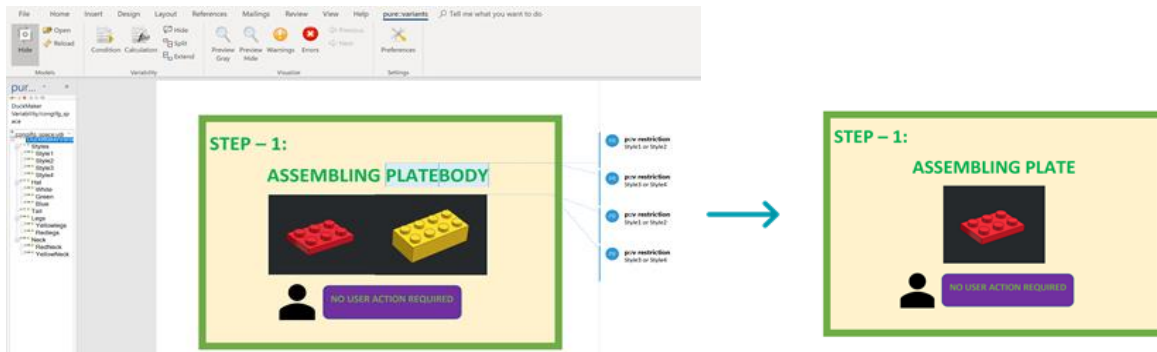


Figure 28 – Transformation of MS Word document

The above image shows how a master document is written. All the possible instructions required are written adjacently and the content is defined and integrated with the pure::variants feature model using comments. Right side image shows how the master document will transform to the required product variant operator instructions whenever respective style/ feature is selected.

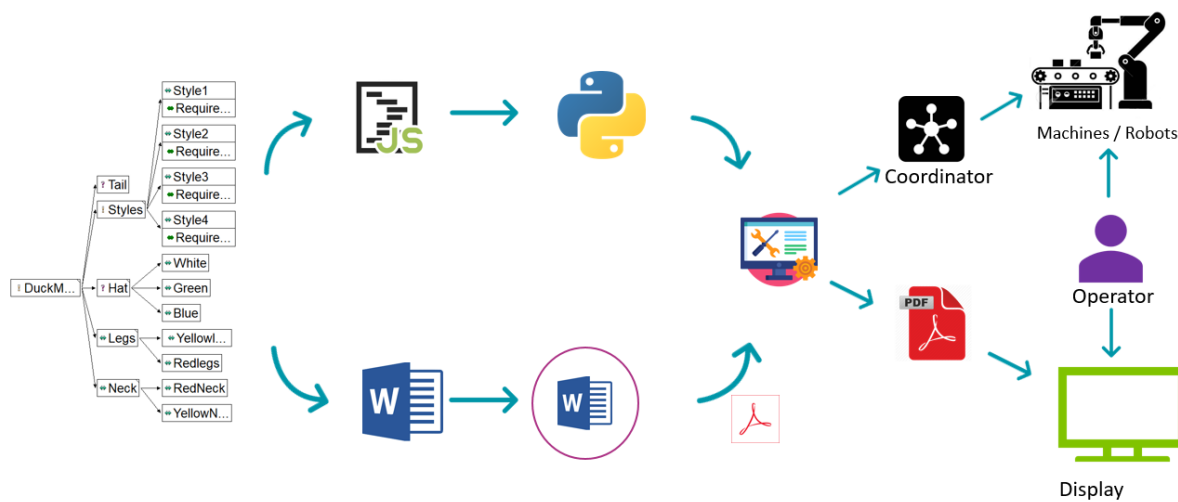


Figure 29 - Toolchain of DuckMaker

The above image shows the work flow in the toolchain, the process starts with user selecting one's desired features in the feature model and based on the selection JavaScript transformation module creates a python script for creating a client. Simultaneously, based on the selection MS Word master document is transformed to the selected variant instructions and are passed for displaying as operator instructions.

External program runner of pure::variants helps in automating the process and runs the python script for robot instructions, converts the word document into pdf document for operator instructions. Since the robot instructions will be sent one by one and of simple language (e.g., PANDP 150) can be interpreted by the robots easily with the help of json, a data interchange format. The whole process is automated and whenever the user needs to produce a new variant, all one has to do is select desired features and click transform model which transforms the model and proceeds to further actions in order to successfully assemble the product variant required. In this case study, variability is relatively easy and has to just add another feature to feature model, hence the tool chain efficiently managing the variability by significantly reducing the time of generating the product variants as wells as variability evolution.

Conclusions and Further work

Managing variability is the key for all companies, irrespective of size for satisfying the customer demands with unique set of demands. The use of SPLE techniques like feature modelling and variability management tools like pure::variants helped to create a tool chain that significantly reduced the development time and evolution time of new variants. In addition to managing the programs required for robots to perform the assembly actions, the proposed methodology has also addressed how semi-automatic assemblies can be supported by providing operator instruction manuals to aid the operators through visual aids and increase the productivity. Duck Maker, case study has demonstrated on how the proposed methodology can be implemented with the help of BRICK PI educational set and Raspberry Pi microcontroller. Complexity for modelling master instructional manual for aiding operators, increases based on number of product variants and variation points within the product family.

Although provided methodology presented a good solution to manage variability in assembly lines but modelling operator instructional manuals could be improved. More sophisticated methods should be investigated on how to model the master document for all the instructions required for the whole product family without much confusion. Even though pure::variants provided a platform to model the feature model and product derivation, its MS Word transformation module could be still improved.

References

1. Nils Boysen, Malte Flidner & Armin Scholl (2009) Production planning of mixed-model assembly lines: overview and extensions, *Production Planning & Control*, 20:5, 455-471, DOI: [10.1080/09537280903011626](https://doi.org/10.1080/09537280903011626)
2. Shtub, A., and Dar-El, E. M. "A Methodology for the Selection of Assembly Systems." *International Journal of Production Research* 27.1 (1989): 175–186. Web.
3. Kull, Hans. *Mass Customization Opportunities, Methods, and Challenges for Manufacturers*. Berkeley, CA: Apress, 2015. Web.
4. Joneja, Ajay, and Lee, Neville K.S. "Automated Configuration of Parametric Feeding Tools for Mass Customization." *Computers & Industrial Engineering* 35.3 (1998): 463–466. Web.
5. Tseng, Mitchell M., Jiao, Jianxin, and Merchant, M. Eugene. "Design for Mass Customization." *CIRP Annals - Manufacturing Technology* 45.1 (1996): 153–156. Web.
6. Suh, Nam P. *The Principles of Design*. New York: Oxford University Press, 1990. Print
7. Schmid, K, and Santana de Almeida, Eduardo. "Product Line Engineering." *IEEE Software* 30.4 (2013): 24–30. Web.
8. AlGeddawy, Tarek, and ElMaraghy, Hoda. "Design of Single Assembly Line for the Delayed Differentiation of Product Variants." *Flexible Services and Manufacturing Journal* 22.3 (2010): 163–182. Web.
9. Flores, Rick, Krueger, Charles, and Clements, Paul. "Mega-Scale Product Line Engineering at General Motors." *Proceedings of the 16th International Software Product Line Conference*. Vol. 1. ACM, 2012. 259–268. Web.

10. A product line of satellite ground control systems for the National Reconnaissance Office - Clements, P.; Northrop, L. *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2002.
11. A product line of weapons test ranges at the Naval Undersea Warfare Center - Cohen, S., Dunn, E., Soule, A. , *Successful Product Line Development and Sustainment: A DoD Case Study*, CMU/SEI2002-TN-018, September 2002.
12. A product line of helicopter avionics systems for the Army's Technical Applications Program Office - Clements, P. and Bergey, J. *The U.S. Army's Common Avionics Architecture System (CAAS) Product Line: A Case Study*, Technical Report CMU/SEI-2005-TR-019, September 2005.
13. A product line of submarine combat systems for the Navy's Submarine Warfare Federated Tactical System - Guertin, N., and Clements, P., "Comparing Acquisition Strategies: Open Architecture vs. Product Lines," *Proceedings of the 2010 Acquisition Research Symposium*, Monterey, May 2010
14. Mura, Michela Dalle, Dini, Gino, and Failli, Franco. "An Integrated Environment Based on Augmented Reality and Sensing Device for Manual Assembly Workstations." *Procedia CIRP* 41.C (2016): 340–345. Web.
15. Blackman, Harold S, Gertman, David I, and Boring, Ronald L. "Human Error Quantification Using Performance Shaping Factors in the SPAR-H Method." *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 52.21 (2008): 1733–1737. Web.

16. Michalos, George, Makris, Sotiris, and Chryssolouris, George. "The Effect of Job Rotation During Assembly on the Quality of Final Product." *CIRP Journal of Manufacturing Science and Technology* 6.3 (2013): 187–197. Web
17. Olsen, K. A., and Saetre, P. "Managing Product Variability by Virtual Products." *International Journal of Production Research* 35.8 (1997): 2093–2108. Web.
18. Bachmann, Felix., & Clements, Paul. 2005. *Variability in Software Product Lines* (Technical Report CMU/SEI-2005-TR-012). Pittsburgh: Software Engineering Institute, Carnegie Mellon University. <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7675>
19. Tomino, Takahiro, et al. "Market Flexible Customizing System (MFCS) of Japanese Vehicle Manufacturers: An Analysis of Toyota, Nissan and Mitsubishi." *International Journal of Production Economics*, vol. 118, no. 2, Elsevier B.V., 2009, pp. 375–86, doi:10.1016/j.ijpe.2008.12.002.
20. P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*: Addison-Wesley, 2002.
21. J. Bosch, *Design & Use of Software Architectures: Adopting and evolving a product-line approach*: Addison-Wesley, 2000
22. Berger, Thorsten. "Variability Modeling in the Wild." *Proceedings of the 16th International Software Product Line Conference*. Vol. 2. ACM, 2012. 233–241. Web.
23. Apel, Sven. et al. *Feature-Oriented Software Product Lines Concepts and Implementation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Web.

24. M. Anastasopoulos, A. Balogh, Model-driven development of particle system families, in: Proceedings of Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, Braga, Portugal, 2007, pp. 102–114
25. D. Beuche, H. Papajewski, W. Schröder-Preikschatb, Variability management with feature models, *Sci. Comput. Program* 53 (2004), 333–352
26. J. Coplien, D. Hoffmann and D. Weiss, "Commonality and Variability in Software Engineering," *IEEE Software*, vol. 15, no. 6, pp. 37-45, 1998
27. Sinnema, M., Deelstra, S., Nijhuis, J., & Bosch, J. (2004). Covamof: A framework for modeling variability in software product families. *Software product lines*, 25-27.
28. Capilla, Rafael., et al. *Systems and Software Variability Management Concepts, Tools and Experiences* . Springer Berlin Heidelberg, 2013, doi:10.1007/978-3-642-36583-6.
29. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility quality attributes and study. Technical report, CMU/SEI-90-TR21, November 1990
30. Entwisle, Susan, Sita Ramakrishnan and Elizabeth Kendall. "Model-Driven Exception Management Case Study." *Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalization*. IGI Global, 2010. 153-173. Web. 21 Mar. 2019. doi:10.4018/978-1-60566-731-7.ch012
31. Parejo, José & Sánchez, Ana B. & Segura, Sergio & Ruiz-Cortés, Antonio & Lopez-Herrejon, Roberto & Egyed, Alexander. (2016). Multi-Objective Test Case Prioritization in Highly Configurable Systems: A Case Study. *Journal of Systems and Software*. 122. 10.1016/j.jss.2016.09.045.

32. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: A Feature-Oriented Reuse Method with Domain Specific Reference Architectures. *Ann. Soft. Eng.* 5, 143–168 (1998)
33. K. K. C. Lee, K., "Usage context as key driver for feature selection", *LNCS Proc. of the 14th SPLC*, pp. 32-46, 2010.
34. Czarnecki, K., Eisenecker, U.W.: Generative programming: methods, tools, and applications. Addison-Wesley, USA (2000)
35. Damiani, Ferruccio, Schaefer, Ina, and Winkelmann, Tim. "Delta-Oriented Multi Software Product Lines." *Proceedings of the 18th International Software Product Line Conference*. Vol. 1. ACM, 2014. 232–236. Web.
36. Rafael Lotufo, Steven She, Thorsten Berger, Krzysztof Czarnecki, and Andrzej Wąsowski. 2010. Evolution of the linux kernel variability model. In *Proceedings of the 14th international conference on Software product lines: going beyond (SPLC'10)*, Jan Bosch and Jaejoon Lee (Eds.). Springer-Verlag, Berlin, Heidelberg, 136-150.
37. Pure Variants Homepage. (n.d.). Retrieved from <http://www.pure-systems.com/pv>
38. *Pure::variants*. 2013, pp. 173–82.
39. Pure Systems Manuals. (n.d.). Retrieved from <https://www.phaedsys.com/principals/puresystems/index.html>
40. pure-systems GmbH. *pure::variants User's Guide* { Version 3.2.4 for pure::variants 3.2. }
41. pure-systems GmbH. *Variant Management with pure::variants*. Technical White Paper, 2004.
42. Products - Magic Draw. (n.d.). Retrieved from <https://www.nomagic.com/products/magicdraw>

43. T. Gardner, UML Modelling of *Automated Business Processes* with a Mapping to BPEL, First European Workshop on Object Orientation and Web Services (in conjunction with ECOOP'03), Darmstad, Germany, July 2003 (see also <http://www.ibm.com/developerworks/webservices/library/ws-uml2bpel/>)
44. User Guide MD18. (n.d.). Retrieved from [https://docs.nomagic.com/display/MD185/User Guide](https://docs.nomagic.com/display/MD185/User+Guide)
45. Embedded Engineer. (n.d.). Retrieved from <https://www.lieberlieber.com/embedded-engineer/>
46. Compiler Construction. (n.d.). Retrieved from <https://www.cs.fsu.edu/~engelen/courses/COP402003/notes5.html>
47. P. Gawthrop, E. W. , and McGookin, “Using BRICK PI in control education,” in Proc. 7th IFAC Symp. Adv. Control Educ., Madrid, Spain, Jun. 21–23, 2006, pp. 1230–1235.
48. D. Benedettelli, M. Casini, A. Garulli, A. Giannitrapani, and A. Vicino, “A BRICK PI Mindstorms experimental setup for multi-agent systems,” in Proc. ISIC, Jul. 2009, pp. 1230–1235.
49. A. Sanchez, E. Aranda-Bricaire, F. Jaimes, E. Hernandez, and A. Nava, “Synthesis of product-driven coordination controllers for a class of discrete-event manufacturing systems,” *Robot. Comput.-Integr. Manuf.*, vol. 26, pp. 361–369, 2010.
50. V. Chandra, Z. Huang, and R. Kumar, “Automated control synthesis for an assembly line using discrete event system control theory,” *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 33, no. 2, pp. 284–289, May 2003.

51. Sanchez, A, and Bucio, J. “Improving the Teaching of Discrete-Event Control Systems Using a BRICK PI Manufacturing Prototype.” *IEEE Transactions on Education* 55.3 (2012): 326–331. Web.
52. F. Cardoso, A. Serrador, T. Canas, Algorithms for road safety based on GPS and communications systems WAVE, *Proc. Technol.* 17 (2014) 640–649, <http://dx.doi.org/10.1016/j.protcy.2014.10.187>.
53. C. Underwood, S. Pellegrino, V.J. Lappas, C.P. Bridges, J. Baker, Using CubeSat/ micro-satellite technology to demonstrate the Autonomous Assembly of a Reconfigurable Space Telescope (AAReST), *Acta Astronautica* 114 (2015) 112– 122, <http://dx.doi.org/10.1016/j.actaastro.2015.04.008>.
54. G. Andria, F. Attivissimo, A. Di Nisio, A.M.L. Lanzolla, A. Pellegrino, Development of an automotive data acquisition platform for analysis of driving behavior, *Measurement* 93 (2016) 278–287, <http://dx.doi.org/10.1016/j.measurement.2016.07.035>.
55. J. Fletcher, W. Malalasekera, Development of a user-friendly, low-cost home energy monitoring and recording system, *Energy* 111 (2016) 32–46, <http://dx.doi.org/10.1016/j.energy.2016.05.027>.
56. M. Tivnan, R. Gurjar, D.E. Wolf, K. Vishwanath, High frequency sampling of TTL pulses on a Raspberry Pi for diffuse correlation spectroscopy applications, *Sensors* 15 (2015) 19709–19722, <http://dx.doi.org/10.3390/s150819709>.
57. M. Schmidt, D. Penner, A. Burkl, R. Stojanovic', T. Schümann, P. Beckerle, Implementation and evaluation of a low-cost and compact electrodermal activity measurement system, *Measurement* 92 (2016) 96–102, <http://dx.doi.org/10.1016/j.measurement.2016.06.007>.

58. Step-1 Assembly. (n.d.). Retrieved from <https://www.dexterindustries.com/BrickPi/BrickPi3-getting-started-step-1-assembly/>
59. Step-1 Assembly. (n.d.). Retrieved from <https://www.dexterindustries.com/BrickPi/BrickPi3-getting-started-step-2-connect-BrickPi/>
60. BrickPi3. (n.d.). Retrieved from <https://github.com/DexterInd/BrickPi3>
61. Blank, Andrew G.. TCP/IP Foundations, John Wiley & Sons, Incorporated, 2006. ProQuest Ebook Central, <https://ebookcentral.proquest.com/lib/mnsu/detail.action?docID=267310>.
62. Kershaw, Stephen, and Hughes-Jones, Richard. "A Study of Constant Bit-Rate Data Transfer over TCP/IP." *Future Generation Computer Systems*, vol. 26, no. 1, Elsevier B.V., 2010, pp. 128–34, doi:10.1016/j.future.2009.03.007.
63. J. Postel, Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFC 3168.